Adv. Manuf. (2019) 7:117–132
https://doi.org/10.1007/s40436-019-00256-3

Hybrid genetic algorithm for a type-II robust mixed-model assembly line balancing problem with interval task times

Jia-Hua Zhang[1,2] · **Ai-Ping Li**[1] · **Xue-Mei Liu**[1]

Received: 11 August 2018 / Revised: 14 February 2019 / Accepted: 19 April 2019 / Published online: 1 June 2019

Abstract The type-II mixed-model assembly line balancing problem with uncertain task times is a critical problem. This paper addresses this issue of practical significance to production efficiency. Herein, a robust optimization model for this problem is formulated to hedge against uncertainty. Moreover, the counterpart of the robust optimization model is developed by duality. A hybrid genetic algorithm (HGA) is proposed to solve this problem. In this algorithm, a heuristic method is utilized to seed the initial population. In addition, an adaptive local search procedure and a discrete Levy flight are hybridized with the genetic algorithm (GA) to enhance the performance of the algorithm. The effectiveness of the HGA is tested on a set of benchmark instances. Furthermore, the effect of uncertainty parameters on production efficiency is also investigated.

Keywords Mixed-model assembly line · Assembly line balancing · Robust optimization · Genetic algorithm (GA) · Uncertainty

List of symbols

i, t	Index of assembly tasks, $i, t = 1, 2, \cdots, n$
j	Index of models, $j = 1, 2, \cdots, p$
k	Index of workstations, $k = 1, 2, \cdots, m$
n	Number of tasks
m	Number of predefined workstations
p	Number of models to be assembled on the line
C	Cycle time
w_j	Weight of model j in total production, $j = 1, 2, \cdots, p$
$\bar{t}_{ij}$	Nominal task time of task i for model j
$\hat{t}_{ij}$	Deviation task time from $\bar{t}_{ij}$
I_t	Set of immediate predecessors of task t, $t = 1, 2, \cdots, n$
γ	Budget for uncertainty, which means the number of uncertain tasks considered in a workstation, $0 \le \gamma \le n$
u_{ik}	Continuous variable
x_{ik}	$x_{ik} = 1$ if task i is assigned to workstation k; otherwise, $x_{ik} = 0$; $i = 1, 2, \cdots, n$; $k = 1, 2, \cdots, m$

1 Introduction

Assembly lines are production systems containing serially located workstations. Assembly tasks are completed in these serial workstations. The first real example of an assembly line was developed by Henry Ford in 1913. The production rate saw an eightfold increase with the introduction of the assembly line production. Since then, assembly lines have been widely used around the world. The assembly line balancing problem (ALBP) is one of the important problems in the design of an assembly line. ALBP is to assign assembly tasks among the workstations to optimize production objectives. This assignment must take the precedence relationship constraint and other technical constraints into consideration. ALBP has been an active area of research since the first mathematical model of ALBP was presented by Salveson [1]. The detailed reviews of research on ALBP can be found in Refs. [2–4].

✉ Jia-Hua Zhang
1510278@tongji.edu.cn

[1] School of Mechanical Engineering, Tongji University, Shanghai 201804, People's Republic of China

[2] Department of Mechatronics Engineering, Wuxi Vocational Institute of Arts and Technology, Yixing 214206, Jiangsu, People's Republic of China

A mixed-model assembly line is capable of assembling similar models from a basic product through a single assembly line. With increasing competition and diversity in customer demands, mixed-model assembly lines have become popular in many industries, such as cars, TVs, and computers. The mixed-model assembly line balancing problem (MMALBP) is classified into two types: MMALBP-I and MMALBP-II [5]. The former is aimed at minimizing the number of workstations for a given cycle time, whereas the latter is aimed at minimizing the cycle time under a given number of workstations.

In many studies of the MMALBP, task times are assumed to be deterministic. However, in real life, assembly tasks are subject to various uncertainties, such as the skill level of the operator, task complexity, and resource availability. Both stochastic mixed-model assembly line balancing [6, 7] and fuzzy mixed-model assembly line balancing [8] have been proposed to deal with uncertain task times. In stochastic mixed-model assembly line balancing, it is assumed that task times are subject to probability distribution, generally normal distribution. In fuzzy mixed-model assembly line balancing, task times are assumed to be fuzzy numbers with given membership. However, both stochastic and fuzzy task times are often impossible in practice because of insufficient preliminary information to deduct the required probability or possibility distribution functions.

Soyster [9] first developed the robust optimization approach in which the probability distribution of uncertain parameters is unknown. However, with the worst-case scenarios that may never happen in real life, this method is very conservative. To control the degree of conservatism, Ref. [10] developed a robust optimization, in which only a subset of uncertain coefficients (only γ of them) is in the worst scenarios. This approach can be extended to discrete optimization problems and has been applied to a variety of problems. Many researchers have adopted the robust optimization approach to study the assembly line balancing problem with uncertain task times: Al-e-hashem et al. [11] considered the type-I robust mixed-model assembly line balancing problem (RMMALBP-I). Hazır and Dolgui [12] studied the type-II robust simple assembly line balancing problem (RSALBP-II) and developed a Benders decomposition algorithm. Gurevsky et al. [13] investigated the type-I robust simple assembly line balancing problem (RSALBP-I) and designed a branch and bound algorithm. Nazarian and Ko [14] considered the uncertain task and inter-task times in RSALBP-II, especially focusing on non-productive times in workstations. Moreira et al. [15] studied RSALBP-I with heterogeneous workers under uncertain task times and developed two mathematical models and a heuristic method. Hazır and Dolgui [16] studied the robust U-shaped assembly line balancing problem, which was solved by a Benders decomposition algorithm. Pereira and Álvarez-Miranda [17] investigated RSALBP-I and developed a heuristic method and an exact algorithm.

To the best of our knowledge, only Al-e-hashem et al. [11] have used the robust optimization method to study the mixed-model assembly line balancing problem with uncertain task times, in addition to formulating a mathematical model of RMMALBP-I. However, there is no related work on RMMALBP-II, which is directly related to the production rate of the mixed-model assembly line. The MMALBP-II with interval task times is studied in this paper using the robust optimization method, which is called robust MMALBP-II (RMMALBP-II). In this problem, each task time is represented by an interval dataset. To solve this problem, the robust optimization model is formulated and a hybrid genetic algorithm (HGA) is developed. The robust optimization model for this problem is nonlinear. To facilitate the use of an exact algorithm, the counterpart of the nonlinear robust optimization model is obtained by duality.

