Journal of Industrial and Management Optimization

https://www.aimspress.com/journal/jimo

JIMO, 22(4): 1571–1608.
DOI:10.3934/jimo.2026058
Received: 22 October 2025
Revised: 15 February 2026
Accepted: 27 February 2026
Published: 05 March 2026

Research article

Modeling and analysis of the walking worker assembly line balancing problem considering worker-dependent task durations and walking times to increase production rate

Murat ŞAHİN*

Department of Industrial Engineering, Manisa Celal Bayar University, Şehit Prof. Dr. Ilhan Varank Campus, Manisa, Türkiye

* **Correspondence:** Email: sahin.murat@cbu.edu.tr; Tel: +90 236 201 22 14.

Abstract: The production efficiency of an assembly line depends on worker characteristics and operational constraints in manufacturing systems. In many practical assembly settings, workers are required to move between stations, making walking times a relevant component of total operational time. This study investigated assembly lines with walking workers, which were rarely examined simultaneously in existing studies, considering both walking times between workstations and worker-dependent task times to analyze their joint impact on line performance. To address this problem, it was formulated as a mixed-integer linear programming model and supported by a lower bound, a constructive heuristic, and a hybrid solution strategy integrating a genetic algorithm–based meta-heuristic to efficiently solve larger instances. Computational experiments were conducted using a benchmark test instances under different levels of task time variability and walking times between stations. The results demonstrated the effectiveness of the proposed method in terms of solution quality and computational efficiency. The hybrid meta-heuristic was able to obtain better-quality solutions compared with algorithms in the literature. The results also revealed how worker variability and walking times influenced the overall production rate. Higher variability in worker-dependent task durations can improve the production rate at a fixed average task time, whereas longer walking times reduce system efficiency. These findings offered practical implications for the design and balancing of assembly lines with walking workers.

Keywords: worker-dependent task time; worker qualification; walking worker assembly line; mixed-integer linear programming; lower bound; constructive heuristic
Mathematics Subject Classification: 90B35, 90C27

1. Introduction

Assembly lines (ALs) are among the most effective manufacturing systems for organizing and optimizing mass production. They structure the manufacturing process into a sequence of interdependent tasks, enabling a continuous and systematic flow of production, thereby enhancing overall productivity and reducing costs [1]. Producing high-quality and innovative products is crucial in these industries; therefore, companies prioritize research and development at the core of their operations [2]. In response to changing production requirements, companies have increasingly designed assembly lines with different configurations to improve flexibility and efficiency. While some lines are built for high-volume, repetitive production, others are designed to accommodate product variety or variable demand. In many practical assembly systems, workers are allowed to move between stations to perform tasks where they are the most productive. Such systems, generally referred to as assembly lines with walking workers, are common in practice and introduce additional challenges for task assignment and line balancing [3–5]. Besides worker mobility, another important practical characteristic of assembly systems is the variability in task processing times among workers. Differences in experience, skill level, and learning ability result in heterogeneous task completion times. In traditional assembly line balancing research, it is usually assumed that all workers perform each task within a fixed and identical duration; however, this assumption rarely reflects real industrial environments. Consequently, recent studies have started to deal with worker-dependent task durations, leading to more realistic problem formulations that explicitly account for individual worker capabilities [6–8].

Assembly line balancing problems (ALBPs) can be defined as the assignment of interdependent tasks to workers and workstations such that precedence, technological, and cycle-time constraints are satisfied while maintaining a balanced production flow. A well-balanced assembly line improves system performance by reducing idle times, minimizing cycle time, and enhancing overall resource utilization. Therefore, ALBPs play a crucial role in optimizing manufacturing performance and ensuring the efficient operation of production systems. Classical ALBP studies assume homogeneous worker characteristics; however, worker heterogeneity has motivated the development of worker-dependent balancing models, often referred to as assembly line worker assignment and balancing problems (ALWABPs), which consider differences in worker productivity [6, 8, 9].

Assembly lines with walking workers are typically observed in systems with relatively high cycle times, where allowing worker movement helps maintain productivity and achieve better workload balance [3, 4, 10–13]. Such mobility increases operational flexibility but also introduces additional complexity into the balancing process, as walking times and worker replacement must be simultaneously considered with task and worker assignments. This feature leads to a more realistic representation of industrial assembly systems, in which worker movements can significantly influence overall line performance.

Based on the existing literature, two main research streams can be identified: assembly line balancing problems with walking workers (WW-ALBPs) and ALWABPs. Research addressing walking workers remains relatively scarce and exhibits considerable variation in terms of station accessibility and the technical structure of the assembly lines. Accordingly, the literature review is divided into two subsections: WW-ALBPs and ALWABPs. Table 1 provides a detailed overview of recent studies in these areas, and the following sections review both research streams separately.

Table 1. Summary of related studies, ALWABP / WW-ALBP and variants.

Author	**Problem**	**LT**	**Aim**	**LWT**		**SM**				**TT**	**WT**
				W	F	MM	E	H	O	WD	WT
Miralles et al. [14]	ALWABP	SL	min C		✓	✓	✓	✓		✓	
Shewchuk [15]	WW-ALBP	UL	min SN	✓		✓		✓			
Blum & Miralles [16]	ALWABP	SL	min C		✓	✓	✓	✓		✓	
Moreira et al. [17]	ALWABP	SL	min C		✓			✓		✓	
Araujo et al. [18]	P/C-ALWABP	SL	min C		✓	✓		✓		✓	
Mutlu et al. [19]	ALWABP	SL	min C		✓	✓		✓		✓	
Borba & Ritt [20]	ALWABP	SL	min C		✓	✓	✓	✓		✓	
Vila & Pereira [21]	ALWABP	SL	min C, SN		✓		✓		✓	✓	
Moreira [7]	ALWABP	SL	min SN		✓	✓		✓		✓	
Moreira [22]	ALWIBP	SL	min SN		✓	✓		✓		✓	
Sungur & Yavuz [6]	ALBPHW	SL	min Cost		✓	✓				✓	
Polat et al. [9]	ALWABP	SL	min C		✓			✓		✓	
Zacharia & Nearchou [23]	ALWABP	SL	(a)		✓			✓		✓	
Oksuz et al. [24]	ALWABP	UL	LE		✓	✓		✓		✓	
Moreira et al. [25]	ALWIBP	SL	(b)		✓	✓		✓		✓	
Sikora et al. [4]	WW-ALBP	SL	min C	✓		✓	✓		✓		✓
Deepak et al. [26]	WW-ALBP	UL	min C	✓					✓		✓
Şahin & Kellegöz [3]	WW-ALBP	MML	min SN, WN	✓		✓		✓			
Akyol & Baykasoğlu [27]	Ergo-ALWABP	SL	(c)		✓			✓		✓	
Janardhanan et al. [28]	ALWABP	TL	min C		✓	✓		✓		✓	
Karas & Özçelik [29]	ALWARP	SL	(d)		✓	✓		✓		✓	
Campana et al. [8]	ALBPHW	SL	min Cost		✓	✓		✓		✓	
Liu et al. [12]	WW-ALBP-TW	SL	(e)	✓		✓		✓	✓	✓	
Petroodi et al. [30]	WW-ALBP	MML	min Cost	✓		✓			✓		
Şahin & Kellegöz [31]	WW-ALBP	MML	min SN, WN	✓		✓	✓				
Ebrahimi et al. [32]	WW-ALBP	MML	min C	✓		✓		✓	✓		✓
Şahin & Kellegöz [33]	WW-ALBP	MML	min SN, WN	✓		✓			✓		
Michels & Costa [34]	MM-ALWABP	SL	min C / other		✓	✓		✓	✓	✓	
Current Study	**WW-ALBP**	**SL**	**min C**	✓		✓		✓	✓	✓	✓

(a) Cycle time and smoothness index are tried to minimize;
(b) The aim is minimizing cycle time, and distributing as evenly as possible the heteregenous workers along the line;
(c) Minimizing the cycle time and ergonomic risk factors;
(d) Minimizing cycle time and the changing assignment to be done after the disruption;
(e) Mininimizing the risk-averse sum of the cycle time and employed temporary workers;
LT = line type (SL = straight assembly line; UL = U-shaped assembly line; TL = two-sided line; MML = multi-manned line);
LWT = line worker type (W = walking worker, F = fixed worker);
SM = solution method (MM = mathematical model, E = exact, H = heuristic, O = other);
Aims (LE = line efficiency; SN = number of workstations; WN = number of workers);
TT = task time attribute (WD = worker-dependent or not);
WT = walking time between stations (considered or not);
WW-ALBP = walking worker assembly line balancing problem; WW-ALBP-TW = WW-ALBP with temporary workers;
ALWABP = assembly line worker assignment and balancing problem; P-ALWABP = parallelization of workers in ALWABP;
C-ALWABP = collaboration of workers in ALWABP; ALWIBP = assembly line worker integration problem;
ALBPHW = Assembly line balancing problem with hierarchical workers; ALWARP = assembly line worker assignment and rebalancing problem.

1.1. Studies About WW-ALBPS

Early work on walking-worker lines examined how movement and waiting affect performance. Lassalle [5] showed that waiting times increase buffers and increase cycle times; Wang et al. [10] compared fixed-worker with walking-worker settings on straight lines and showed that mobility can reach target rates with fewer resources. Early investigations mainly examined the impact of worker movement on system throughput and waiting behavior, rather than integrated balancing decisions involving worker heterogeneity. In particular, simulation-based analyses by [5] and [10] highlighted how worker waiting behavior and station occupancy directly affect buffer formation, cycle time, and achievable production rates in walking worker assembly lines. Later studies considered different layouts and assignment strategies. On U-shaped lines, Shewchuk [15] focused on worker–machine assignment (with a mathematical model and a heuristic to avoid overlapping routes) to reduce the number of workers and improve labor utilization, while Sirovetnukul and Chutima [35] showed that explicitly accounting for walking times tends to increase required workers and proposed a hierarchical assignment supported by a mathematical model (MM). Al-Zuheri et al. [36] validated an MM under stochastic task times and later combined a genetic algorithm (GA) with MM to enhance productivity and ergonomics; Cevikcan [11] used a hierarchical approach to determine station counts and improve output. Explicitly considering walking distances and worker–machine interactions has been shown to significantly influence workload balance and staffing requirements, motivating the development of hierarchical and assignment-based solution approaches.

More recent contributions have extended existing solution approaches. Sikora et al. [4] studied traveling worker assembly line balancing problem with an MM targeting cycle time and walking distance; Deepak et al. [26] used simulation and showed mobility reduces waiting times and improves production rates; Şahin and Kellegöz [3] addressed multi-manned ALBP with walking workers, proposing a mixed integer linear programming (MILP) model and an improved electromagnetic field optimization algorithm. Liu et al. [12] modeled risk-averse ALWABP with uncertain availability of disabled workers (two-stage stochastic solved by GA), while Hashemi Petroodi et al. [30] used an MM for reconfigurable mixed-model lines to reduce total cost. Despite methodological advancements, these studies typically address either worker mobility or worker heterogeneity, and do not explicitly analyze their combined impact on cycle time.

Nowadays, Calzavara et al. [37] analyze mixed-model assembly systems that commonly adopt fixed-worker and walking-worker strategies to enhance flexibility and throughput. Their simulation-based analyses show that the choice between fixed-worker and walking-worker systems mainly depends on task-time variability, walking times, and the ratio between the number of workers and workstations, which jointly influence system throughput and appropriate application conditions. In aircraft assembly lines, combining walking workers with parallel workstations introduces complex task allocation and scheduling challenges, which are addressed using constraint programming and validated on industrial instances [38]. In related line balancing studies, disassembly line balancing problems have also been addressed in settings involving mobile workers and human–robot collaboration. For instance, Qin et al. [39] proposed an improved evolutionary algorithm for U-shaped disassembly lines with mobile workers, while Qin et al. [40] employed a reinforcement learning framework for circular disassembly lines with human-assisted robotic workstations. These studies further highlight the growing interest in modeling worker mobility and collaborative environments in line balancing problems.

Overall, WW-ALBP studies primarily aim to maximize production rate [4, 12, 26, 32] and minimize the number of workers or stations [3, 11, 33, 35]; cost reduction appears as a secondary goal [30]. Most studies analyze straight or U-shaped lines [4, 10, 15, 26]; methods are predominantly MMs and heuristics, with simulation used in a subset [26, 32]; exact approaches typically handle smaller instances [4, 31]. While significant developments have been achieved, a large portion of WW-ALBP studies focus on worker mobility under the assumption of homogeneous task processing times, thereby limiting their applicability to assembly systems with heterogeneous worker capabilities.

1.2. Studies About Assembly Line with Worker-Dependent Task Times

In the problem where worker qualifications are considered (i.e., task durations vary across workers), it is essential not only to assign tasks to workstations but also to determine which worker performs each task. Therefore, this problem is known as the ALWABP. Most ALWABP studies focus on straight assembly lines, while fewer have investigated U-shaped [24, 41] and two-sided configurations [28]. The majority of ALWABP studies emphasize worker heterogeneity in task processing times while assuming fixed worker locations, thereby abstracting away worker mobility considerations.

The ALWABP was first introduced by Miralles et al. [14], who formulated the problem to minimize cycle time through simultaneous task–worker assignment. Blum and Miralles [16] expanded this formulation using a beam search algorithm based on lower bounds. Later studies proposed alternative models and heuristics. Araujo et al. [18] incorporated worker collaboration and parallelization to enhance efficiency, while Borba and Ritt [20] developed an MM and a beam search based heuristic with new reduction rules. Ramezanian and Ezzatpanah [42] presented a dual objective model minimizing both cycle time and worker-related operating costs, emphasizing skill based variability and multi-model production. Within this line of research, foundational studies established the importance of simultaneously assigning tasks and workers in heterogeneous environments but did not consider worker movement between stations. Further studies extended the ALWABP to various line structures and objectives. Oksuz et al. [24] modeled the U-shaped ALWABP using a nonlinear formulation solved by artificial bee colony and GA. Moreira et al. [25] developed heuristics that integrate heterogeneous workers while maintaining productivity. Janardhanan et al. [28] tackled large-scale traveling ALWABPs using the migrating birds optimization algorithm. The suggested configurations introduce spatial aspects, while worker movement is typically constrained by predefined assignments. Subsequent research introduced ergonomic and resilience-based extensions. Akyol and Baykasoğlu [27] proposed the Ergo-ALWABP, incorporating ergonomic risks through a multi-rule constructive randomized search heuristic. These extensions improve the realism of ALWABPs but continue to assume stationary workers throughout the balancing process. Campana et al. [8] proposed a hierarchical worker qualification framework optimizing both cycle time and cost via mixed-integer programming and meta-heuristics. Karas and Özçelik [29] developed a MILP model with an artificial bee colony algorithm to reassign tasks and workers after disruptions. Michels and Costa [34] examined worker coordination in multi-manned lines with heterogeneous and homogeneous workers. Overall, most ALWABP studies aim to increase production rate [14, 16, 18, 20, 27, 28, 34, 42], while others target different objectives such as minimizing smoothness index [23], reducing reassignment after disruptions [29], or balancing heterogeneous worker distribution [25]. Additionally, many studies have focused on minimizing the number of stations [7, 22] or overall costs [6, 8]. In parallel, several ALWABP variants have emerged in the literature to address different operational considerations. These include the ALWIBP, which integrates disabled workers [25];

the Ergo-ALWABP, which accounts for ergonomic risk factors [27]; the ALWABP which models worker collaboration and parallelization (P-/C-ALWABP) [18]; and the hierarchical worker ALBP, which minimizes cost based on worker qualifications [6]. Building on these developments, Mao et al. [43] extended the ALWABP to a human–robot collaboration setting and proposed a MILP model along with an adaptive simulated annealing approach to minimize cycle time under worker heterogeneity. These extensions illustrate the growing effort to represent realistic workforce diversity in assembly lines.