The remainder of this paper is organized as follows: the RMMALBP-II is described and the mathematical models are formulated in Sect. 2. The HGA is developed in Sect. 3. In Sect. 4, computational experiments are implemented, along with the illustration of the results. Finally, conclusions are drawn in Sect. 5.

2 Problem description and model formulation

2.1 Problem description

A mixed-model assembly line with m workstations is considered. p similar models are assembled simultaneously in an intermixed sequence. The ratio of the unit number of each model j to the overall demand is w_j. The precedence relationship of each model is predefined and all relationships can be combined into only one precedence graph with n tasks. Each task has an uncertain task time. The uncertainty information about the probability distribution or fuzzy membership function is not easy to know, and each uncertain task time can only be represented by an interval $\left[\bar{t}_{ij} \pm \hat{t}_{ij}\right]$. $\bar{t}_{ij}$ is the nominal task time for task i and model j, and $\hat{t}_{ij}$ corresponds to the deviation task time from $\bar{t}_{ij}$. If the interval $\left[\bar{t}_{ij} \pm \hat{t}_{ij}\right]$ equals 0, it means that model j does not need task i to be assembled. The aim is to minimize cycle time C.

The RMMALBP-II has the following assumptions:

(i) The task time of each model is uncertain and is represented by a given interval dataset.
(ii) Each common task for different models must be assigned to the same workstation for economy.

(iii) The precedence graphs for different models are predefined, and a combined precedence graph can be obtained.
(iv) The line is a paced line with a fixed cycle time.
(v) The line is serial with no feeder lines or parallel workstations.
(vi) There are no assignment restrictions of tasks except precedence constraints.
(vii) There are no buffers between workstations.

2.2 Model formulation

The RMMALBP-II can be formulated as a nonlinear robust model:

$$\min C \tag{1}$$

subject to

$$\sum_{k=1}^{m} x_{ik} = 1, \quad \forall i, \tag{2}$$

$$\sum_{k=1}^{m} k x_{ik} \le \sum_{k=1}^{m} k x_{tk}, \quad \forall i \in I_t, \tag{3}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{p} w_j \bar{t}_{ij} x_{ik} + \max\left\{ \sum_{i=1}^{n}\sum_{j=1}^{p} w_j \hat{t}_{ij} x_{ik} u_{ik} : \sum_{i=1}^{n} u_{ik} \le \gamma \right\} \le C, \quad \forall k, \tag{4}$$

$$0 \le u_{ik} \le 1, \quad \forall i, k, \tag{5}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i, k, \tag{6}$$

$$C > 0, \quad C \text{ is integer.} \tag{7}$$

Objective function (1) minimizes cycle time C. Constraint (2) is known as the occurrence constraint that each task can be assigned to a workstation. Constraint (3) is the precedence constraint to guarantee the technological sequencing requirements. Constraint (4) ensures that the weighted uncertain workstation times do not exceed the cycle time C. The left-hand side of constraint (4) consists of two parts: $\sum_{j=1}^{p} w_j \bar{t}_{ij}$ is the weighted nominal task time of each task i which can be assigned into a workstation and $\sum_{j=1}^{p} w_j \hat{t}_{ij}$ is the weighted deviation task time for each task i. The number of uncertain tasks considered in a workstation is bounded by parameter γ. The larger γ is, the more the deviation task times should be considered. Constraint (5) determines the bound of variable u_{ik}, which indicates the level of an uncertain task time deviating from the nominal time. When $u_{ik} = 0$, it means that the task time for task i in workstation k does not deviate from the nominal time and is deterministic. When $u_{ik} = 1$, task i in workstation k is under the worst case, and the task time is equal to $\bar{t}_{ij} + \hat{t}_{ij}$. Constraint (6) is the nondivisibility constraint, which means that a task cannot be split among two or more workstations. Constraint (7) makes sure that C has an integer value, which is always the case in a real-world environment [5].

2.3 Counterpart of the nonlinear robust optimization model

The robust optimization model proposed in Sect. 2.2 is a nonlinear model. It can be linearized by duality [10] and solved by an integer solver. The counterpart of the nonlinear robust optimization model is developed as follows:

$$\min C \tag{8}$$

subject to

$$\sum_{i=1}^{n}\sum_{j=1}^{p} w_j \bar{t}_{ij} x_{ik} + \sum_{i=1}^{n} p_{ik} + \gamma z_k \le C, \quad \forall k, \tag{9}$$

$$z_k + p_{ik} \ge \sum_{j=1}^{p} w_j \hat{t}_{ij} y_{ik}, \quad \forall i, k, \tag{10}$$

$$p_{ik} \ge 0, \quad \forall i, k, \tag{11}$$

$$x_{ik} \le y_{ik}, \quad \forall i, k, \tag{12}$$

$$z_k \ge 0, \quad \forall k, \tag{13}$$

and constraints (2), (3), (6), and (7). y_{ik}, p_{ik}, and z_k are the variables used in duality.

3 Proposed hybrid genetic algorithm for RMALBP-II

Type-II simple assembly line balancing problem (SALBP-II) is known to be nondeterministic polynomial (NP)-hard [2]. It can be found that SALBP-II is a special case of RMALBP-II. Similar to SALBP-II, RMLABP-II is NP-hard. Owing to its NP-hard nature, RMALBP-II can be time-consuming to obtain an optimal solution using exact algorithms. In this section, an HGA is developed to solve this problem.

The genetic algorithm (GA), a popular meta-heuristic algorithm, has been used to solve various assembly line balancing problems such as stochastic assembly line balancing problem, fuzzy assembly line balancing problem, and so on [18]. Research shows that GA is competitive against the best-known constructive methods. There is no published research reporting the application of the GA to solve the robust assembly line balancing problem. In this section, an HGA is proposed to solve RMMALBP-II. The flowchart of this algorithm is shown in Fig. 1.

This algorithm starts with the generation of an initial feasible population, which is followed by the evaluation of each chromosome through fitness evaluation. Special genetic operators for the assembly line balancing problem (crossover and mutation) are performed. An adaptive local search procedure and a discrete Levy flight are used to improve the quality of solutions. This loop keeps running until the maximum iteration number is reached. Finally, the best cycle time of RMMALBP-II is obtained.

3.1 Encoding scheme and initial population

A combined precedence graph, G, is formed by combining the precedence graphs of each model. Based on the combined precedence graph, common tasks of each model can be assigned to the same workstation. Each chromosome, according to their precedence constraint, is designed as a sequence of tasks. The number of genes in the chromosome is equal to that of tasks n in G, and each gene is an integer representing a task.

Population initialization is a crucial step in evolutionary algorithms (EAs). Many researchers have proposed to seed EAs with good initial solutions, whenever it is possible, to obtain important improvement in the convergence of the algorithm and the quality of the solutions [19]. However, the excessive use of good solutions in the initial population can decrease the exploration capacity of the GA, thereby trapping the population in local optimums quickly [20, 21].

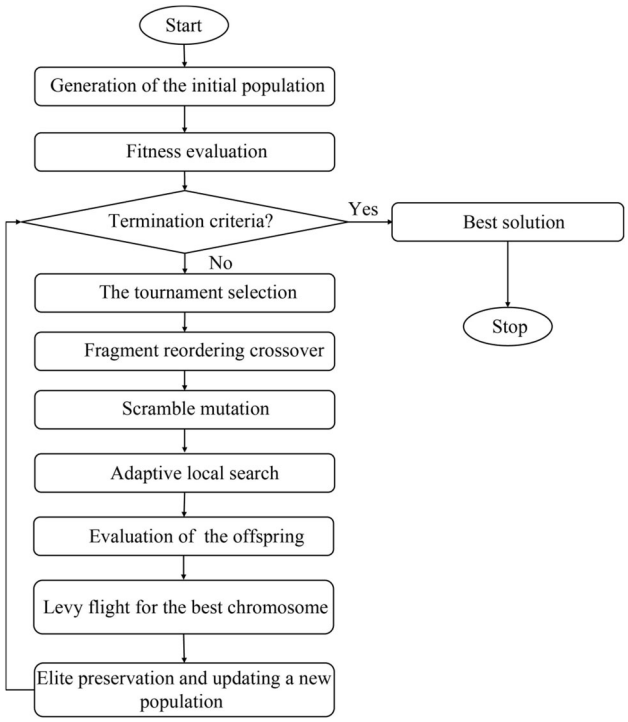

Fig. 1 Flowchart of the proposed HGA

In this case, it is proposed to seed part of the initial population with heuristic solutions to improve the performance of the algorithm. Based on the experimental results, 20% of the initial population can be seeded by the proposed heuristic and 80% can be generated randomly. This initialization method can obtain better solutions than that of a randomly generated population, as shown in Sect. 4.1.

The heuristic method is based on Pereira and Álvarez-Miranda [17] and Sewell and Jacobson [22], and builds solutions from the forward or backward direction of G. Task j in G, which maximizes the priority rule $\sum_{i\in U}(w_i\bar{t}_i + aw_i\hat{t}_i + b|F_i| - c)$, can be selected. U is the set of tasks that have not yet been selected; $w_i\bar{t}_i$ is the weighted nominal task time; $w_i\hat{t}_i$ is the weighted deviation time; $|F_i|$ is the number of immediate successors of task i in the forward direction search or that of immediate predecessors of task i in the backward direction search. a, b, and c are input parameters. The recommended values for a, b, and c in Ref. [22] are employed. Every random combination of $a \in \{0,0.005,0.010,0.015,0.020\}$, $b \in \{0, 0.005,0.010,0.015,0.020\}$, and $c \in \{0,0.01,0.02,0.03\}$ is used to select a task. The random method selects a task at random and assures the feasibility of the precedence relationship. The precedence matrix is used to describe the precedence relationship of tasks [23].

The procedure of the initial feasible population generation is described as follows.

Step 1 Build an empty vector $\boldsymbol{A}$; read the precedence matrix $\boldsymbol{P}$ of the combined precedence graph G.

Step 2 Choose the random creation method or the heuristic method.

(i) If the random creation method is chosen, go to Step 3.

(ii) If the heuristic method is chosen, decide the search direction randomly. If it is a backward direction search, $\boldsymbol{P} = \boldsymbol{P}'$ and go to Step 3; otherwise, go to Step 3.

Step 3 If $j \leq n$, store the tasks where the sum of the column of $\boldsymbol{P}$ is equal to 0 into $\boldsymbol{A}$; if $j > n$, go to Step 7.

Step 4 Choose a task from $\boldsymbol{A}$ for a gene in a chromosome.

(i) If it is the random creation method, choose a task from $\boldsymbol{A}$ at random for the jth gene in a chromosome. Empty $\boldsymbol{A}$, and go to Step 5.

(ii) If it is the heuristic method, generate a random combination of input parameters a, b, and c, and choose the task that maximizes the priority rule from $\boldsymbol{A}$. The task is assigned for the jth gene in the forward direction search or for the $(n - j + 1)$th in the backward direction search. Empty $\boldsymbol{A}$, and go to Step 5.

Step 5 Update the ith row of $\boldsymbol{P}$ by putting a big number D (e.g., 999) into tasks i and 0 into other tasks.

Step 6 Update $j = j + 1$, and go to Step 3.
Step 7 End the procedure.

Repeat the above procedure and get the initial population. Figure 2 is the combined precedence graph G of an illustrative example. Table 1 is the precedence matrix $\boldsymbol{P}$ of graph G. If the heuristic method from backward search is chosen, make $\boldsymbol{P} = \boldsymbol{P}'$. Table 2 shows the result of Steps 2 and 3 of the heuristic method from the backward search direction. After these two steps, the element of vector $\boldsymbol{A}$ is task 11 when $j = 1$. Because there is only one element in $\boldsymbol{A}$ to calculate the priority rule, the $(n - j + 1)$th gene (it is the nth gene now) in the chromosome is task 11 in Step 4.

Table 3 is the result of Step 5. Repeat this procedure until the last task is selected into a chromosome. Table 4 is the final result of the sum of columns. Figure 3 shows some chromosomes in the initial population produced by the two methods.

3.2 Evaluation procedure

The evaluation procedure aims to find objective values based on the task sequence. With each chromosome being a feasible task sequence, the procedure decides which tasks can be assigned to predefined workstations respecting the cycle time constraint. The evaluation procedure is described as follows.