Based on a review of the existing literature, a wide range of studies has investigated various configurations of ALWABP by incorporating heterogeneous worker capabilities and optimizing diverse objectives. However, the specific configuration that simultaneously considers worker-dependent task processing times within a walking worker assembly line framework has not been sufficiently investigated in the assembly line balancing literature. Recent studies have increasingly highlighted the importance of explicitly modeling walking times in assembly systems, demonstrating that worker movement and walking-related delays can substantially affect line performance, task sequencing, and system design decisions [32, 37, 44, 45]. In such environments, worker movement between stations constitutes an inherent part of task execution rather than a secondary effect. Accordingly, walking times are explicitly incorporated into the proposed formulation as a structural component to ensure feasibility and internal consistency when worker mobility is allowed. This setting reflects realistic assembly environments frequently encountered in practice and highlights a clear gap in the existing literature regarding the joint consideration of worker mobility and heterogeneous task processing times. Motivated by this gap, the present study develops an integrated assembly line balancing formulation that simultaneously accounts for worker movement and worker-dependent task completion times. The resulting problem, which also incorporates walking times between stations with the objective of minimizing cycle time, is denoted as WW-WALBP-II.

The main contribution of this study is to address the WW-WALBP-II, an assembly line balancing problem that integrates worker-dependent task durations with walking times between workstations within a unified framework. To model this environment realistically, a MILP formulation has been proposed. A lower-bound calculation method and a constructive heuristic are also suggested. The constructive heuristic is then enhanced by a GA and local search mechanisms to efficiently solve larger instances. The proposed meta-heuristic framework is evaluated against well-known, migrating bird optimization (MBO), particle swarm optimization (PSO) and simulated annealing (SA) algorithms. Furthermore, the study examines how key parameters, such as walking times and variability in worker-dependent task durations, affect the overall production rate. The analysis provides valuable insights into how operational flexibility and worker heterogeneity influence line efficiency.

The rest of the article is organized as follows. While Section 2 defines the problem in detail, Section 3 presents the suggested MILP. Section 4 discusses lower bound computation methods and Section 5 gives detail information about the constructive heuristic and GA based meta-heuristic. Section 6 presents experimental studies and sensitivity analysis. Lastly, Section 7 concludes the paper and discusses future work.

2. Problem Definition

The WW-WALBP-II assigns a set of tasks $i \in I$ (where I is the set of all tasks) to workers $k \in K$ (with K as the set of workers) and workstations $j \in J$ (where J is the set of workstations). It schedules these tasks according to precedence relations (rules dictating the order in which tasks must be performed),

cycle-time limits (maximum allowable time per production cycle), and worker-load restrictions (limits on the workload assigned to each worker). The objective is to minimize the cycle time, consistent with the Type-II assembly line balancing problem formulation. Cycle time is the time needed to complete one full production cycle before another begins. It is equal to the maximum total workload among all workers, where total workload consists of both the time spent processing assigned tasks and the time spent walking between workstations to perform these tasks. Ultimately, the worker with the highest combined task and walking time, also called the bottleneck worker, determines the cycle time, making this metric the critical constraint for line performance.

Each task i has a worker-dependent processing time t_{ik}, which varies between workers due to differences in skill levels and efficiencies. Workers can move between stations but are limited by a maximum station interval ψ. Walking times between stations influence task start times and, consequently, the total cycle time. At the beginning of each cycle, each worker starts at their first station. Regardless of their final station, they must return to their initial station before the next cycle starts. This ensures synchronization of the production process. The main assumptions of the handled problem are given below:

(1) Each task must be assigned to only one workstation and one worker.

(2) Once a task is started, it cannot be interrupted.

(3) The assembly line has a straight-line structure, and a single model is produced.

(4) Precedence constraints among tasks must be satisfied.

(5) Task durations are deterministic.

(6) At least one task must be assigned to each worker, in line with the cyclic work pattern assumed in the proposed formulation.

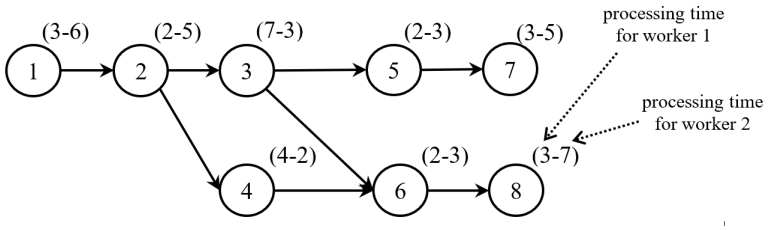
(7) Walking times between adjacent workstations are identical, reflecting a uniform workstation layout and equal inter-station distances commonly adopted in assembly line modeling. (e.g., walking time from station 1 to station 2 is the same as from station 2 to station 3).

(8) Walking times between workstations are independent of the worker, as walking paths and inter-station distances are determined by the physical layout of the line rather than individual worker characteristics.

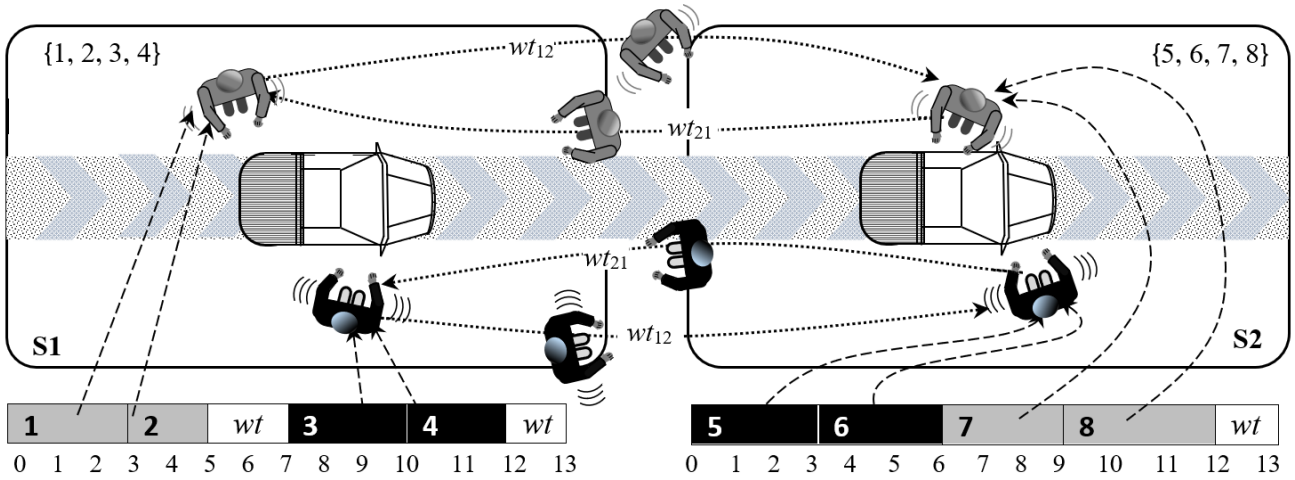
(9) Walking times between workstations have a symmetric structure, as workers follow the same physical path in both directions, resulting in identical travel times between station pairs.

(10) Only one worker can operate on a single semi-product, at any given time.

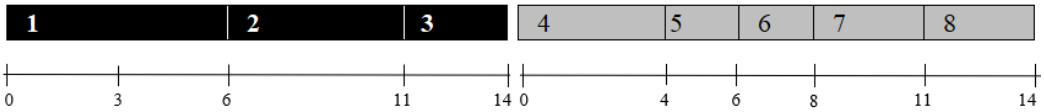
An illustrative example of the WW-WALBP-II is provided in Figure 1b, based on the precedence diagram in Figure 1a with 2 workers. In this figure, *task* 1 and *task* 2 at *station* 1, along with *task* 7 and *task* 8 at *station* 2, are assigned to *worker* 1. The remaining tasks (3, 4, 5, and 6) are assigned to *worker* 2. After *worker* 1 completes *task* 1 and *task* 2 at *station* 1, worker moves to *station* 2 to perform *task* 7 and *task* 8. The walking times from *station* 1 to *station* 2 are considered as 1 ($wt_{12} = wt_{21} = 1$). Once these tasks are completed, *worker* 1 returns to *station* 1 to start his first task of the next cycle before it begins. As illustrated in Figure 1b, *task* 3 does not start immediately after *task* 2 is completed. The reason is that *worker* 2, who is responsible for performing *task* 3, is working on a different task at *station* 2 during the time interval 5–6. The walking time from *station* 2 to *station* 1 during the time interval 6–7 also contributes to the delay. After the last task in a cycle is completed, the walking times to the stations where the workers need to be at the start of the next cycle are added.



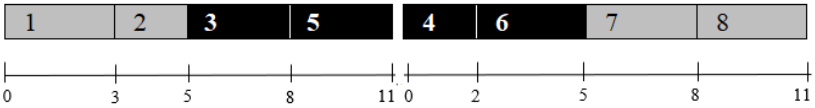
(a) Bowman precedence diagram with worker-dependent task times.



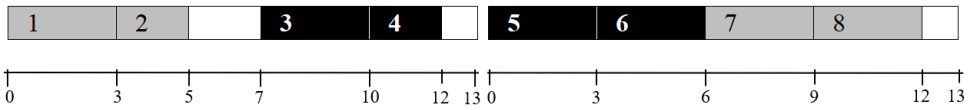
(b) Illustrating the optimal solution of WW-WALBP-II for the Bowman precedence diagram with 2 workers.



(c1) Optimal solution when the workers are not allowed walking between the stations.



(c2) Optimal solution when the workers are allowed walking among the work-stations with neglecting walking times



(c3) Optimal solution when the workers are allowed walking among the stations considering walking time.

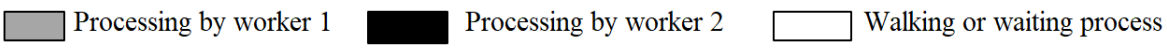


Figure 1. Illustration of the worker-dependent precedence diagram, the optimal WW-WALBP-II solution, and optimal solutions under different walking assumptions.

Since task times differ in ALWABP, assigning tasks to the fastest workers can boost efficiency and line performance. If workers are not assigned to the same workstation for tasks they can perform quickly, allowing them to move between stations can further enhance efficiency. However, ignoring walking times can result in impractical solutions. Figure 1c illustrates the optimal solutions for minimizing cycle time under three scenarios: (1) workers are not allowed to move between stations (Figure 1 c1); (2) workers are allowed to move between stations, but walking times are ignored (Figure 1 c2); and (3) workers are allowed to move between stations, with walking times of one unit considered (Figure 1 c3). These cases, based on the precedence diagram in Figure 1a, demonstrate how worker movement and walking times can affect production efficiency. Figure 1 shows that when workers are not allowed to move between stations, the cycle time is 14 units. Allowing worker movement and ignoring walking times reduces it to 11 units, though this assumption is unrealistic. When walking times of one unit are included, the cycle time increases to 13 units. Thus, allowing worker transition between workstations still improves production rate compared to fixed-worker lines.The following section presents the MILP model of the problem, which is developed under the specified assumptions above.

3. Mathematical Model

The WW-WALBP-II is formulated as a deterministic MILP, where all problem parameters such as walking times, worker dependent task durations and precedence constraints are assumed to be known and fixed. The formulation is structured through linear assignment, timing, and sequencing constraints that jointly ensure the feasibility of task execution and worker movements between stations. Within this framework, the model simultaneously assigns tasks to stations and workers while considering worker-dependent processing times and walking times between stations. It also determines the start times and the cyclic movement of workers. Each worker begins the cycle at the station of their first assigned task, and the walking time required to return from the last visited station to the start station is explicitly incorporated. In this formulation, the cyclic work pattern is explicitly represented by identifying a first and a last task for each worker. To maintain consistency with this modeling choice and to avoid degenerate cycle definitions, at least one task is assigned to every worker. All sets, parameters and decision variables used in the formulation are summarized in Table 2.

The objective function of the model is

$$minimize \quad c \tag{3.1}$$

Subject to

$$\sum_{j \in J} x_{ij} = 1 \qquad \forall i \in I \tag{3.2}$$

$$\sum_{k \in K} w_{ik} = 1 \qquad \forall i \in I \tag{3.3}$$

$$\sum_{k \in K} t_{ik} \cdot w_{ik} \leq ta_i \qquad \forall i \in I \tag{3.4}$$

$$0 \leq l_i \qquad \forall i \in I \tag{3.5}$$

Table 2. Sets, parameters, and decision variables used in the WW-WALBP-II model.