Step 1 Read the nominal time matrix $\boldsymbol{T}$ and the deviation task time matrix $\boldsymbol{V}$.
Step 2 Calculate the theoretical minimum cycle time for the initial trial cycle time $\bar{C}$. This initial cycle time value is the lower boundary (LB), and the calculation equation is presented as

$$\mathrm{LB} = \begin{cases} \mathrm{ceil}\left(\max\left(\max(\bar{t}_i), \sum_{i=1}^{n} \bar{t}_i \Big/ m\right)\right), & \gamma = 0, \\ \mathrm{ceil}\left(\max\left(\max(\bar{t}_i + \hat{t}_i), \left(\sum_{i=1}^{n} \bar{t}_i + \max(\hat{t}_i)\right)/m\right)\right), & \gamma \geq 1, \end{cases} \tag{14}$$

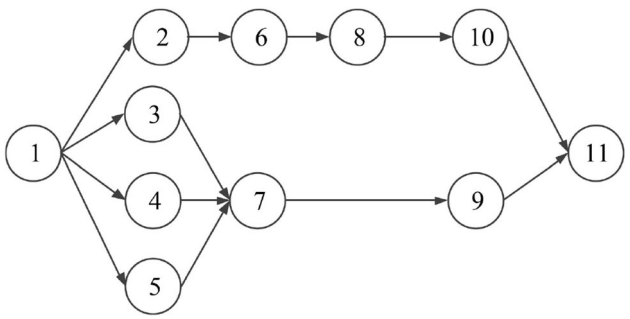

Fig. 2 Combined precedence graph G of the illustrative example

Table 1 Precedence matrix $\boldsymbol{P}$ of G

Task	1	2	3	4	5	6	7	8	9	10	11
1	0	1	1	1	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0

Table 2 Searching for the tasks to formulate a feasible task sequence from the backward direction

Task	Step 2: Transposed matrix of $\boldsymbol{P}$										
	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0	0
7	0	0	1	1	1	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	1	1	0
Step 3: Sum of columns	4	1	1	1	1	1	1	1	1	1	0

where $\bar{t}_i = \sum_{j=1}^{p} w_j \bar{t}_{ij}$ represents the weighted nominal task time for task i and $\hat{t}_i = \sum_{j=1}^{p} w_j \hat{t}_{ij}$ represents the weighted deviation task time for task i.

If $\gamma = 0$, it means that no task deviation time is considered. The LB is equal to the LB of the deterministic MMALBP-II. For the cycle time set as an integer in RMMALBP-II, the LB is taken as an integer. If $\gamma \geq 1$, it means that at least one task deviation time is considered. The indivisibility of tasks requires that $C \geq (\bar{t}_i + \hat{t}_i)$. With at least one task deviation time being considered, $mC \geq \left(\sum_{i=1}^{n} \bar{t}_i + \max(\hat{t}_i)\right)$ is obtained.

Step 3 Assign the tasks into predetermined workstations. Tasks are assigned to the first station in the order of the

Table 3 Updating ***P*** after a task selected

Task	Step 5: Matrix ***P***										
	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0	0
7	0	0	1	1	1	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	*D*

Table 4 Final content of ***P*** after forming a feasible task sequence

Task	Matrix ***P***										
	1	2	3	4	5	6	7	8	9	10	11
1	*D*	0	0	0	0	0	0	0	0	0	0
2	0	*D*	0	0	0	0	0	0	0	0	0
3	0	0	*D*	0	0	0	0	0	0	0	0
4	0	0	0	*D*	0	0	0	0	0	0	0
5	0	0	0	0	*D*	0	0	0	0	0	0
6	0	0	0	0	0	*D*	0	0	0	0	0
7	0	0	0	0	0	0	*D*	0	0	0	0
8	0	0	0	0	0	0	0	*D*	0	0	0
9	0	0	0	0	0	0	0	0	*D*	0	0
10	0	0	0	0	0	0	0	0	0	*D*	0
11	0	0	0	0	0	0	0	0	0	0	*D*
Sum of columns	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*	*D*

1	5	3	2	6	8	10	4	7	9	11
1	2	4	5	6	8	10	3	7	9	11

Fig. 3 Some chromosomes in an initial population

gene sequence complying with the precedence relationship. The cycle time constraint as shown in inequality (4) cannot be violated. The cycle time constraint can be described as

$$\sum_{i=1}^{n} \overline{t}_i x_{ik} + \max\left(\sum_{i=1}^{n} \hat{t}_i x_{ik} u_{ik} : \sum_{i=1}^{n} u_{ik} \le \gamma\right) \le \overline{C}. \qquad (15)$$

Rank the deviation time of tasks in this workstation in the descending order. The worst scenario of the workstation time is equal to the sum of the nominal time assigned to this workstation and the largest γ deviation time among these tasks. Once the workstation time exceeds the cycle time, the next station is opened for assignment. If the number of workstations is equal to the given m, stop the assignment and calculate the worst scenario workstation time of each station. The maximum workstation time is defined as the bottleneck workstation time C_w.

Step 4 If $C_w \le \bar{C}$, go to Step 5; otherwise, update $\bar{C} = \bar{C} + 1$ and go to Step 3.

Step 5 The cycle time is equal to $\bar{C}$; end this procedure.

Table 5 is an illustration of the decoding/evaluating procedure. The predetermined workstation number is $m = 4$.

3.3 Tournament selection

The tournament selection strategy [24] is used to select the parent chromosomes. Some chromosomes are randomly chosen from the current population along with the one with the best objective value being selected for reproduction. The tournament selection strategy works as follows.

Step 1 k chromosomes in the population are selected at random.

Step 2 The chromosome with the best objective value (minimum cycle time) from these selected chromosomes is chosen as the best one and added to the mating pool.

Step 3 This procedure is repeated until the number of individuals in the mating pool reaches the required population size and the population is updated through this procedure.

3.4 Fragment reordering crossover

Because it is designed particularly for the assembly line balancing problem [25], the fragment reordering crossover can preserve the feasibility of the offspring structure with its procedure working as follows.

Step 1 Two parent individuals are selected from the population in order.

Step 2 Two parents selected are divided by two randomly cut points into three sections: head, middle, and tail.

Step 3 The head and tail parts of the first offspring are taken from the first parent and the middle part of the first offspring is filled by adding missing tasks according to the order in which they are contained in the second parent.

Table 5 Evaluation procedure for the illustrative example with $\gamma = 1$

Chromosome		1	5	2	6	4	3	7	9	8	10	11
Step 1	$\overline{t_i}$	6	1	2	2	7	5	3	5	6	5	4
	$\hat{t}_i$	0.6	0.1	0.2	0.2	0.7	0.5	0.3	0.5	0.6	0.5	0.4
Step 2	LB	LB = ceil(max (7.7, 4.8848)) = 8										
Step 3	Workstation (WS)	WS1(1 5) WS2(2 6) WS3(4) WS4(3 7 9 8 10 11)										
	C_w	WS4 is bottleneck; $C_w = 28 + 0.6 = 28.6 > \bar{C}$										
Step 4	Update $\bar{C}$	$\bar{C} = \bar{C} + 1$										
		Repeat										
Step 5	C	The cycle time of this chromosome is $C = 14$										
		End the procedure										

Step 4 Like the building process for the first offspring, the head and tail of the second offspring are formed from the same part of the second parent, and the middle is filled by the missing tasks according to the order in which they are contained in the first parent.

Step 5 Get a new population with the offspring chromosomes.

The fragment reordering crossover is demonstrated in Fig. 4.

3.5 Scramble mutation

Scramble mutation for the assembly line balancing problem was developed by Leu et al. [26]. With scramble mutation, the chromosome, reconstructed drastically, still remains feasible. It works as follows.

Step 1 The mutation point is generated randomly, and one chromosome from the population is divided into head and tail.

Step 2 The head of the chromosome is kept, and the tail of the chromosome is regenerated respecting the precedence relationship. The precedence matrix needs to be proceeded to eliminate the task already in the head of the chromosome.

Step 3 Get a new population with the mutation procedure.

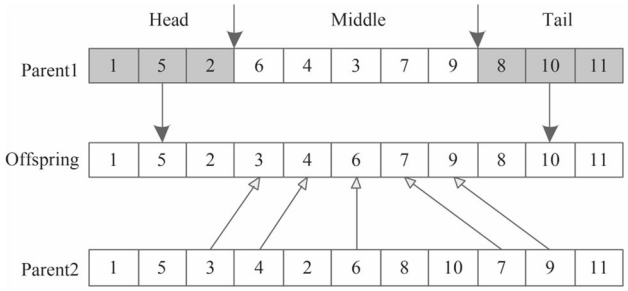

Fig. 4 Fragment reordering crossover

An illustration of a scramble mutation is given in Fig. 5. The mutation point is 3 in the chromosome of this example.

3.6 Adaptive local search

The pure genetic algorithm is good at global search but slow to converge [27]. Local search is a promising approach to improve the quality of the objective value and convergence speed [28]. Within the proposed algorithm, a local search procedure is applied to every chromosome of the population. The local search tries to transfer tasks in the bottleneck workstation to other workstations to reduce the cycle time. To tackle the increased computation time of the local search procedure, an adaptive local search scheme is adopted. The basic concept of applying the local search to the population is to consider whether GA has converged to the global optimal solution or not [29]. The converging criterion for the line balancing problem is the ratio of the average fitness of the chromosomes to the fitness of the best chromosome less than 1.01 [30].

The fitness value ratio (FVR), R_{fv}, of the average fitness of the chromosomes to the fitness of the best chromosome at each generation is defined as

$$R_{fv} = \frac{a_f}{b_f}, \tag{16}$$

where a_f is the average fitness of the chromosomes and b_f is the fitness of the best chromosome.

If $R_{fv} > 1.01$, apply the local search; otherwise, only GA is implemented.

The local search procedure is explained as follows.

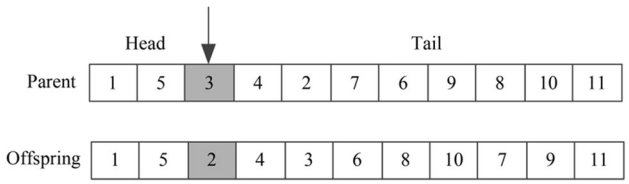

Fig. 5 Scramble mutation

Step 1 Identify the workstation with the largest workstation times as the bottleneck workstation.
Step 2 Let n_b be the number of tasks in the bottleneck workstation. If $i \le n_b$, find the ith task in the bottleneck workstation. According to the precedence relationship, find the earliest workstation $E(i)$ and the latest workstation $L(i)$ to which task i can be transferred. If $i > n_b$, go to Step 6.
Step 3 Rank workstations between $E(i)$ and $L(i)$ according to workstation times in the ascending order.
Step 4 k is the workstation between $E(i)$ and $L(i)$. Transfer task i to the workstations arranged in order of Step 3. Task i can be transferred to a workstation with $\left(\bar{T}_k + \bar{t}_i + \max\left\{\sum_{j\in M_k\cup\{i\}} \hat{t}_j x_{jk} u_{jk} : \sum_{j\in M_k\cup\{i\}} u_{jk} \le \gamma\right\}\right) \le T_b$, and go to Step 6. T_b is the workstation time of the bottleneck workstation considering task deviation times. $\overline{T_k}$ is the total nominal task time of station k. M_k is the set of tasks in workstation k.
Step 5 Update $i = i + 1$, and go to Step 2.
Step 6 Get the new chromosome and end the local search procedure.

The application of the local search procedure is described in Table 6 for a chromosome.

3.7 Discrete Levy flight

The Levy flight can improve the performance of nature-inspired algorithms [31]. A new solution x^{t+1} can be obtained through the Levy flight:

$$x_k^{t+1} = x_k^t + \alpha\lambda, \tag{17}$$

where α is the information about the step length and λ is the random step length drawn from the Levy distribution. λ is calculated as

$$\lambda = \frac{u}{|v|^{1/\beta}}, \tag{18}$$

where u and v are drawn from normal distributions. That is,

$$\begin{cases} u \sim N(0, \sigma_u^2), \\ v \sim N(0, \sigma_v^2), \end{cases} \tag{19}$$

$$\begin{cases} \sigma_u = \left(\dfrac{\Gamma(1+\beta)\sin\left(\dfrac{\pi\beta}{2}\right)}{\Gamma\left(\dfrac{(1+\beta)}{2}\right)\beta^{\frac{\beta-1}{2}}}\right)^{\frac{1}{\beta}}, \\ \sigma_v = 1, \end{cases} \tag{20}$$

where distribution parameter $\beta \in [1, 2]$ and Γ denotes the gamma function.