Notations	
Sets and Parameters	
m	Number of total workstations
n	Number of total tasks
wn	Number of total workers
I	Set of tasks $(1, 2, \ldots, n)$
J	Set of workstations $(1, 2, \ldots, m)$
K	Set of workers $(1, 2, \ldots, wn)$
F_i	Set of direct followers of task i
P_i	Set of direct predecessors of task i
F_i^*	Set of all followers of task i
P_i^*	Set of all predecessors of task i
t_{ik}	Duration time of task i when it is performed by worker k
wt_{jv}	The required time for walking from workstation j to v
ψ	The value of station range a worker can travel
$T_{sum}^{\max}$	Total duration of maximum times of tasks (maximum times of each worker)
M	Sufficiently big number
Binary variables	
x_{ij}	Binary variable with value 1 if task i is assigned to station j
w_{ik}	Binary variable with value 1 if task i is assigned to worker k
r_{ih}	Binary variable used to model the sequential execution of tasks i and h when both tasks are assigned to the same worker
rr_{ih}	Binary variable used to model the sequential execution of tasks i and h when they are assigned to different workstations but processed by the same worker
rrr_{ih}	Binary variable used to model the sequential execution of tasks i and h when both tasks are assigned to the same workstation
ts_{ih}	Binary variable used to model whether task i starts before task h
sw_{ih}	Binary variable indicating whether tasks i and h are assigned to the same workstation
swr_{ih}	Binary variable indicating whether tasks i and h are assigned to the same worker
pl_{ik}	Binary variable with value 1 if task i is performed as a last task by worker k
pf_{ik}	Binary variable with value 1 if task i is performed as a first task by worker k
Positive variables	
c	Cycle time
l_i	Positive variable with the interval 0 and c; shows the start time of task i in the workstation to which it is assigned
ta_i	Actual processing time of task i computed according to the assigned worker

$$l_i + ta_i \le c \qquad \forall i \in I \tag{3.6}$$

$$x_{ij} + x_{hj} \le 1 + sw_{ih} \qquad \forall j \in J,\ \forall i, h \in I,\ i \ne h \tag{3.7}$$

$$\sum_{j \in J} j \cdot x_{ij} - \sum_{j \in J} j \cdot x_{hj} \le M \cdot (1 - sw_{ih}) \qquad \forall i, h \in I,\ i \ne h \tag{3.8}$$

$$\sum_{j \in J} j \cdot x_{hj} - \sum_{j \in J} j \cdot x_{ij} \le M \cdot (1 - sw_{ih}) \qquad \forall i, h \in I,\ i \ne h \tag{3.9}$$

$$w_{ik} + w_{hk} \le 1 + swr_{ih} \qquad \forall k \in K,\ \forall i, h \in I,\ i \ne h \tag{3.10}$$

$$\sum_{k \in K} k \cdot w_{ik} - \sum_{k \in K} k \cdot w_{hk} \le wn \cdot (1 - swr_{ih}) \qquad \forall i, h \in I,\ i \ne h \tag{3.11}$$

$$\sum_{k \in K} k \cdot w_{hk} - \sum_{k \in K} k \cdot w_{ik} \le wn \cdot (1 - swr_{ih}) \qquad \forall i, h \in I,\ i \ne h \tag{3.12}$$

$$\sum_{j \in J} j \cdot x_{hj} \le \sum_{j \in J} j \cdot x_{ij} \qquad \forall i \in I,\ \forall h \in P_i \tag{3.13}$$

$$l_h + ta_h \le l_i + T_{\text{sum}}^{\max} \cdot (1 - sw_{ih}) \qquad \forall i \in I,\ \forall h \in P_i \tag{3.14}$$

$$ta_i \le l_h - l_i + T_{\text{sum}}^{\max} \cdot (2 - swr_{ih} - r_{ih}) \qquad \forall i, h \in I,\ i < h \tag{3.15}$$

$$ta_h \le l_i - l_h + T_{\text{sum}}^{\max} \cdot (1 - swr_{ih} + r_{ih}) \qquad \forall i, h \in I,\ i < h \tag{3.16}$$

$$ta_i \le l_h - l_i - wt_{jv} + T_{\text{sum}}^{\max} \cdot (4 - swr_{ih} - x_{ij} - x_{hv} + sw_{ih} - rr_{ih}) \qquad \forall i, h \in I,\ i < h;\ \forall j, v \in J,\ j \ne v \tag{3.17}$$

$$ta_h \le l_i - l_h - wt_{vj} + T_{\text{sum}}^{\max} \cdot (3 - swr_{ih} - x_{ij} - x_{hv} + sw_{ih} + rr_{ih}) \qquad \forall i, h \in I,\ i < h;\ \forall j, v \in J,\ j \ne v \tag{3.18}$$

$$ta_i \le l_h - l_i + T_{\text{sum}}^{\max} \cdot (2 - sw_{ih} - rrr_{ih}) \qquad \forall i, h \in I,\ i < h;\ \forall j \in J \tag{3.19}$$

$$ta_h \le l_i - l_h + T_{\text{sum}}^{\max} \cdot (1 - sw_{ih} + rrr_{ih}) \qquad \forall i, h \in I,\ i < h;\ \forall j \in J \tag{3.20}$$

$$\sum_{j\in J} j\cdot(x_{hj}-x_{ij})\le \psi + M\cdot(1-swr_{ih}) \qquad \forall i,h\in I \tag{3.21}$$

$$\sum_{j\in J} j\cdot(x_{ij}-x_{hj})\le \psi + M\cdot(1-swr_{ih}) \qquad \forall i,h\in I \tag{3.22}$$

$$l_h - l_i \le T_{\text{sum}}^{\max}\cdot ts_{ih} \qquad \forall i,h\in I,\ i\neq h \tag{3.23}$$

$$\sum_{i\in I} pl_{ik} = 1 \qquad \forall k\in K \tag{3.24}$$

$$\sum_{i\in I} pf_{ik} = 1 \qquad \forall k\in K \tag{3.25}$$

$$pl_{ik}\le pl_{hk} + (3-ts_{ih}-w_{ik}-w_{hk}) \qquad \forall k\in K,\ \forall i,h\in I,\ i\neq h \tag{3.26}$$

$$pf_{hk}\le pf_{ik} + (3-ts_{ih}-w_{ik}-w_{hk}) \qquad \forall k\in K,\ \forall i,h\in I,\ i\neq h \tag{3.27}$$

$$pl_{ik}\le w_{ik} \qquad \forall i\in I,\ \forall k\in K \tag{3.28}$$

$$pf_{ik}\le w_{ik} \qquad \forall i\in I,\ \forall k\in K \tag{3.29}$$

$$c\ge l_i + ta_i - T_{\text{sum}}^{\max}\cdot(4-pl_{ik}-pf_{hk}-x_{ij}-x_{hv}) + wt_{jv} \qquad \forall i,h\in I,\ \forall j,v\in J,\ \forall k\in K \tag{3.30}$$

$$sw_{ih},\ swr_{ih},\ ts_{ih},\ r_{ih},\ rr_{ih},\ rrr_{ih}\in\{0,1\} \qquad \forall i,h\in I \tag{3.31}$$

$$w_{ik},\ pl_{ik},\ pf_{ik}\in\{0,1\} \qquad \forall i\in I,\ \forall k\in K \tag{3.32}$$

$$x_{ij}\in\{0,1\} \qquad \forall i\in I,\ \forall j\in J \tag{3.33}$$

$$0\le l_i,\ 0\le ta_i \qquad \forall i\in I \tag{3.34}$$

$$0\le c \tag{3.35}$$

The objective (3.1) of the suggested MILP is to minimize the cycle time. Constraint set (3.2) assures that each task is assigned to only one workstation. Constraint set (3.3) ensures that exactly one worker performs each task. The actual processing time of a task cannot be lower than the time required by the assigned worker to complete it, as enforced by (3.4). According to (3.5), (3.6), the start and finish times of each task must be within the interval $[0, c]$. The constraint set (3.7)–(3.9) is satisfied by appropriate values of sw_{ih}. If tasks i and h are assigned to the same workstation, $sw_{ih} = 1$ by (3.7). Otherwise, when task i is assigned to a workstation with a higher index than task h, $sw_{ih} = 0$ to satisfy (3.8); similarly, if task i is assigned to a workstation with a lower index than task h, $sw_{ih} = 0$ to ensure (3.9). Constraint sets (3.10)–(3.12) regulate task assignments to workers via swr_{ih}. If tasks i and h are assigned to the same worker, then $swr_{ih} = 1$ by (3.10). Otherwise, when task i is assigned to a worker with a higher index than task h, $swr_{ih} = 0$ to satisfy (3.11), and when it is assigned to a worker with a lower index, $swr_{ih} = 0$ to satisfy (3.12). Constraint sets (3.13) and (3.14) enforce precedence relations. Precedence restrictions workstations basis is ensured by (3.13), while precedence for tasks assigned to the same workstation is ensured by (3.14). Tasks assigned to the same worker must be executed in sequence, which is guaranteed by (3.15)–(3.16). Thanks to the auxiliary variable r_{ih}, only one of these two constraints is active at a time, ensuring that either task h or task i starts earlier. If tasks i and h are assigned to the same worker but different workstations, either (3.17) or (3.18) becomes active. Accordingly, either the start time of task h is computed as the completion of task i plus the walking time wt_{jv}, or the start time of task i is computed as the completion of task h plus the walking time wt_{vj}. Tasks assigned to the same workstation must also be executed in sequence, which is imposed by (3.19)–(3.20). Via the auxiliary variable rrr_{ih}, only one of these constraints is active at a time, ensuring the correct order between tasks i and h. Constraints (3.21)–(3.22) ensure that the distance (in station indices) between any two tasks performed by the same worker does not exceed the walking range ψ. Constraint (3.23) assigns a consistent value to ts_{ih}: if task i starts before task h (i.e., $l_i \leq l_h$), then $ts_{ih} = 1$. Since at least one task is assigned to each worker, there must be exactly one first and one last task per worker, which is confirmed by (3.24)–(3.25). Constraint (3.26) ensures that the last task of each worker in a cycle has the latest start time among that worker's tasks, while (3.27) ensures that the first task has the earliest start time. According to (3.28)–(3.29), if task i is not assigned to worker k, then pl_{ik} and pf_{ik} cannot be equal to 1. When task i at workstation j is the last task of worker k and task h at workstation v is the first task of the same worker, constraint (3.30) becomes active. This constraint adds the required walking time wt_{jv} to the completion of the last task, i.e., $l_i + ta_i$. Finally, constraints (3.31)–(3.35) impose binary and sign restrictions. Taken together, constraints (3.1)–(3.35) define the MILP model of the WW-WALBP-II.

4. Lower Bound Calculation

One of the commonly used lower bound (LBC_1) is obtained by taking the greater of the average minimum task time per worker and the maximum value among the minimum task times, as shown in (4.1) [17].

$$LBC_1 = \left\lceil \max\left(t_{min}^{max}, \frac{T_{sum}^{min}}{wn} \right) \right\rceil \tag{4.1}$$

In Equation (4.1), T_{sum}^{min} denotes the sum of the smallest processing time of each task, wn is the number of workers, and t_{min}^{max} is the largest value among the minimum task times. LBC_1 can be tightened by introducing new approaches suitable for the WW-WALBP-II. The extended lower bound, LBC_2, can

be obtained by using only constraint (3.3) and (4.2) together with the objective function (3.1), besides the sign restrictions:

$$\sum_{i \in I} t_{ik} \cdot w_{ik} \leq \mathrm{LBC}_2 \qquad \forall k \in K \tag{4.2}$$

Although LBC_2 improves LBC_1, it still relies on repeatedly solving an MM, which becomes computationally infeasible for large-sized instances. An alternative lower-bound calculation method for the WW-WALBP-II is developed that does not rely on the MM approach. The procedure iteratively tests the feasibility of candidate lower bound values and increases the bound step by step. The resulting bound, denoted by LBC_3, together with the associated notation, is summarized in Table 3. In LBC_3, feasibility is checked by verifying whether the minimum total task load, increased by unavoidable required time (URT), can be processed by the available workforce under the given cycle time. URT captures the additional processing time arising when tasks cannot be executed at their shortest durations due to task-station assignment limitations and walking requirements. By incrementally adding URT, the lower bound becomes tighter for the considered assembly line configuration. Accordingly, $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$ denotes the idle time associated with worker k at station j, reflecting the unavoidable workload imbalance in the lower-bound calculation.

Table 3. Abbreviations and definitions for lower bound computation.

Abbreviation	**Definition**
WMT_k	Set of tasks completed in the shortest time by worker k.
PS_j^k	Tasks assignable to station j among those completed the fastest by worker k, given a lower bound cycle time and number of stations.
$\mathrm{IT}_k^{\mathrm{LBC}_3}$	Total required extra time for the tasks completed in the shortest time by worker k when the lower bound cycle time equals LBC_3.
$\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$	Total required extra time for the tasks completed in the shortest time by worker k when worker k is located at station j and the lower bound cycle time equals LBC_3.
$\mathrm{S}\theta_{jl}^k$	Unavoidable extra time for tasks that could be assigned to station l but not to station j, completed by worker k in the shortest time, when worker k is located at station j and does not walk to another station.
$\mathrm{wt}_j^{\min}$	Minimum walking time from any station j to any other station.
URT	Unavoidable required time assigned to the minimum total task load.

The general steps of calculating LBC_3 are outlined in Algorithm 1. This method follows the same logic as LBC_1. Given a lower bound value of the cycle time and a predefined number of stations, it is not always possible to assign all tasks to the fastest worker. In such cases, an unavoidable required time is added to the minimum total task load ($\mathrm{T}_{\mathrm{sum}}^{\min}$). Then, it is verified whether the total task load can be performed with the given number of workers and the considered lower bound of the cycle time. If this is not possible, the lower bound value is increased by one unit and the procedure is repeated until a feasible bound is obtained. At the end of this iterative process, a new and tighter lower bound, denoted as LBC_3, is obtained.

URT can be easily calculated once the minimum total required extra time for the tasks completed in the shortest time by worker k under the lower bound cycle time LBC_3, denoted by $\mathrm{IT}_k^{\mathrm{LBC}_3}$, is known.

Algorithm 1 Computing LBC_3

Input: LBC_1, $\mathrm{T}_{\mathrm{sum}}^{\min}$, $\mathrm{IT}_k^{\mathrm{LBC}_3}$, wn, problem information
Output: LBC_3

1: $\mathrm{LBC}_3 \leftarrow \mathrm{LBC}_1$
2: $\mathrm{URT} \leftarrow \sum_{k\in K} \mathrm{IT}_k^{\mathrm{LBC}_3}$ computed according to the current LBC_3
3: **while** $(\mathrm{T}_{\mathrm{sum}}^{\min} + \mathrm{URT}) > (wn \cdot \mathrm{LBC}_3)$ **do**
4: Increase the lower bound value by one unit: $\mathrm{LBC}_3 \leftarrow \mathrm{LBC}_3 + 1$
5: Update $\mathrm{URT} \leftarrow \sum_{k\in K} \mathrm{IT}_k^{\mathrm{LBC}_3}$ computed according to the current LBC_3
6: **end while**
7: **Output:** the computed LBC_3

The value of $\mathrm{IT}_k^{\mathrm{LBC}_3}$ can be derived from Equation (4.3) if $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$ is available:

$$\mathrm{IT}_k^{\mathrm{LBC}_3} = \min_{j\in J}\left(\mathrm{IT}_{kj}^{\mathrm{LBC}_3}\right) \tag{4.3}$$

Once the earliest (E_i) and latest (L_i) station intervals are computed, the set of tasks assignable to station j with minimum processing time for worker k (PS_j^k) can be easily determined, indicating the tasks eligible for assignment under the current lower bound cycle time.