However, the Levy flight cannot be directly used in discrete optimization problems. In this paper, the discrete Levy flight proposed by Li et al. [32] is modified by considering uncertain task times and implemented on the chromosome with the best solution to generate a new chromosome stochastically:

$$x_k(t+1) = x_k(t) \oplus \bar{\lambda}, \tag{21}$$

where $x_k(t)$ is the task at location k in the chromosome of generation t; $\bar{\lambda}$ is the new task, and $x_k(t+1)$ is the new task at location k in the chromosome of generation $t+1$.

The fitness values of the two chromosomes before and after the discrete Levy flight are compared, and the chromosome with the better solution is kept in the population.

The discrete Levy flight procedure is described as follows.

1	3	4	2	6	8	10	5	7	9	11

Fig. 6 Chromosome for discrete Levy flight

Table 6 Local search procedure for the illustrative example with $\gamma = 1$

Chromosome		1	2	5	6	3	4	7	8	10	9	11
Step 1	Workstation (WS)	WS1				WS2		**WS3**			WS4	
	Time	11.6				12.7		**14.6**			9.5	
Step 2	$E(7)$, $L(7)$	$E(7)$ = WS2, $L(7)$ = WS4										
Step 3	Ascending order	WS4(9.5) → WS2(12.7) → WS3(14.6)										
Step 4	Transferring	Task 7 → WS4										
		WS4 time = 9 + 3 + 0.5 = 12.5 < 14.6										
Step 6	New chromosome	1	2	5	6	3	4	8	10	**7**	9	11
		End local search procedure										

Table 7 Discrete Levy flight procedure for the illustrative example

	Task	1	2	3	4	5	6	7	8	9	10	11
	1	*D*	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	1	0	0	0	0	0
	3	0	0	*D*	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	1	0	0	0	0
Step 2	6	0	0	0	0	0	0	0	1	0	0	0
	7	0	0	0	0	0	0	0	0	1	0	0
	8	0	0	0	0	0	0	0	0	0	1	0
	9	0	0	0	0	0	0	0	0	0	0	1
	10	0	0	0	0	0	0	0	0	0	0	1
	11	0	0	0	0	0	0	0	0	0	0	0
Step 3		λ =2.915										
Step 4		*S*=(2,4,5);*q*=3.2065; S_1=(2,5)										
Step 5	λ >1; $\bar{\lambda}=2$	1	3	2								
		Repeat										

Table 8 Benchmark instances

Name	*n*	*m*	Models	Product mix	ψ
Small size					
Mertens	7	4	2	(0.4,0.6)	0.5
Bowman	8	5	2	(0.5,0.5)	0.5
Jaeschke	9	4	2	(0.8,0.2)	0.3
Mansoor	11	4	2	(0.7,0.3)	0.1
Jackson	11	4	2	(0.4,0.6)	0.3
Medium size					
Mitchell	21	4	2	(0.4,0.6)	0.2
Rosizeg	25	7	4	(0.2,0.3,0.1,0.4)	0.4
Buxey	29	6	2	(0.3,0.7)	0.3
Sawyer	30	8	2	(0.5,0.5)	0.2
Gunther	35	6	3	(0.2,0.3,0.5)	0.1
Large size					
Kilbridge	45	5	2	(0.9,0.1)	0.1
Warnecke	58	12	2	(0.6,0.4)	0.1
Tong	70	16	2	(0.1,0.9)	0.2
Wee-Mag	75	20	3	(0.4,0.3,0.3)	0.3
Mukherje	94	22	2	(0.7,0.3)	0.1

Step 1 Select location *k* randomly as the starting point of the Levy flight.

Table 9 Parameters of the proposed algorithm

Parameter	Value
Population size	50
Maximal iteration number	100
Tournament size	2
Crossover probability/%	80
Mutation probability/%	15

Table 10 Results of Sawyer using different initializations with γ = 1

Initialization method	RI			PI		
	Min	Mean	SD	Min	Mean	SD
Sawyer	46	46.7	0.67	45	46.4	0.65

Step 2 Obtain the precedence matrix of the part before the chromosome location *k*.

Step 3 Calculate step length λ according to Eq. (18), where β = 1.5.

Step 4 Obtain the feasible task set *S* using the precedence matrix, which makes the Levy flight obtain a feasible solution. Calculate value $q = \min\{\lambda(\bar{t}_i+\hat{t}_i)\}$, where

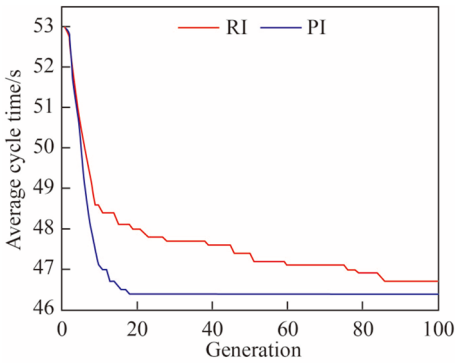

Fig. 7 Convergence behaviors of different initializations

$i \in S$. Choose task s that meets $(\bar{t}_s + \hat{t}_s) \le q, (s \in S)$ to form the task set S_1.

Step 5 If $\lambda \le 1$, select task j with the minimum $(\bar{t}_j + \hat{t}_j)$ in set S as $\bar{\lambda}$; otherwise, choose the task with maximum task times in set S_1 as $\bar{\lambda}$.

Step 6 $k = k + 1$, go to Step 2 until $k = n$.

Take a chromosome in Fig. 6 for example. Location 3 in the chromosome is randomly selected as the starting point of the Levy flight. The discrete Levy flight is described in Table 7.

3.8 Elite preservation

Each individual with minimum cycle time is preserved for the next generation. The fitness values of individuals from the current population and the offspring after the Levy flight are compared, and the individuals with best fitness values are preserved to form a new generation.

4 Numerical experiments

The proposed algorithm is implemented in MATLABR2013b on a PC with Intel®Core™i5-4210U CPU, 1.70 GHz and tested on 15 benchmark instances from www.assembly-line-balancing.de. Fifteen benchmark instances are classified into small size (7–11 tasks), medium size (21–35 tasks), and large size (45–94 tasks). The original benchmark instances only take the deterministic situation into consideration. In our numerical experiments, the precedence relationships and the nominal task times are obtained from the benchmark instances. Moreover, to define the robust problem, the coefficient of variation ψ is used to create the deviation task time from the nominal task time, $\hat{t}_{ij} = \psi \bar{t}_{ij}$. ψ is randomly generated using uniform distribution in interval [0.1, 0.5] for obtaining $\hat{t}_{ij}$. The product mix is generated at random. ψ and the product mix of each instance are given in Table 8. In each instance, the number of workstations is fixed, with the objective being to

Table 11 Results of benchmark instances with $\gamma = 1$

Name	LINGO		GA				HGA				Improvement/%
	OCT	CPU/s	Min	Mean	SD	CPU/s	Min	Mean	SD	CPU/s	
Small size											
Mertens	12	1	12	12	0	30.02	12	12	0	31.52	0
Bowman	26	1	26	26	0	41.71	26	26	0	43.02	0
Jaeschke	12	1	12	12	0	30.37	12	12	0	31.2	0
Mansoor	52	1	52	52	0	32.57	52	52	0	33.79	0
Jackson	14	1	14	14	0	37.09	14	14	0	60.11	0
Medium size											
Mitchell	29	1	29	29	0	49.13	29	29	0	73.71	0
Rosizeg	22	1	22	22.3	0.48	75.1	22	22	0	110.36	1.35
Buxey	61	48	62	63	0.63	212.11	61	61	0	347.67	3.17
Sawyer	45	292	47	48	1.05	151.6	45	46.4	0.65	311.83	3.33
Gunther	86	323	87	88	1.15	228.69	86	86.4	0.84	309.01	1.82
Large size											
Kilbridge	114	548	114	114.7	0.48	328.21	114	114	0	517.89	0.61
Warnecke	N/A		138	141	2.32	987.1	137	138.6	1.1	1 720.01	1.7
Tong	N/A		252	253.2	1.55	2 189.08	246	247.4	1.64	2 798.53	2.29
Wee-Mag	N/A		87	87.9	0.99	1 563.5	86	87.3	0.64	2 964.67	0.68
Mukherje	N/A		217	218.5	1.18	1 048.02	215	216.7	1.06	1 478.24	0.82

Table 12 Results of benchmark instances with $\gamma = 2$

Name	LINGO		GA				HGA				Improvement /%
	OCT	CPU/s	Min	Mean	SD	CPU/s	Min	Mean	SD	CPU/s	
Small size											
Mertens	14	1	14	14	0	36.73	14	14	0	39.63	0
Bowman	26	1	26	26	0	43.06	26	26	0	46.36	0
Jaeschke	13	1	13	13	0	35.76	13	13	0	38.88	0
Mansoor	53	1	53	53	0	38.77	53	53	0	39.34	0
Jackson	15	1	15	15	0	45.53	15	15	0	73.21	0
Medium size											
Mitchell	31	1	31	31	0	51.53	31	31	0	77.44	0
Rosizeg	25	2	25	25	0	79.05	25	25	0	142.30	0
Buxey	65	59	66	67.7	0.82	240.24	65	65.7	0.48	440.23	2.95
Sawyer	48	2 094	49	51.1	1.2	158.70	48	48.6	0.84	312.23	4.89
Gunther	88	2 778	88	90.2	1.4	244.35	88	88.5	0.97	370.34	1.88
Large size											
Kilbridge	116	3 665	117	117	0	348.58	116	116.2	0.42	535.05	0.68
Warnecke	N/A		143	144.7	1.34	1 035.84	140	142.2	1.2	1803.90	1.73
Tong	N/A		265	268.7	2.70	2 379.79	258	259.4	1.08	2 906.46	3.46
Wee-Mag	N/A		94	95.5	0.81	1 782.12	93	93.2	0.63	3 131.39	2.41
Mukherje	N/A		221	223.6	1.71	1 226.63	219	221.4	0.97	1 806.04	0.98

Table 13 Results of benchmark instances with $\gamma = 3$

Name	LINGO		GA				HGA				Improvement/%
	OCT	CPU/s	Min	Mean	SD	CPU/s	Min	Mean	SD	CPU/s	
Small size											
Mertens	14	1	14	14	0	38.03	14	14	0	39.89	0
Bowman	26	1	26	26	0	43.95	26	26	0	53.13	0
Jaeschke	13	1	13	13	0	35.52	13	13	0	52.50	0
Mansoor	53	1	53	53	0	31.05	53	53	0	37.30	0
Jackson	16	1	16	16	0	46.41	16	16	0	72.58	0
Medium size											
Mitchell	32	1	32	32	0	54.07	32	32	0	72.94	0
Rosizeg	26	1	26	26	0	77.38	26	26	0	160.70	0
Buxey	68	27	71	71.5	0.55	259.58	68	68.8	0.42	474.01	3.78
Sawyer	49	1 384	51	51.7	1.17	176.04	49	50	0.47	359.62	3.29
Gunther	90	1 452	90	91.5	1.43	281.98	90	90.7	0.95	399.67	0.87
Large size											
Kilbridge	117	22 790	118	118.1	0.32	351.45	118	118	0	639.35	0.08
Warnecke	N/A		145	147.8	1.66	1 061.91	143	144.2	0.63	1 832.84	2.44
Tong	N/A		271	272.67	1.65	2 573.45	268	269.2	1.55	3 313.65	1.27
Wee-Mag	N/A		101	103.1	0.74	1 798.79	99	100	0.72	3 224.32	3.01
Mukherje	N/A		227	228.3	0.95	1 308.91	224	225.4	0.92	1 833.76	1.27