The algorithm for computing the value of each $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$ is presented Algorithm 2. One of the assumptions of the considered problem is that each worker performs at least one task. It is evident that every worker must be located in at least one workstation. When worker k is at station j, there are two scenarios for the tasks that worker k performs in the shortest time and that are located at station l. In the first scenario, worker k goes to station l, either directly from station j or via a nearby station, to perform the required tasks. In the second scenario, worker k stays at station j and cannot complete the fast tasks assigned to station l. This situation results in an increase in the value of $\mathrm{T}_{\mathrm{sum}}^{\min}$ by $\mathrm{S}\theta_{jl}^k$. Here, $\mathrm{S}\theta_{jl}^k$ is the unavoidable extra time for tasks that can be assigned to station l but not to station j, which are completed in the shortest time by worker k ($i \in \mathrm{PS}_l^k \setminus \mathrm{PS}_j^k$). It can be calculated using Equation (4.4).

$$\mathrm{S}\theta_{jl}^k = \sum_{i\in(\mathrm{PS}_l^k \setminus \mathrm{PS}_j^k)} \left(t_i^{(2^{\mathrm{nd}}-\min)} - t_i^{\min}\right) \tag{4.4}$$

Let t_i^{min} be the minimum processing time for task i. $t_i^{(2^{nd}-min)}$ is the minimum time to process task i without the fastest worker. If at least two workers can finish a task in the shortest time, t_i^{min} and $t_i^{(2^{nd}-min)}$ will be the same. The sets PS_j^k are updated when the lower bound value is increased. Subsequently, the list of tasks that can be assigned to station j can be easily determined once the earliest and latest stations of each task are identified. By considering the minimum processing times of the tasks, the line can be assumed to have a simple assembly line structure for determining the earliest and latest station assignments, given a predetermined cycle time. The following Equations (4.5)–(4.6) are used to compute E_i and L_i values for each task [46].

$$E_i = \left\lceil \frac{t_i^{\min} + \sum\limits_{h\in P_i^*} t_h^{\min}}{\mathrm{LBC}_1} \right\rceil \tag{4.5}$$

$$L_i = m + 1 - \left\lceil \frac{t_i^{\min} + \sum\limits_{h \in F_i^*} t_h^{\min}}{\mathrm{LBC}_1} \right\rceil \tag{4.6}$$

Algorithm 2 Computing required extra time $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$ when worker k is located at station j

Input: k, j, LBC_3, WMT_k, PS_j^k, and problem information
Output: $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$

1: Initialize $\mathrm{IT}_{kj}^{\mathrm{LBC}_3} \leftarrow 0$; $\mathrm{WMT}_k \leftarrow \mathrm{WMT}_k \setminus \mathrm{PS}_j^k$; $J_1 \leftarrow J \setminus \{j\}$; $Iswalked \leftarrow$ **false**
2: **while** $\mathrm{WMT}_k \neq \emptyset$ **do**
3: Update $\mathrm{S}\theta_{jl}^k$ for each $l \in J_1$ according to the new task allocation
4: Find station l having the largest $\mathrm{S}\theta_{jl}^k$
5: **if** $\mathrm{S}\theta_{jl}^k > wt_l^{\min}$ **then**
6: $\mathrm{IT}_{kj}^{\mathrm{LBC}_3} \leftarrow \mathrm{IT}_{kj}^{\mathrm{LBC}_3} + wt_l^{\min}$
7: **if** $\mathrm{S}\theta_{jl}^k \geq wt_j^{\min} + wt_l^{\min}$ **then**
8: $Iswalked \leftarrow$ **true**
9: **end if**
10: **else**
11: $\mathrm{IT}_{kj}^{\mathrm{LBC}_3} \leftarrow \mathrm{IT}_{kj}^{\mathrm{LBC}_3} + \mathrm{S}\theta_{jl}^k$
12: **end if**
13: $\mathrm{WMT}_k \leftarrow \mathrm{WMT}_k \setminus \mathrm{PS}_l^k$
14: $J_1 \leftarrow J_1 \setminus \{l\}$
15: **end while**
16: **if** $Iswalked =$ **true then**
17: $\mathrm{IT}_{kj}^{\mathrm{LBC}_3} \leftarrow \mathrm{IT}_{kj}^{\mathrm{LBC}_3} + wt_j^{\min}$ *// add minimum return walking time to the initial station j*
18: **end if**
19: **Output:** the computed $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$

After computing the values of E_i and L_i for each task, the set of PS_j^k can be calculated easily. If a task $i \in \mathrm{WMT}_k$ and station j is within the interval $[E_i, L_i]$, then task i is inserted PS_j^k ($i \in \mathrm{PS}_j^k$). A task may belong to more than one possible station assignment set. Therefore, directly summing the values of $\mathrm{S}\theta_{jl}^k$ for each station when calculating $\mathrm{IT}_{kj}^{\mathrm{LBC}_3}$ would be misleading. To address this issue, the additional required time is calculated using a task-based approach. It focuses on the set of tasks completed in the shortest time by worker k (WMT_k) that are not assignable to worker k's current station j.

As seen in Algorithm 2 first, the tasks at worker k's current station are removed from WMT_k, as no extra time is needed for them. Then, the required sets and variables are arranged as shown in *line* 1 of Algorithm 2. The algorithm is repeated until WMT_k is empty. As seen in *line* 3, $\mathrm{S}\theta_{jl}^k$ for each $l \in J_1$ are recalculated, since changes in WMT_k affect these values. *Line* 4 selects the station l with the largest $\mathrm{S}\theta_{jl}^k$ among the remaining stations. The reason for choosing station l with the highest $\mathrm{S}\theta_{jl}^k$ is as follows. The set PS_l^k leads to the smallest increase in total workload when deciding if worker k should work at that station. Specifically, if $\mathrm{PS}_l^k \supset \mathrm{PS}_s^k$ and $\mathrm{S}\theta_{jl}^k > \mathrm{S}\theta_{js}^k$, then having *worker k* move to *station l* instead of station s minimizes the increase in total task load.

When walking times are considered, the minimum walking time to the relevant station ($wt_l^{\min}$) is considered, and walking times between adjacent stations are assumed to be equal. As shown in *line* 5, $S\theta_{jl}^k$ is compared with $wt_l^{\min}$ to determine whether *worker k* should move to *station l*. If *worker k*'s walking is sensible, the minimum walking time to *station l* is added as $IT_{kj}^{LBC_3} \leftarrow IT_{kj}^{LBC_3} + wt_l^{\min}$. Otherwise, $IT_{kj}^{LBC_3}$ is updated as $IT_{kj}^{LBC_3} \leftarrow IT_{kj}^{LBC_3} + S\theta_{jl}^k$.

When a worker moves to another station, they must return to their initial station until the new cycle starts. This requirement ensures the procedure's consistency. This return process is managed by the procedure in *lines* 7–8, where minimum return time is added to $IT_{kj}^{LBC_3}$ (see *lines* 16–17).

In summary, for each worker, tasks already assigned to their current station are removed from consideration. The unavoidable extra times for remaining tasks are computed iteratively, choosing stations that minimize total workload increase. Walking times are incorporated when moving between stations, and return times ensure consistency. This process is repeated until all tasks are accounted for, resulting in a tightened lower bound LBC_3. In the computational experiments, the maximum of LBC_1, LBC_2, and LBC_3 is used as the final lower bound to provide a tight reference value and to support the assessment of solution quality.

5. Solution Methodology

The proposed solution methodology first establishes a feasible solution using a constructive heuristic. This approach tries to build high-quality solution through a tailored encoding scheme that represents task–station and worker relations. To enhance search, the constructive procedure is hybridized with components of a meta-heuristic. GA operators are used to diversify the generated solutions. Compared to single solution-based meta-heuristics, population-based methods offer enhanced robustness and adaptability in complex scheduling problems. In particular, GAs have been shown to perform effectively in uncertain and dynamic scheduling environments due to their population-based search structure [47]. Accordingly, GAs are frequently adopted in assembly line balancing studies to effectively explore complex and highly combinatorial solution spaces involving assignment and sequencing decisions [9,48,49]. Moreover, the population-based structure of GAs allows the seamless integration of problem-specific constructive heuristics, local search operators, and regeneration mechanisms, which is particularly suitable for the proposed WW-WALBP-II.

5.1. Constructive Heuristic

Integrated encoding structures and priority-based representations have been widely employed in various optimization and scheduling problems [50–53]. The proposed heuristic constructs a solution from an integrated encoding scheme that represents both task order and station segmentation. The encoding structure includes $(n + m - 1)$ integer and (wn) continuous elements within the interval $[0, 1]$. The first $(n + m - 1)$ integers encode a feasible task sequence, with (-1) marking station boundaries. The next (wn) continuous elements specify worker priority for assignments. This representation is called the feasible sequence (FS). It preserves task precedence relations and also embeds station segmentation information. In an FS given in Figure 2 derived from the suggested precedence diagram (Figure 1a), *task* 1 and *task* 2 are assigned to *station* 1, whereas tasks (3–8) are assigned to *station* 2. Although this encoding provides precedence feasibility and station grouping, it cannot alone determine worker

assignments and start time of each task. Therefore, a constructive algorithm is required to decode the FS and generate a complete feasible configuration.

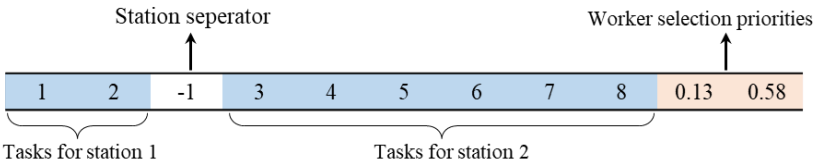


Figure 2. Illustrative example of the encoding scheme.

The developed heuristic converts each FS into a fully defined solution. It assigns tasks to workers and determines their execution order while minimizing the cycle time. Unlike standard ALBPs, in WW-WALBP-II, workers are not fixed to a single station. They can operate within a limited range of stations. Walking between stations introduces additional time and affects start times of tasks. The procedure must consider precedence, walking limits, walking times, and cycle-time constraints together.

The algorithm sequentially selects tasks from the FS. It identifies feasible worker–station pairs and schedules them to minimize the cycle time. This continues until all tasks are assigned. Standard heuristics for ALBPs do not work here. Integrating the FS encoding with the constructive procedure enables the proposed method to generate high-quality, feasible solutions for the complex WW-WALBP-II environment. The steps of the procedure are given in Algorithm 3.

In the first step of the algorithm, required sets and parameters are defined, and their values are determined (*line* 1). Steps in *lines* 2–33 are repeated until all tasks are assigned to a worker. Their start and finish times are detected. In the step located in *line* 3, the value of *srt* is calculated as the earliest start time. This is done by taking the minimum of the workers' availability times (wrt_k). If multiple workers have the same earliest start time (*srt*), the worker k with the highest worker priority (wp_k), which reflects the preference or suitability for assignment, is chosen to receive the task, as stated in *line* 4.

Then, the earliest start time (*earliesttime*) is determined by comparing the et_i values of the first task i in each S_j, where S_j denotes the set of tasks assigned to station j (*line* 5). Only the first tasks at each station are considered to satisfy precedence relationships, which means that tasks must be started in a specific order based on defined dependencies between them. By examining the first task at each station (*lines* 6–10), the algorithm checks whether there are multiple tasks with the same *earliesttime*; these tasks are then added to the first task set (TS_1), where TS_1 contains such earliest-start candidates. Then, in *lines* (11–15), it is checked whether the stations to which these tasks are assigned fall within the selected worker's walking range, and eligible tasks are included in the final candidate task list (TS_2).

Algorithm 3 Building a solution from the encoding schema of the Feasible Sequence (FS)

Input: Problem parameters and an FS
Output: Worker–task assignment (twa_i), start times (l_i), finish times (ft_i), and cycle time (c)

1: Generate required sets S_j and $WP = \{wp_k \mid k \in K\}$ from FS; initialize $WRT = \{wrt_k = 0 \mid k \in K\}$, $ET = \{et_i = 0 \mid i \in I\}$, $TS_1 = \emptyset$, $TS_2 = \emptyset$, $ES = \{es_k = 0 \mid k \in K\}$, $LS = \{ls_k = 0 \mid k \in K\}$, $CS = \{cs_k = 0 \mid k \in K\}$, $IS = \{is_k = 0 \mid k \in K\}$, and $U = K$.
2: **while** (*there is any task not yet assigned to a worker*) **do**
3: $srt \leftarrow \min_{l \in U}(wrt_l)$
4: Select worker $k \in K$ such that $wrt_k = srt$ and wp_k is maximized.
5: Determine the earliest ready time as $earliesttime \leftarrow \min_{j \in J} et_{S_j^1}$, where S_j^1 denotes the first task in the ordered set S_j.
6: **for** each $j \in J$ **do**
7: **if** $et_i = earliesttime$ where i is the first element of S_j **then**
8: $TS_1 \leftarrow TS_1 \cup \{i\}$
9: **end if**
10: **end for**
11: **for** each $i \in TS_1$ **do**
12: **if** $\max(ls_k, x_i) - \min(es_k, x_i) \leq \psi$ **then**
13: $TS_2 \leftarrow TS_2 \cup \{i\}$
14: **end if**
15: **end for**
16: **if** $TS_2 = \emptyset$ **then**
17: $U \leftarrow U \setminus \{k\}$;
18: **if** $U = \emptyset$ **then**
19: Return infeasible; **Break** while loop
20: **else**
21: $srt \leftarrow \min_{l \in U}(wrt_l)$; **go to line 4**
22: **end if**
23: **else**
24: Select task i for assigning to worker k with $mindifference = \min_{h \in TS_2}(t_{hk} - t_h^{min})$
25: $i \in \{h \mid (t_{hk} - t_h^{min}) = mindifference,\ h \in TS_2\}$
26: Select station j for $i \in S_j$
27: **end if**
28: **if** $srt = 0$ **then**
29: $l_i \leftarrow 0$; $is_k \leftarrow j$; $ft_i \leftarrow l_i + t_{ik}$; $wrt_k \leftarrow ft_i$
30: **else**
31: $l_i \leftarrow \max(earliesttime, srt + wt_{cs_k,j})$; $ft_i \leftarrow l_i + t_{ik}$; $wrt_k \leftarrow ft_i$
32: **end if**
33: Update $cs_k \leftarrow j$; $twa_i \leftarrow k$; $es_k \leftarrow \min(es_k, j)$; $ls_k \leftarrow \max(ls_k, j)$
34: $U \leftarrow K$; $S_j \leftarrow S_j \setminus \{i\}$
35: **if** $S_j \neq \emptyset$ **then**
36: $et_{S_j^1} = ft_i$
37: **end if**
38: **end while**
39: Compute $c = \max_{k \in K}(wrt_k + wt_{cs_k,is_k})$
40: Output twa_i, l_i, ft_i, and c

twa_i: index of the worker assigned to task i; l_i: start time of task i ($0 \leq l_i \leq c$); ft_i: finish time of task i; wt_{jl}: required walking time from *station* j to *station* l ; wrt_k: time when worker k is ready to start walking or performing a task; wp_k: priority value of worker k; S_j: set of tasks assigned to workstation j from FS; et_i: earliest ready time of task i; FS: feasible sequence with separators and worker assignment priority; is_k: first station of worker k; cs_k: current station of worker k; es_k: earliest station visited by worker k; ls_k: latest station visited by worker k; x_i: station index of task i.

If $TS_2 = \emptyset$, no task with the earliest time can be assigned to the worker; therefore, the worker is removed from the set U to prevent its selection in the next iteration. If the set U becomes empty, no feasible schedule exists for the given encoding. In this case, the encoding is penalized and the loop is terminated (*line* 16–19). Otherwise, the *srt* value is updated to select a different worker (*lines* 20–21). If there are tasks in TS_2, this means there are tasks that can be assigned to the selected worker within his/her walking range. The task i with the lowest $mindifference$ value is selected (*lines* 23–26). The $mindifference$ value represents how much the selected task's duration differs from the shortest processing time that could be achieved by this specific worker. A value of 0 means that the task is performed by the fastest possible worker. The station j, where task i is assigned, is determined, and the next step is proceeded with. It is checked whether the task assigned to the selected worker is the first task that the worker performs in the cycle (*line* 28). If so, the worker's first station is determined as $is_k \leftarrow j$, and without considering the walking times, the selected task's start and finish times and the selected worker's ready time are updated (*line* 29). If not, the walking time from the station where the worker was previously performing a task to the station j is added to the worker's ready time. Note that station j is where task i is assigned (*lines* 31). The start and finish times of task i are determined by considering the maximum of the task's ready time and the worker's ready time. Then, the current station, earliest, and latest stations for worker k are updated (*line* 33). The set U is reset to K, and task i is removed from S_j (*line* 34). The earliest start time of the first task in the set S_j is updated as ft_i, if S_j is not empty (lines 35–37). When the "while loop" ends, all tasks have been assigned to a worker, and their start and finish times have been determined. However, at the end of each cycle, each worker must walk from the last station to the station where the first task of the next cycle is performed. This situation is considered in the calculation of the cycle time, as shown in *line* 39. Finally, task–worker assignments, start times of the tasks, and the cycle time are provided as the output.

5.2. GA Components

Constructive heuristic is integrated into a GA framework combined with a local search procedure to further improve the solution quality and enhance exploration capability. The main steps of the procedure are given in Algorithm 4.

5.2.1. Initial Population, Selection, Crossover, and Mutation

The initial population is a set of candidate solutions that initiates the GA search, influencing both the diversity and quality of solutions. Population size, how many candidates are in the set, is one of the most influential parameters, affecting both exploration and solution quality. In this study, the initial population is generated randomly using the encoding scheme in *Section* 5.1, and its size is determined by preliminary experiments.

To achieve the goal of favoring high-quality individuals while maintaining sufficient diversity to avoid premature convergence, this study employs a binary tournament selection mechanism, in which two individuals are chosen at random and the better one is selected for reproduction. In the proposed GA framework, crossover is applied with a probability of 1.0, meaning that every selected parent pair undergoes the crossover operation. Although crossover is always applied, an elitist strategy is employed to ensure that the best individual is preserved between generations. This elitist strategy prevents the loss of high-quality solutions and supports steady convergence of the search process.

Algorithm 4 Genetic Algorithm (GA)

Input: Problem information
Output: Best solution and corresponding cycle time

$n \leftarrow 0$; Generate initial population (Pop_n) randomly; $CycleBest \leftarrow maxvalue$, *best solution*
while (termination criteria is not met) **do**
  Generate new population Pop_{n+1} from Pop_n by applying selection, crossover, and mutation operators
  Evaluate and find the *best chromosome*
  if $rnd(0–1) < P_{\text{local1}}$ **then**
    Apply local search I to the *best chromosome*
  end if
  if $rnd(0–1) < P_{\text{local2}}$ **then**
    Apply local search II to the *best chromosome*
  end if
  if obj(*best chromosome*) $< CycleBest$ **then**
    $CycleBest \leftarrow obj$(*best chromosome*)
    Update *best solution* accordingly
  end if
  if (*regeneration condition is satisfied*) **then**
    Generate population Pop_{n+1} randomly
  end if
  $n \leftarrow n + 1$
end while
Output: $CycleBest$ and *best solution*

The partially matched crossover operator proposed by [54] has been applied in the crossover method with minor modifications to ensure valid offspring. Figure 3 illustrates the crossover process for the considered problem. For obtaining offspring 1, a crossover point is selected, genes from position 0 to this point (including worker priority values) are copied from *parent* 1, and copied genes are then removed from *parent* 2 before adding the remaining genes sequentially from it. Therefore, precedence restrictions are not violated. The same procedure is repeated to obtain offspring 2.

The mutation operator introduces random changes into chromosomes to maintain diversity and prevent premature convergence. The operator swaps a gene with its immediate neighbor gene if the mutation rate ($P_{mutation}$) is met. This operation is applied only when no precedence relation exists between the selected tasks (genes).

5.2.2. Local Search Procedure

Local search is executed in every iteration and is designed to be computationally efficient. Its effectiveness is ensured by applying two different partial local search procedures to the best chromosome of the current population, each of which is applied with a certain probability. This approach further improves the best solution obtained in each generation.

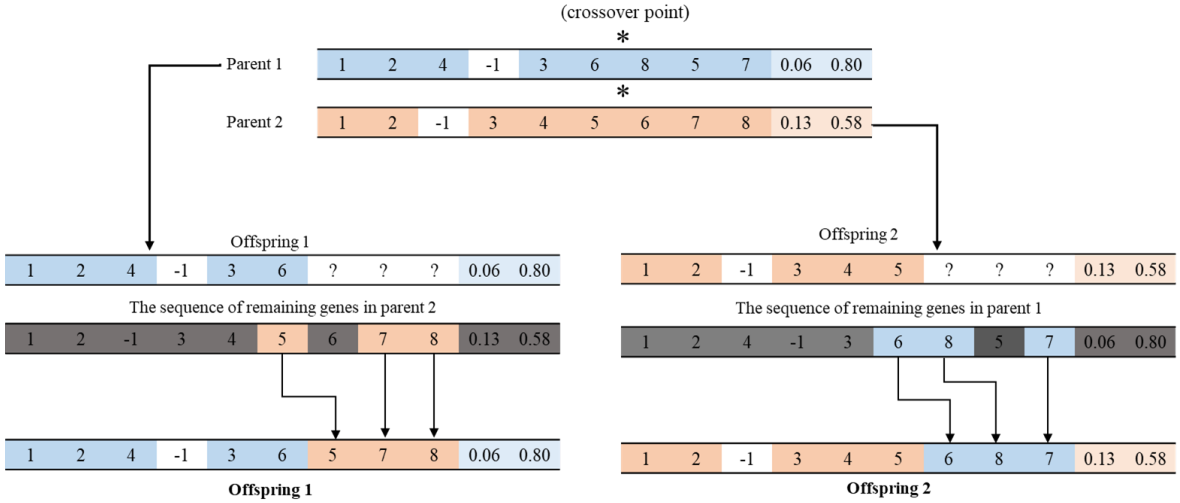


Figure 3. An illustrative of crossover operator.

5.2.3. Local Search I

One primary cause of high cycle time is task imbalance among stations. The first local search procedure aims to reduce the workload of the heaviest-loaded station, applied with a probability (P_{local1}). The station with the highest load is identified in the encoding using t^{min} values. To reduce its load while maintaining feasibility, a station boundary separator is swapped with a neighboring task to the left or right. Consider a chromosome represented as $(1, 2, \text{-}1, 3, 4, 5, 6, \text{-}1, 7, 8, 0.13, 0.58)$ for a problem with three stations, two workers, and the precedence diagram in Figure 1a. . The station workloads from t^{min} are 5, 9, and 6 for stations 1, 2, and 3. Since *station 2* (tasks 3, 4, 5, 6) has the highest workload, reducing it helps achieve a shorter cycle time. Neighbor solutions are generated by swapping *task* 3 and -1 or *task* 6 and -1. These solutions are then evaluated using the solution construction algorithm to check for objective function improvement. Although this procedure generates few neighbor solutions each time, it runs at every GA iteration, providing substantial cumulative opportunity for solution improvement. The algorithm steps are given in Algorithm 5.

5.2.4. Local Search II

The main limitation of local search I is its restricted neighborhood size. To overcome this limitation, local search II is proposed as an alternative. This procedure generates a considerably larger set of neighbor solutions and is applied to the best chromosome with a certain probability (P_{local2}). Specifically, in local search II, a pair of adjacent tasks in the encoding is selected, provided there is no precedence relationship between them. These two tasks are then swapped in their positions within the encoding to create a neighbor solution. The objective value of this new solution is evaluated using the constructive heuristic described in *Section* 5.1. The steps of the proposed procedure are summarized in Algorithm 6.

Swapping the positions of two adjacent tasks, or swapping a task with a separator, directly changes the order in which tasks are performed. This reordering affects when tasks begin and how workers are assigned. While this approach may not always yield improvements as significant as those from local search I, using it alongside the GA increases exploration of possible task orders and helps produce higher-quality solutions overall.

Algorithm 5 Local Search I

Input: Initial solution S, maximum number of consecutive non-improving iterations (NI_{max}), objective value of S $obj(S)$
Output: Improved solution S^*

$S^* \leftarrow S$
$c_{\text{best}} \leftarrow obj(S^*)$
$counter \leftarrow 0$
while *improvement observed in the last* NI_{max} *iterations* ($counter < NI_{max}$) **do**
  $S' \leftarrow$ generate a neighbor of S^* using a swap move, as described in Section 5.2.3
  $c' \leftarrow obj(S')$
  if $c' < c_{\text{best}}$ **then**
    $S^* \leftarrow S'$ *// accept improving solution*
    $c_{\text{best}} \leftarrow c'$ *// update* c_{best}
    $counter \leftarrow 0$ *// reset the counter of non-improving iterations*
  else
    $counter \leftarrow counter + 1$ *// increase the counter of non-improving iterations*
  end if
end while
return S^*

Algorithm 6 Local Search II

Input: Initial solution S, maximum number of iterations $Iter_{\max}$, the number of neighbors to explore in each iteration $Neighbor_{\text{number}}$, objective value of S $obj(S)$
Output: Improved solution S^*

$S^* \leftarrow S$
$c_{\text{best}} \leftarrow obj(S^*)$
$Iter \leftarrow 0$
while *the maximum iterations have not been reached* ($Iter < Iter_{\max}$) **do**
  $Iter \leftarrow Iter + 1$
  $counter \leftarrow 0$ *// counter for the neighbors explored in this iteration*
  for ($counter < Neighbor_{\text{number}}$) **do**
    $S' \leftarrow$ generate a neighbor of S^* using a swap move, as described in Section 5.2.4
    $c' \leftarrow obj(S')$
    if $c' < c_{\text{best}}$ **then**
      $S^* \leftarrow S'$
      $c_{\text{best}} \leftarrow c'$
      break the for loop
    end if
    $counter \leftarrow counter + 1$
  end for
end while
return S^*

5.2.5. Regeneration and Parameter Tuning

Toward the end of a GA, all chromosomes tend to converge and resemble the best chromosome found so far [55]. To prevent premature convergence, the degree of similarity between the population and the

best chromosome is periodically measured. If this similarity ratio exceeds a predefined threshold, the entire population is regenerated randomly. This mechanism effectively maintains population diversity and ensures exploration of new regions in the search space. The similarity ratio is calculated using Equation 5.1.

$$\mathrm{SR} = 100 \cdot \frac{\sum_{i=1}^{popsize} \sum_{j=1}^{(n+m-1)} sc_{ij}}{popsize \cdot (n+m-1)} \tag{5.1}$$

In this calculation, sc_{ij} is assigned a value of 1 when the j^{th} gene of chromosome i matches the j^{th} gene of the best chromosome; otherwise, it is assigned a value of 0. The similarity ratio (SR) is calculated every 20 iterations by summing sc_{ij} values across the population and dividing by the total number of genes. If SR exceeds the predetermined threshold value ($SR_{\text{threshold}}$), the entire current population is randomly regenerated. Parameter tuning was conducted via preliminary experiments for the suggested GA and the comparative meta-heuristics considered in this study, which are described in the following section. The tested and selected parameter values are presented in Table 4.

Table 4. Parameter tuning of the proposed and comparative algorithms.

Parameter	Algorithm	Tested values	Selected value
ps	GA, PSO	n; $2n$; $3n$; $4n$	$3n$
$SR_{threshold}$	GA	75; 80; 85; 90	80
$P_{mutation}$	GA	0.02; 0.05; 0.08; 0.10	0.05
P_{local1}	GA, PSO	0.70; 0.90; 1.00	0.90
P_{local2}	GA, PSO	0.70; 0.90; 1.00	0.90
NI_{max}	GA, PSO	$0.25wn$; $0.5wn$; wn	wn
$Iter_{max}$	GA, PSO	$3n$; $5n$; $7n$	$5n$
$Neighbor_{number}$	GA, PSO	n; $n/2$; $n/3$; $n/4$	$n/3$
B	MBO	11; 15; 19; 23	15
H	MBO	6; 10; 14; 18	10
q	MBO	6; 10; 14	10
x	MBO	2; 4; 6	6
$Cooling_{rate}$	SA	0.85; 0.90; 0.95; 0.98	0.95
$Temp_{init}$	SA	5000; 10000; 15000; 20000	10000
$NN_{each\,temp}$	SA	n; $2n$; $4n$; $6n$	$4n$

ps: population size; $SR_{\text{threshold}}$: similarity threshold for regeneration; P_{mutation}: mutation rate; P_{local1} and P_{local2}: probabilities of applying Local Search 1 and 2; $Iter_{\text{max}}$: maximum number of iteration for the local search; $Neighbor_{\text{number}}$: number of neighbor solutions per local search; $NN_{\text{each temp}}$: number of neighbor solutions per temperature level in SA; B: total number of birds (solutions) in the population; H: number of improvement steps applied to the leader bird and its following birds during each iteration of the MBO; q: number of neighboring birds generated in an iteration around the leader bird; x: number of the best remaining neighboring solutions shared with other birds. $Cooling_{\text{rate}}$: cooling rate in SA; $Temp_{\text{init}}$: initial temperature;

5.2.6. General Information About Comparative Algorithms

To evaluate the effectiveness of the proposed GA, its performance is compared with well-known meta-heuristic approaches frequently used in assembly line balancing problems, namely SA, PSO, and MBO. The considered problem includes characteristics that are not directly addressed by existing algorithms in the literature. Therefore, the PSO algorithm originally developed for the ALBP by [56] is adapted to the present problem by incorporating the proposed solution construction scheme and the local search procedures developed in this study. To ensure methodological consistency, the same solution construction framework is employed across all comparative algorithms. The SA algorithm proposed

by [55] is similarly adapted for comparative purposes. The MBO algorithm proposed by [57] is included as a comparative approach due to its competitive performance in assembly line balancing problems and its inherent improvement mechanism embedded within the search process. For detailed information about the comparative algorithms SA, PSO, and MBO, the reader is referred to, [55–57].

6. Experimental Study

The effectiveness of the proposed solution methods and the impact of key parameters, such as walking times and task variability, are evaluated using a benchmark dataset created in this study. The following subsection provides information about the dataset.

6.1. Dataset

A dataset of 99 test instances was generated by modifying 11 well-known precedence diagrams. These diagrams are commonly used in ALBP studies and most of them are publicly available at `https://assembly-line-balancing.de/`. Only precedence diagrams of Instances 13, 15, and 17 were specifically generated within the scope of this study, in order to examine how the MMs perform at these problem sizes. The number of tasks ranges from 8 to 58. Two main parameters directly influence solution quality: (i) variations in task processing times among workers can affect task allocation efficiency, and (ii) walking times between stations may increase overall completion times and disrupt workflow. Accordingly, three test scenarios were generated by varying the degree of task-time variation across workers. This design enables a direct assessment of the impact of worker-dependent task times on task assignments. In addition, three further scenarios were created by varying walking times between stations. These scenarios allow an evaluation of how worker mobility influences overall system performance. Table 5 summarizes the parameters used to characterize the test problems.

The first column of the table presents the average task processing times $t^{mean}_{average}$. It is calculated as the mean of all task completion times by all workers ($t^{\text{mean}}_{\text{average}} = \frac{1}{n}\sum_{i\in I}\bar{t}_i$). Here, $\bar{t}_i$ denotes the average processing time of task i across all workers.

Table 5. Characteristics of the generated test instances.

Precedence	ID	n	#	$t^{\text{mean}}_{\text{average}}$	σ_1	σ_2	σ_3	wt_1	wt_2	wt_3
Bowman	1	8	9	{9.44, 9.06, 8.81}	0.80	1.50	2.04	1.00	2.00	4.00
Mansoor	2	11	9	{12.05, 12.05, 12.14}	1.10	1.48	2.38	1.00	2.20	3.20
Jackson	3	11	9	{4.41, 4.00, 4.05}	0.45	0.64	0.96	1.00	2.00	3.00
Instance13	4	13	9	{9.62, 9.62, 9.62}	1.52	1.74	2.39	1.00	2.00	3.00
Instance15	5	15	9	{13.71, 13.71, 13.71}	3.08	4.61	5.36	1.33	2.33	3.67
Instance17	6	17	9	{12.59, 12.75, 12.75}	2.67	3.08	3.78	1.33	2.33	3.67
Roszieg	7	25	9	{5.55, 6.06, 6.99}	1.68	3.21	4.50	3.00	4.35	5.25
Buxey	8	29	9	{10.81, 10.91, 10.73}	1.08	2.31	4.31	2.10	3.00	7.00
Gunther	9	35	9	{13.46, 13.31, 13.43}	1.41	2.53	5.76	3.00	4.35	5.25
Kilbrid	10	45	9	{11.93, 11.79, 12.00}	1.35	2.60	5.75	2.47	3.67	5.30
Warnecke	11	58	9	{26.84, 27.19, 27.84}	3.38	4.96	5.71	3.14	4.33	5.93

$t^{mean}_{average}$: mean of task durations for all workers; σ_1–σ_3: standard deviations for three task–time variation levels; wt_1–wt_3: average walking times for three configurations.

The second parameter measures the degree of variation in task processing times among workers. It indicates the extent to which the processing time of each task deviates from its average value. The

standard deviation for each task i is computed using Equation (6.1):

$$\sigma_i = \sqrt{\frac{\sum_{k \in K} (\mathrm{t}_{ik} - \bar{\mathrm{t}}_{\mathrm{i}})^2}{|K|}} \tag{6.1}$$

In Equation 6.1, t_{ik} represents the processing time of task i by worker k, $\bar{\mathrm{t}}_{\mathrm{i}}$ is the mean processing time of task i, and $|K|$ is the total number of workers. Another important parameter is the average walking time among workstations, and this value can be computed using Equation 6.2

$$wt_{average}^{mean} = \frac{1}{|\mathrm{J}| \cdot (|\mathrm{J}| - 1)} \sum_{(j,l) \in \mathrm{J},\, j \neq l} wt_{jl} \tag{6.2}$$

Three different values were considered for the average walking time between stations. In the first case, walking times are low. In the second case, they are at a medium level. In the third case, walking times are high. Table 5 shows that the first precedence diagram (Bowman) includes eight tasks. Three different examples were created by varying the task durations. For these examples, the average task time ranges between 8.81 and 9.44. The average standard deviations of the task times are 0.80, 1.50, and 2.04. The average walking times are 1, 2, and 4, respectively. By combining the three levels of task times and walking times ($3 \cdot 3 = 9$), nine test cases were made for each precedence diagram.

6.2. Time Complexity and Convergence Behavior

The computational runtime of the proposed GA depends on the population size as well as the number of tasks, workers, and stations. Therefore, the convergence behavior should be examined in relation to the algorithm's computational complexity, since both are affected by problem parameters such as the number of tasks, workers, and stations. Each fitness evaluation focuses on the solution construction phase, during which tasks are assigned to workers and stations through iterative decision-making and feasibility checks subject to task precedence relations, station capacities, and worker-specific parameters. The computational effort of a single evaluation scales linearly with the number of tasks and workers. Since each evaluation is performed once for every solution in the population at each generation, the overall time complexity of the algorithm per run can be expressed, in asymptotic terms, as $O(g \cdot ps \cdot f(n))$, where g denotes the number of generations, ps is the population size, and $f(n)$ represents the computational cost of a single solution evaluation as a function of the problem size.

Figure 4 depicts the convergence behavior of the examined algorithms. The problems are based on the Gunther and Buxey precedence diagrams with varying the values of σ. Specifically, *Gunther*(5)(1)(1) represents a five-station problem (see Table 5) with low task-time variability ($\sigma = 1.41$) and low mean walking time ($wt_{average}^{mean} = 3$), while *Gunther*(5)(3)(1) corresponds to a high-variability case ($\sigma = 5.76$). The same notation and interpretation are adopted for the *Buxey* instances.

As shown in Figure 4, the GA, MBO, PSO, and SA algorithms quickly attain high-quality solutions during the initial stages of the search, followed by a more gradual refinement phase. This behavior indicates that the algorithms are able to efficiently exploit promising regions of the solution space early in the search. This early stabilization suggests strong computational efficiency and fast convergence in practice, even though each iteration may involve relatively high computational effort for GA.

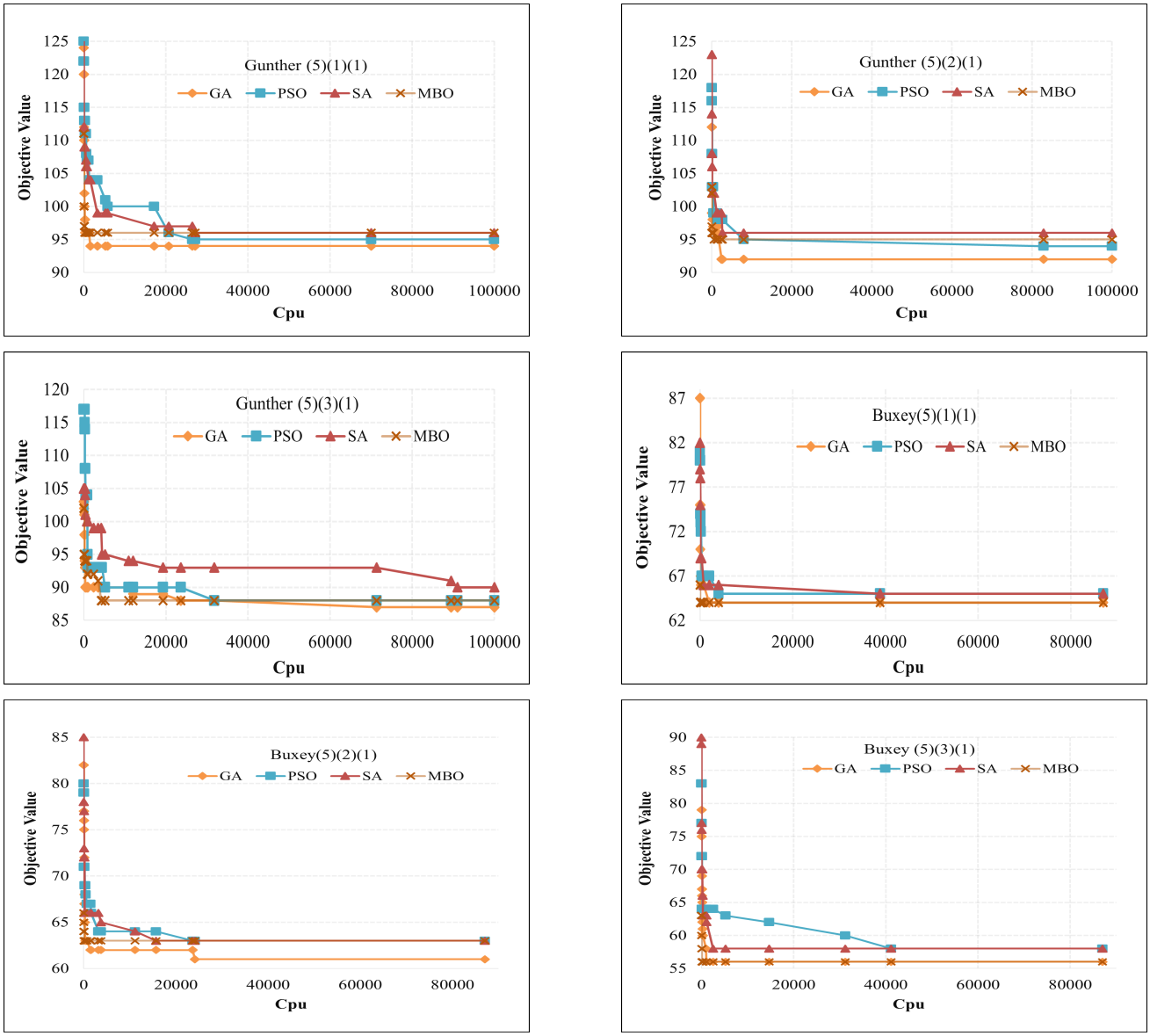


Figure 4. Convergence behavior of GA, PSO, SA, and MBO for Gunther and Buxey instances.

6.3. Computational Experiments

The mathematical models were solved using the *CPLEX 22.1* solver on a web-based high-performance computing platform, with a maximum computation time limit of 5400 seconds per instance. The meta-heuristic algorithms were implemented and executed on a local workstation equipped with an Intel i5 processor running at 1.3 GHz and 24 GB of RAM. Each meta-heuristic was allowed a runtime proportional to the problem size. Specifically, the time limit was set to a maximum of $3 \cdot n$ seconds, where n denotes the number of tasks in the problem instance. The meta-heuristic may terminate earlier if an optimal solution is reached, i.e., when the obtained solution matches the optimal result of the MM or attains the corresponding lower bound. Due to differences in computing environments and hardware configurations, runtime values are reported for descriptive purposes only and are not used for direct performance comparisons between the mathematical models and the meta-heuristic algorithms.

In contrast, runtime comparisons among the meta-heuristic algorithms are meaningful since they were executed under the same computational environment. To ensure the reliability and robustness of the obtained results, each test problem was solved five times.

6.3.1. Analyzing the Effect of Local Searches and Regenerating Strategy

This section examines how the integration of local search procedures and a regenerating strategy influence the performance of the proposed GA. The performances of the algorithms are evaluated using a set of test instances, and each test instance is solved 20 times with each algorithm. As is seen in Figure 5, the variations in algorithm performance are first examined using boxplot-based comparisons, while the statistical significance of the observed differences is assessed through nonparametric tests. The following algorithm variants are considered: the pure GA (GA), GA enhanced with Local Search I (GA–LI), GA enhanced with Local Search II (GA–LII), GA enhanced with both local searches (GA-LI-LII), and versions incorporating the regenerating strategy (GA–R and GA–R–LI–LII). The boxplot results indicate that local search operators generally improve solution quality compared to the baseline GA, although no clear dominance is observed between the two local search variants. Combining both local search operators yields only marginal additional improvements, whereas the regenerating strategy exhibits a stronger positive impact on solution quality. The best overall performance is consistently obtained by GA–R–LI–LII, particularly for the Gunther test instances.

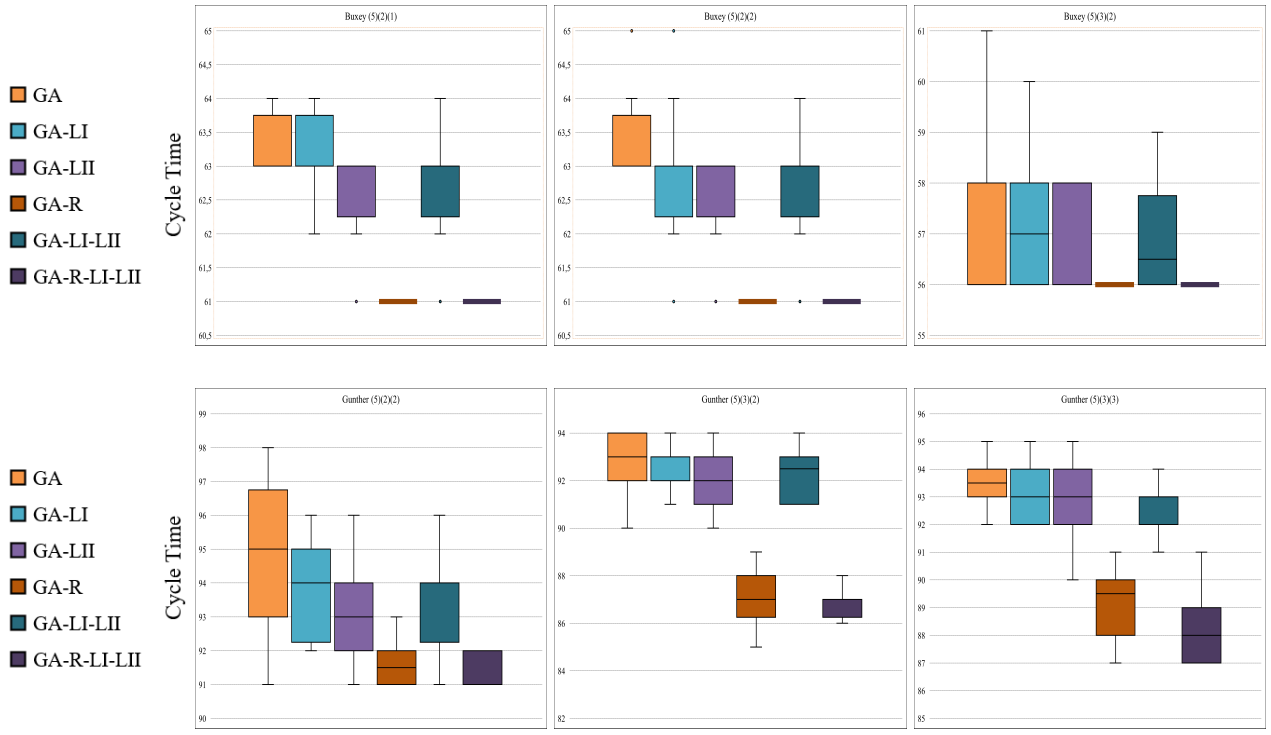


Figure 5. Comparative analysis of GA variants based on boxplot statistics

Since all algorithms were evaluated on the same benchmark instances and the normality assumption could not be ensured due to the limited sample size, nonparametric tests were employed. The Friedman test was first applied to detect overall performance differences among the algorithm variants, followed by

the Wilcoxon signed-rank test for selected pairwise comparisons. Comparisons between the baseline GA and its improved variants were performed to quantify the individual contributions of the regenerating strategy and the local search mechanisms. A direct comparison between GA–LI and GA–LII was additionally included to examine potential differences between the two local search operators. Then GA–LI and GA–LII were compared with GA–LI–LII to evaluate the effect of combining both operators. Finally, GA–R was compared with GA–R–LI–LII to assess whether integrating local search with regeneration leads to a statistically distinguishable improvement. The results summarized in Table 6 confirm that the improved variants significantly outperform the pure GA for the majority of the benchmark instances. Moreover, GA–LII shows a stronger dominance pattern than GA–LI, and regeneration provides a statistically supported performance gain when combined with local search.

Table 6. Wilcoxon signed-rank test results based on average performance.

Comparison	N	Z	p-value	Dominance pattern
GA – GA-R	6	−2.20	0.028	GA-R outperforms GA in all instances
GA – GA-LI	6	−1.99	0.046	GA-LI outperforms GA in 5 of 6 instances
GA – GA-LII	6	−2.20	0.028	GA-LII outperforms GA in all instances
GA-LI – GA-LII	6	−2.06	0.039	GA-LII outperforms GA-LI in 5 of 6 instances
GA-LI – GA-LI-LII	6	−2.21	0.027	GA-LI-LII outperforms GA-LI in all instances
GA-LII – GA-LI-LII	6	−0.63	0.528	No statistically significant difference
GA-R-LI-LII – GA-R	6	−1.46	0.144	GA-R-LI-LII outperforms GA-R in 3 instances (ties: 2; GA-R better: 1)

6.3.2. Analyzing the Suggested Solution Method

In this section, the comparative performance relationship between the proposed GA (uses both local searches and regenerating strategy), MBO, PSO, SA, and the MILP is evaluated. Initially, the capability of the MILPs to generate optimal or near-optimal solutions is assessed. This is followed by a comparison of the meta-heuristics in terms of solution quality. The comparison is performed by measuring the relative deviation of the solutions from the lower bound, calculated using Equation 6.3. The values of LBCs are derived as described in Section 4. If MILP identifies an optimal solution, the corresponding LBC is updated accordingly.

$$\%\text{gap} = 100 \cdot \frac{obj(\text{sol}) - \text{LBC}}{obj(\text{sol})} \tag{6.3}$$

Table 7 summarizes the effectiveness of suggested algorithms. For each problem, the table reports the number of test instances. Then, it shows the average percentage gap between the obtained solutions and the corresponding LBC values (%gap), as well as the average meta-heuristic CPU time required for computation. The column labeled MILP presents the results from the mathematical model. An asterisk (*) indicates cases where the solver failed to find a feasible solution within the time limit.

Each meta-heuristic was executed five times per instance. Their performance is reported in terms of both the mean and best results over these runs. The average mean %gap is the mean deviation; the best %gap is the minimum deviation over the five runs. Table 7 shows that the MILP produced effective solutions for WW-WALBP-II instances with up to 17 tasks. However, for larger instances (25 tasks or more), MILP frequently failed to produce feasible solutions within the 5400 seconds and occasionally experienced memory limitations. For solvable instances, the GA consistently achieved solutions with small deviation values relative to the MILP solutions, indicating its ability to approximate

the optimal solutions closely. This demonstrates its superior efficiency and robustness in addressing larger, more complex problems. GA has achieved solutions with deviations ranging from 8.73% to 17.80% on medium-sized instances (25 tasks or more). The small difference between the mean and best values also shows the robustness of the algorithm. MBO, PSO, and SA are also effective, but perform worse than GA. The MBO algorithm exhibits strong and stable performance for all instance sizes and consistently outperforms SA and PSO, while remaining slightly inferior to the GA, particularly for large-sized instances. The differences in performance are mainly related to the distinct search strategies employed by the algorithms, with MBO providing a robust structured exploration and the GA exhibiting greater adaptability in complex solution landscapes.

Table 7. Comparing the performances of the suggested solution methods

ID	n	#	MILP		SA			PSO			MBO			GA		
			avr. %gap	CPU	mean %gap	best %gap	CPU	mean %gap	best %gap	CPU	mean %gap	best %gap	CPU	mean %gap	best %gap	CPU
1	8	9	**0.00**	4	0.30	0.30	5	0.42	0.30	5	0.30	0.30	5	0.30	0.30	5
2	11	9	**0.00**	1487	0.30	0.17	3	1.08	0.51	5	**0.00**	**0.00**	4	**0.00**	**0.00**	4
3	11	9	**0.00**	160	**0.00**	**0.00**	<1	**0.00**	**0.00**	<1	**0.00**	**0.00**	<1	**0.00**	**0.00**	<1
4	13	9	**0.00**	1286	2.63	2.63	39	2.70	2.63	39	2.63	2.63	39	2.63	2.63	39
5	15	9	**1.93**	5258	2.93	2.93	30	3.72	2.93	33	2.96	2.93	32	2.96	2.93	32
6	17	9	**0.00**	2950	3.71	3.71	71	4.47	3.71	71	3.71	3.71	71	3.71	3.71	71
7	25	9	50.54	5400	23.13	21.21	75	22.37	19.95	75	23.13	20.46	75	18.50	**17.80**	75
8	29	9	54.26	5400	12.96	11.21	87	14.01	11.84	87	11.18	9.90	87	9.51	**9.24**	87
9	35	9	*	5400	14.75	13.31	105	14.26	12.39	105	14.56	12.97	105	9.25	**8.73**	105
10	45	9	*	5400	20.26	18.61	135	18.10	15.12	135	12.98	11.35	135	11.28	**10.12**	135
11	58	9	*	5400	17.87	16.00	174	15.26	13.44	174	10.36	9.64	174	10.08	**9.60**	174

* No feasible solution found within the given CPU time limit.
Bold values indicate the best performance among the all solution methods.
n: number of tasks; #: number of test instances; avr. %gap: average percentage gap; CPU: average computation time (s); mean %gap: mean deviation from lower bound across five runs; best %gap: best deviation from lower bound across five runs.

Table 8 shows the overall performance of MILP and the four meta-heuristics on 99 test instances. The results are presented in terms of the number of feasible solutions (#FS), number of optimal solutions (#OS), number of best solutions (#BS), number of unsolved instances (#NS), and average percentage gap (for the best solution among the five runs) from LBC (avr. %gap).

The MILP found 72 FS out of 99 instances. It achieved optimality in 52 of them, but failed to return a solution within the time limit in 27 test instances. All meta-heuristics produced FSs for all 99 instances. GA achieved the highest number of OSs (36) among the meta-heuristics, followed closely by SA (34), PSO and MBO (33). Based on the number of BS, the GA also demonstrates superior performance by achieving 80 BSs, whereas MILP and MBO exhibit nearly identical performance according to this metric. Consistently with this observation, the GA outperformed the other methods by yielding the lowest average percentage gap (5.91%). MBO also proves to be an effective approach, achieving the second-best performance with an average gap of 6.81%. These gap values are deviations from a known lower bound, which may not necessarily be optimal. In particular, for larger or more complex problem instances, the known lower bound can be quite distant from the true OS, so the reported gap values might overstate the lack of optimality.

Table 8. Overall comparison of the suggested solution methods.

Algorithm	# of instance	# of NS	# of FS	# of OS	# of BS	avr. %gap	avr. CPU (s)
MILP	99	27	72	**52**	53	13.35	3472.38
SA	99	**0**	**99**	34	39	8.11	67.22
PSO	99	**0**	**99**	33	40	7.51	69.15
MBO	99	**0**	**99**	33	52	6.81	68.15
GA	99	**0**	**99**	36	**80**	**5.91**	**66.18**

NS: no solution; FS: feasible solution; OS: optimal solution; BS: number of instances where the algorithm achieved the best-known solution.

6.4. Sensitivity Analysis

A sensitivity analysis was conducted among three test instances to evaluate the impact of increased walking times between stations ($wt_{average}^{mean}$) and varying levels of task duration variability (σ) on the production rate.

6.5. Analyzing the Effect of Task Time Variability and Walking Times

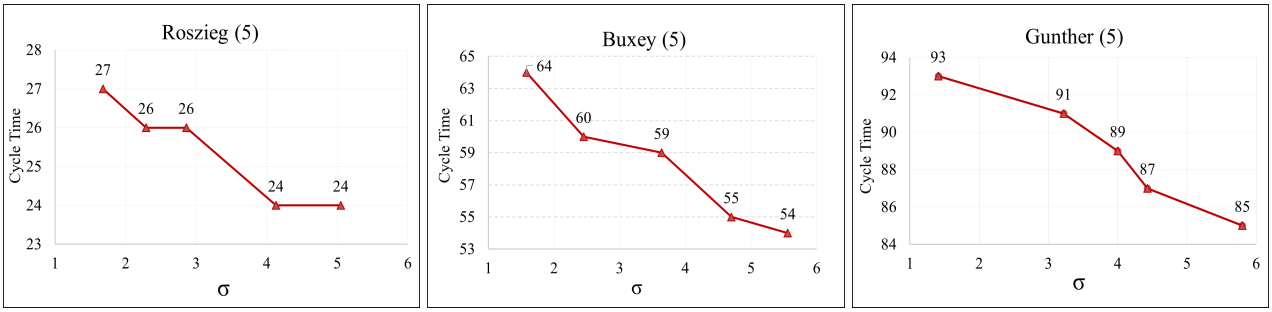
The sensitivity analysis of the three benchmark instances clearly shows that task duration variability (σ) is a key factor affecting production rate. Figure 6 shows the results of this analysis. In all cases, a moderate increase in σ leads to improved production rates. This improvement occurs because it enables more effective line balancing and better resource utilization. The analysis also shows that increasing walking time between workstations leads to longer cycle times, but only up to a certain threshold. As shown in the Figure 6 (particularly in the Buxey and Rozsieg examples), further increases in average walking time have no noticeable effect on production rates after the threshold. In summary, the key findings are that task duration variability can enhance production performance under some conditions, but achieving optimal results requires managing both task duration variability and walking time together.

Table 9 summarizes the results by degree of task time variability and presents walking time between stations, based on the test instances that yielded OSs. In this table, the first column lists the problem ID, and the second column indicates the number of instances solved for that problem group. The next three columns display the average cycle times for task variability levels 1, 2, and 3. The columns are presented in order. The final three columns show how changes in walking times affect the production rate. The walking times range from low to high.

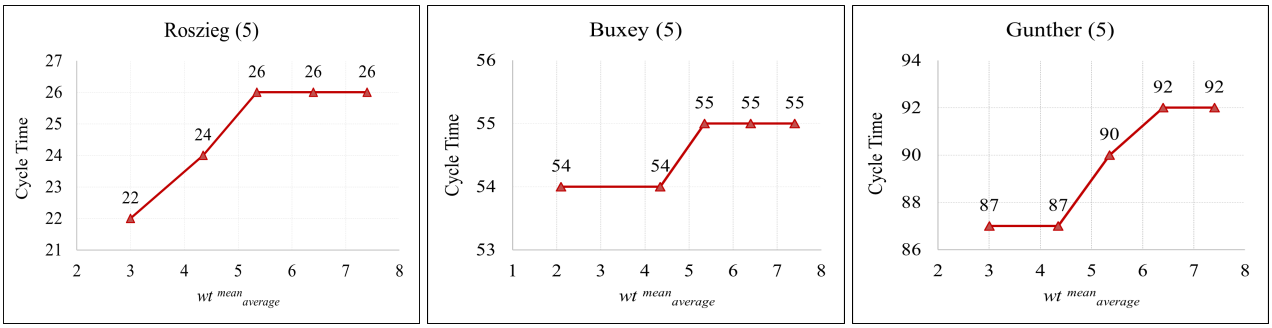
Table 9. Summary of the solutions according to degree of the variability of the task times and the increase of the walking times.

ID	#	Avr. Sol. with different σ			Avr. Sol. with different $wt_{average}^{mean}$		
		1	**2**	**3**	**1**	**2**	**3**
1	3	37.00	36.67	**35.00**	**36.00**	36.33	36.33
2	3	**65.00**	**65.00**	**65.00**	**65.00**	**65.00**	**65.00**
3	3	24.00	**22.00**	**22.00**	22.67	22.67	22.67
4	3	61.33	61.67	**60.67**	**59.67**	61.67	62.33
5	3	66.67	61.67	**59.67**	**61.33**	62.33	63.33
6	3	68.00	68.67	**67.33**	**66.00**	68.33	69.33

σ: task time variability; $wt_{average}^{mean}$: average walking time attribute between stations; #: number of solved instances; Avr. Sol.: average solution value.
Bold values denote the lowest average solution value among the considered cases.



(a) Effect of different task duration variability (σ)



(b) Effect of increase average walking time among the workstations ($wt^{mean}_{average}$)

Figure 6. Sensitivity analysis of different values of (σ) and ($wt^{mean}_{average}$).

As reported in Table 7, for the problem instances with IDs 1, 2, 3, 4, 5, and 6, lower cycle times (i.e., higher production rates) were achieved with increased task duration variability. It can therefore be concluded that higher production rates can be achieved for the instances with high task-duration variability. This pattern is seen in problem groups 2, 4, 5, and 6. On the other hand, when examining the effect of walking time, it is evident that, as expected, shorter walking times between stations lead to shorter cycle times. These results demonstrate that in assembly lines where differences in task completion times between workers are high and walking times between stations are low, allowing workers to walk between stations can increase the overall production rate.

7. Conclusions and Future Research Directions

Optimizing assembly line balancing problems with walking workers and worker-dependent task times remains a challenging yet highly relevant research topic. It directly depends on the operational realities of production systems. This study addresses a novel problem configuration that jointly considers worker mobility and heterogeneous task processing times within a unified framework. The main scientific contribution of this study lies in jointly considering assembly line balancing with walking workers and worker-dependent task times within a unified modeling and solution framework. To the best of our knowledge, this integrated problem configuration (WW-WALBP-II) has not been explicitly addressed in the existing assembly line balancing literature.

From a methodological perspective, a MILP formulation was developed to provide exact solutions and benchmark optimality limits, while a tailored constructive heuristic and a GA-based solution

approach were proposed to cope with larger instances. Comparative experiments against adapted MBO, PSO, and SA algorithms demonstrate that although the MILP is effective for small-scale instances, its applicability rapidly decreases as problem size increases. When exact solutions are obtainable, the proposed GA produces solutions with deviation values close to the optimal solutions. This indicates its capability to approximate high-quality schedules within reasonable computational effort. For medium and large-sized instances, the GA demonstrates competitive solution quality and robustness.

The obtained results are consistent with empirical observations commonly reported in studies on walking workers and ALWABPs. In particular, worker mobility can improve line performance when sufficient slack is available in the cycle time [5, 31, 37], whereas its impact becomes more limited as walking times increase. By jointly considering worker-dependent task durations and walking workers, this study extends these observations to a more integrated and operationally realistic setting. While walking times are considered explicitly, they are assumed to be fixed and workers independent. This assumption does not influence feasibility but mainly affects bound calculations, allowing the analysis to focus on the interaction between heterogeneous task assignments and worker capabilities. Nevertheless, it allows isolating the interaction between task-time heterogeneity and worker mobility, providing a clear baseline for future extensions.

From a managerial perspective, the results indicate that allowing workers to move between stations can increase production rates, particularly in environments with higher cycle times and heterogeneous worker capabilities. At the same time, explicitly accounting for walking times yields more realistic and implementable solutions. However, the benefits of worker mobility are not unbounded, since excessive walking times can limit the practical effectiveness of worker movement along the line. This insight provides practical guidance for decision-makers when defining station spacing and worker movement policies.

Future research may extend the proposed framework to more dynamic and uncertain production environments. Possible directions include incorporating stochastic task durations, worker fatigue, learning effects, and machine reliability. Hybrid configurations involving collaborative robots or multi-manned workstations with walking workers and worker-dependent task times also represent promising avenues for further investigation. From a methodological perspective, alternative exact and hybrid solution approaches could be explored as complementary solvers.

Use of Generative-AI tools declaration

AI-assisted tools (ChatGPT provided by OpenAI) were used solely to improve grammar and language clarity, without contributing to the scientific content.

Conflicts of interest

The authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Data availability statement

The data of problem instances are available upon reasonable request from the corresponding author.

References

1. F. Güner, A. K. Görür, B. Satır, L. Kandiller, J. H. Drake, A constraint programming approach to a real-world workforce scheduling problem for multi-manned assembly lines with sequence-dependent setup times, *Int. J. Prod. Res.*, **62** (2024), 3212–3229. https://doi.org/10.1080/00207543.2023.2226772
2. H. Güçdemir, G. Taşoğlu, Part transformation-based spare parts inventory control model for the high-tech industries, *Int. J. Ind. Eng. Comput.*, **15** (2024), 1–20. https://doi.org/10.5267/j.ijiec.2023.9.008
3. M. Şahin, T. Kellegöz, Balancing multi-manned assembly lines with walking workers: problem definition, mathematical formulation, and an electromagnetic field optimization algorithm, *Int. J. Prod. Res.*, **57** (2019), 6487–6505. https://doi.org/10.1080/00207543.2019.1566672
4. C. G. S. Sikora, T. C. Lopes, L. Magatão, Traveling worker assembly line (re) balancing problem: Model, reduction techniques, and real case studies, *Eur. J. Oper. Res.*, **259** (2017), 949–971. https://doi.org/10.1016/j.ejor.2016.11.027
5. S. Lassalle, Q. Wang, G. W. Owen, A. R. Mileham, A study of in-process waiting time on a linear walking worker assembly line, *Proc. Inst. Mech. Eng. B*, **221** (2007), 1763–1770. https://doi.org/10.1243/09544054JEM769
6. B. Sungur, Y. Yavuz, Assembly line balancing with hierarchical worker assignment, *J. Manuf. Syst.*, **37** (2015), 290–298. https://doi.org/10.1016/j.jmsy.2014.08.004
7. M. C. O. Moreira, J. F. Cordeau, A. M. Costa, G. Laporte, Robust assembly line balancing with heterogeneous workers, *Comput. Ind. Eng.*, **88** (2015), 254–263. https://doi.org/10.1016/j.cie.2015.07.004
8. N. P. Campana, M. Iori, M. C. O. Moreira, Mathematical models and heuristic methods for the assembly line balancing problem with hierarchical worker assignment, *Int. J. Prod. Res.*, **60** (2022), 2193–2211. https://doi.org/10.1080/00207543.2021.1884767
9. O. Polat, C. B. Kalayci, Ö. Mutlu, S. M. Gupta, A two-phase variable neighbourhood search algorithm for assembly line worker assignment and balancing problem type-II: an industrial case study, *Int. J. Prod. Res.*, **54** (2016), 722–741. https://doi.org/10.1080/00207543.2015.1055344
10. Q. Wang, G. W. Owen, A. R. Mileham, Determining numbers of workstations and operators for a linear walking-worker assembly line, *Int. J. Comput. Integr. Manuf.*, **20** (2007), 1–10. https://doi.org/10.1080/09511920600667358
11. E. Cevikcan, A mathematical programming approach for walking-worker assembly systems, *Assem. Autom.*, **34** (2014), 56–68. https://doi.org/10.1108/AA-07-2013-067
12. M. Liu, Z. Liu, F. Chu, R. Liu, F. Zheng, C. Chu, Risk-averse assembly line worker assignment and balancing problem with limited temporary workers and moving workers, *Int. J. Prod. Res.*, **60** (2022), 7074–7092. https://doi.org/10.1080/00207543.2021.2002960

13. Z. Mao, J. Zhang, Y. Sun, D. Huang, Y. Xu, A matheuristic approach for the multi-manned assembly line balancing problem with collaborative robots, *Comput. Oper. Res.*, **187** (2026), 107322. https://doi.org/10.1016/j.cor.2025.107322

14. C. Miralles, J. P. García-Sabater, C. Andrés, M. Cardós, Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled, *Discrete Appl. Math.*, **156** (2008), 352–367. https://doi.org/10.1016/j.dam.2005.12.012

15. J. P. Shewchuk, Worker allocation in lean U-shaped production lines, *Int. J. Prod. Res.*, **46** (2008), 3485–3502. https://doi.org/10.1080/00207540601115997

16. C. Blum, C. Miralles, On solving the assembly line worker assignment and balancing problem via beam search, *Comput. Oper. Res.*, **38** (2011), 328–339. https://doi.org/10.1016/j.cor.2010.05.008

17. M. C. O. Moreira, M. Ritt, A. M. Costa, A. A. Chaves, Simple heuristics for the assembly line worker assignment and balancing problem, *J. Heuristics*, **18** (2012), 505–524. https://doi.org/10.1007/s10732-012-9195-5

18. F. F. Araujo, A. M. Costa, C. Miralles, Two extensions for the ALWABP: Parallel stations and collaborative approach, *Int. J. Prod. Econ.*, **140** (2012), 483–495. https://doi.org/10.1016/j.ijpe.2012.06.032

19. Ö. Mutlu, O. Polat, A. A. Supciller, An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II, *Comput. Oper. Res.*, **40** (2013), 418–426. https://doi.org/10.1016/j.cor.2012.07.010

20. L. Borba, M. Ritt, A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem, *Comput. Oper. Res.*, **45** (2014), 87–96. https://doi.org/10.1016/j.cor.2013.12.002

21. M. Vila, J. Pereira, A branch-and-bound algorithm for assembly line worker assignment and balancing problems, *Comput. Oper. Res.*, **44** (2014), 105–114. https://doi.org/10.1016/j.cor.2013.10.016

22. M. C. O. Moreira, C. Miralles, A. M. Costa, Model and heuristics for the assembly line worker integration and balancing problem, *Comput. Oper. Res.*, **54** (2015), 64–73. https://doi.org/10.1016/j.cor.2014.08.021

23. P. T. Zacharia, A. C. Nearchou, A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem, *Eng. Appl. Artif. Intell.*, **49** (2016), 1–9. https://doi.org/10.1016/j.engappai.2015.11.007

24. M. K. Oksuz, K. Buyukozkan, S. I. Satoglu, U-shaped assembly line worker assignment and balancing problem: A mathematical model and two meta-heuristics, *Comput. Ind. Eng.*, **112** (2017), 246–263. https://doi.org/10.1016/j.cie.2017.08.030

25. M. C. O. Moreira, R. Pastor, A. M. Costa, C. Miralles, The multi-objective assembly line worker integration and balancing problem of type-2, *Comput. Oper. Res.*, **82** (2017), 114–125. https://doi.org/10.1016/j.cor.2017.01.003

26. A. Deepak, R. Srivatsan, V. Samsingh, A case study on implementation of walking worker assembly line to improve productivity and utilisation of resources in a heavy duty manufacturing industry, *FME Trans.*, **45** (2017), 497–504.

27. Ş. D. Akyol, A. Baykasoğlu, ErgoALWABP: a multiple-rule based constructive randomized search algorithm for solving assembly line worker assignment and balancing problem under ergonomic risk factors, *J. Intell. Manuf.*, **30** (2019), 291–302. https://doi.org/10.1007/s10845-016-1246-6

28. M. N. Janardhanan, Z. Li, P. Nielsen, Model and migrating birds optimization algorithm for two-sided assembly line worker assignment and balancing problem, *Soft Comput.*, **23** (2019), 11263–11276. https://doi.org/10.1007/s00500-018-03684-8

29. A. Karas, F. Ozcelik, Assembly line worker assignment and rebalancing problem: A mathematical model and an artificial bee colony algorithm, *Comput. Ind. Eng.*, **156** (2021), 107195. https://doi.org/10.1016/j.cie.2021.107195

30. S. E. Hashemi-Petroodi, S. Thevenin, S. Kovalev, A. Dolgui, Markov decision process for multi-manned mixed-model assembly lines with walking workers, *Int. J. Prod. Econ.*, **255** (2023), 108661. https://doi.org/10.1016/j.ijpe.2022.108661

31. M. Şahin, T. Kellegöz, Benders' decomposition based exact solution method for multi-manned assembly line balancing problem with walking workers, *Ann. Oper. Res.*, **321** (2023), 507–540. https://doi.org/10.1007/s10479-022-05118-z

32. M. Ebrahimi, M. Mahmoodjanloo, B. Einabadi, A. Baboli, E. Rother, A mixed-model assembly line sequencing problem with parallel stations and walking workers: a case study in the automotive industry, *Int. J. Prod. Res.*, **61** (2023), 993–1012. https://doi.org/10.1080/00207543.2021.2022801

33. M. Şahin, T. Kellegöz, Novel mathematical modelling approaches and a new lower bounding scheme for multi-manned assembly line balancing problems with walking workers, *Comput. Ind. Eng.*, **190** (2024), 110043. https://doi.org/10.1016/j.cie.2024.110043

34. M. A. Sato, A. M. Costa, Model and heuristics for the multi-manned assembly line worker integration and balancing problem, *Int. J. Prod. Res.*, **62** (2024), 8719–8744. https://doi.org/10.1080/00207543.2024.2347572

35. R. Sirovetnukul, P. Chutima, The impact of walking time on U-shaped assembly line worker allocation problems, *Eng. J.*, **14** (2010), 53–59.

36. A. Al-Zuheri, L. Luong, K. Xing, A framework supporting the design of walking worker assembly line towards improving productivity and ergonomics performance, *Int. J. Eng. Res. Appl.*, **4** (2014), 514–523.

37. M. Calzavara, M. Faccio, S. Finco, A. Persona, I. Zennaro, A selection procedure for the design of mixed-model assembly systems considering walking workers and fixed workers, *Int. J. Prod. Res.*, **63** (2025), 1028–1045. https://doi.org/10.1080/00207543.2024.2370013

38. X. Pucel, S. Roussel, Constraint programming model for assembly line balancing and scheduling with walking workers and parallel stations, In *Proc. Int. Conf. Principles Pract. Constraint Program. (CP)*, 2024.

39. S. Qin, S. Dai, J. Wang, S. Liu, X. Guo, L. Qi, Y. Ji, Improved carnivorous plant algorithm for human–robot collaborative U-shaped disassembly line balancing with mobile workers, *IEEE Trans. Comput. Soc. Syst.*, (2025). https://doi.org/10.1109/TCSS.2025.3596163

40. S. Qin, Y. Feng, J. Wang, S. Liu, X. Guo, L. Qi, Optimization of circular disassembly lines with human-assisted robotic workstations using two-stage greedy PPO algorithm, *IEEE Trans. Comput. Soc. Syst.*, (2025). https://doi.org/10.1109/TCSS.2025.3620906

41. Z. Zhang, Q. Tang, D. Han, Z. Li, Enhanced migrating birds optimization algorithm for U-shaped assembly line balancing problems with workers assignment, *Neural Comput. Appl.*, **31** (2019), 7501–7515. https://doi.org/10.1007/s00521-018-3596-9

42. R. Ramezanian, A. Ezzatpanah, Modeling and solving multi-objective mixed-model assembly line balancing and worker assignment problem, *Comput. Ind. Eng.*, **87** (2015), 74–80. https://doi.org/10.1016/j.cie.2015.04.017

43. Z. Mao, Y. Sun, K. Fang, D. Huang, J. Zhang, Model and metaheuristic for human–robot collaboration assembly line worker assignment and balancing problem, *Comput. Oper. Res.*, **165** (2024), 106605. https://doi.org/10.1016/j.cor.2024.106605

44. F. Catalano, I. Zennaro, N. Berti, A. Persona, Comparing fixed and walking worker strategies: design implications of individual worker efficiency on assembly line performance, *Int. J. Prod. Res.*, **63** (2025), 9089–9111.

45. M. Calzavara, M. Faccio, A. Persona, I. Zennaro, Walking worker vs fixed worker assembly considering the impact of components exposure on assembly time and energy expenditure, *Int. J. Adv. Manuf. Technol.*, **112** (2021), 2971–2988.

46. A. Scholl, *Balancing and sequencing of assembly lines*, Springer, 1999.

47. A. Goli, Efficient optimization of robust project scheduling for industry 4.0: A hybrid approach based on machine learning and meta-heuristic algorithms, *Int. J. Prod. Econ.*, **278** (2024), 109427. https://doi.org/10.1016/j.ijpe.2024.109427

48. M. Şahin, T. Kellegöz, Increasing production rate in U-type assembly lines with sequence-dependent set-up times, *Eng. Optim.*, **49** (2017), 1401–1419. https://doi.org/10.1080/0305215X.2016.1256394

49. A. Mellouli, R. Mellouli, H. Triki, F. Masmoudi, An efficient hybridization of ant colony optimization and genetic algorithm for an assembly line balancing problem of type II under zoning constraints, *Ann. Oper. Res.*, **351** (2025), 903–935. https://doi.org/10.1007/s10479-024-06071-9

50. A. Goli, E. B. Tirkolaee, G.-W. Weber, I. Mahdavi, A robust optimization model to design an IoT-based sustainable supply chain network with flexibility, *Cent. Eur. J. Oper. Res.*, **31** (2023), 1225–1253. https://doi.org/10.1007/s10100-023-00870-4

51. A. Hamidoglu, P. Khaleghi, Ö. M. Gul, A patient-centered equilibrium strategy for selecting anti-epileptic drugs in juvenile myoclonic epilepsy management, *J. Ind. Manag. Optim.*, **20** (2024), 3596–3616. https://doi.org/10.3934/jimo.2024011

52. A. Goli, T. Keshavarz, Just-in-time scheduling in identical parallel machine sequence-dependent group scheduling problem, *J. Ind. Manag. Optim.*, **18** (2022), 3807–3830. doi:10.3934/jimo.2021124

53. J. Taheri, A. Mirzazadeh, Optimization of inventory system with defects, rework failure and two types of errors under crisp and fuzzy approach, *J. Ind. Manag. Optim.*, **18** (2022), 2022–2050. https://doi.org/10.3934/jimo.2021068

54. D. E. Goldberg, R. Lingle, Alleles, Loci, and the Traveling Salesman Problem, *Proc. Int. Conf. Genetic Algorithms Appl.* (1985), 154–159.

55. M. Şahin, T. Kellegöz, An efficient grouping genetic algorithm for U-shaped assembly line balancing problems with maximizing production rate, *Memetic Comput.*, **9** (2017), 213–229. https://doi.org/10.1007/s12293-017-0239-0

56. D. I. Petropoulos, A. C. Nearchou, A particle swarm optimization algorithm for balancing assembly lines, *Assem. Autom.*, **31** (2011), 118–129. https://doi.org/10.1108/01445151111117700

57. K. Meng, Q. Tang, Z. Zhang, Balancing and sequencing of mixed-model assembly line considering preventive maintenance scenarios: mathematical model and a migrating birds optimization algorithm, *Flex. Serv. Manuf. J.*, **35** (2023), 1175–1205. https://doi.org/10.1007/s10696-022-09477-4

AIMS Press