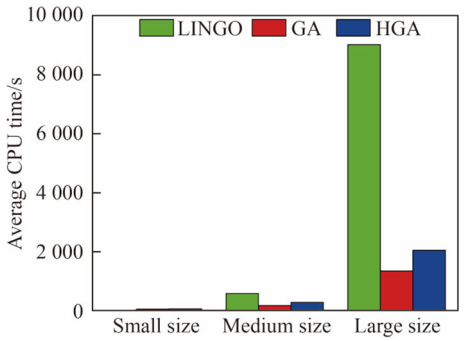

Fig. 8 Average CPU time of benchmark instances

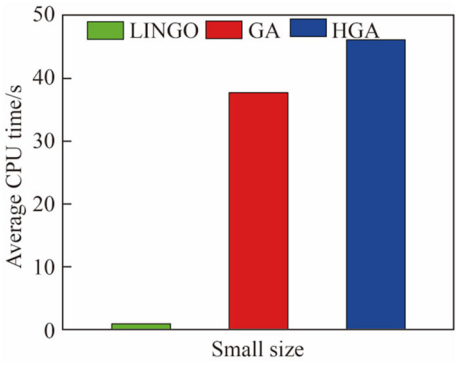

Fig. 9 Average CPU time of small size instances

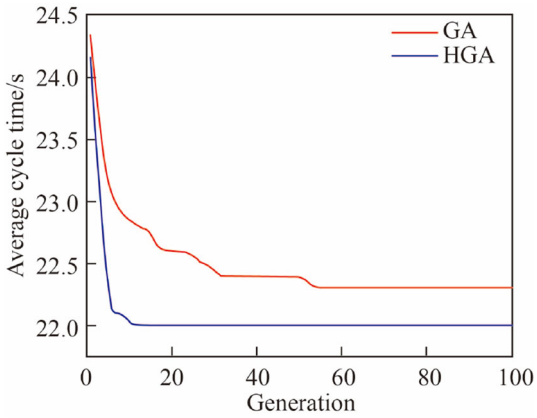

Fig. 10 Convergence behavior of HGA and GA for Rosizeg with γ=1

minimize cycle time under different γ ranging from 1 to 3. The parameters adopted in the algorithm are summarized in Table 9. Each instance is implemented for 10 runs.

4.1 Comparison of different initializations

In the proposed algorithm, 20% initial population is seeded by a heuristic method designed in Sect. 3.1 and 80% initial population is generated randomly. To test its performance, the Sawyer instance with $\gamma = 1$ is initialized by two approaches: the proposed initialization (PI) and random initialization (RI). The results are shown in Table 10. "Min" and "Mean" columns are the minimum and mean values of the solutions with the test instance of ten replications; "SD" column means standard deviation of the solutions. The convergence behaviors of the average cycle time are demonstrated in Fig. 7. As shown in Table 10 and Fig. 7, better solutions and faster convergence are obtained by the proposed initialization.

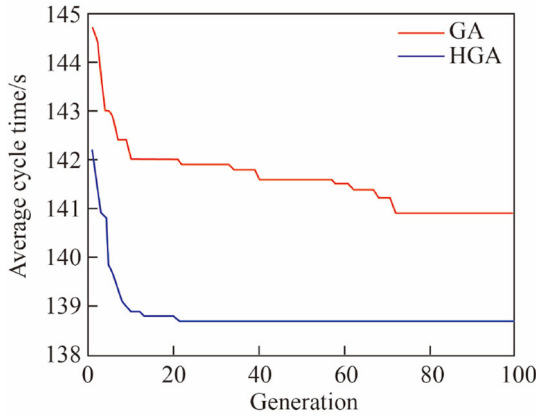

Fig. 11 Convergence behavior of HGA and GA for Warnecke with γ=1

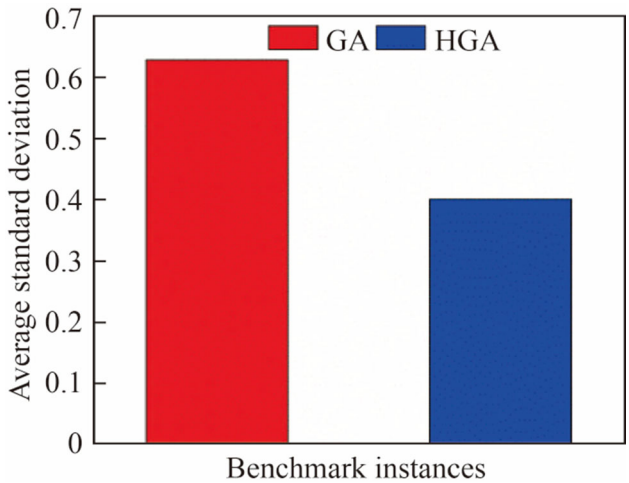

Fig. 12 Average standard deviation of benchmark instances

Table 14 Kruskal-Wallis test for Gap

Algorithm	*N*	Median	Mean rank	*Z* value
GA	150	0.022	178.2	5.53
HGA	150	0	122.8	− 5.53
Overall	300		150.5	
H = 30.58DF = 1P = 0 (Not adjusted for ties)				
H = 35.94DF = 1P = 0 (Adjusted for ties)				

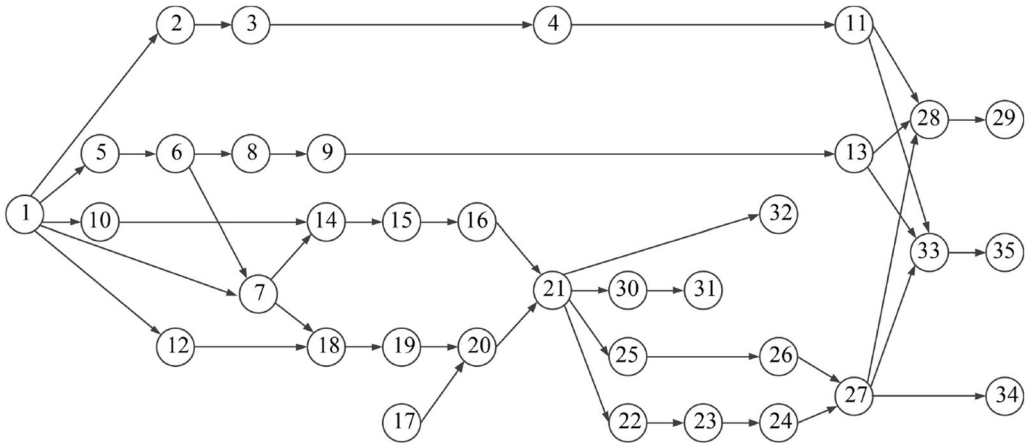

Fig. 13 Combined precedence graph of problem Gunther-35

Table 15 Task times of problem Gunther-35

Tasks	Model 1	Model 2	Model 3	Tasks	Model 1	Model 2	Model 3
1	29	29	29	19	17	21	20
2	3	3	3	20	17	21	20
3	5	5	6	21	10	2	2
4	20	24	22	22	10	10	10
5	10	2	2	23	20	12	10
6	10	18	19	24	20	26	28
7	2	2	2	25	10	0	0
8	9	1	0	26	4	6	7
9	20	24	26	27	4	6	7
10	35	25	20	28	40	40	40
11	23	23	25	29	4	0	0
12	35	25	20	30	3	7	8
13	20	26	25	31	3	7	8
14	4	0	0	32	1	1	1
15	18	20	20	33	38	42	45
16	28	30	32	34	2	2	2
17	0	4	5	35	2	2	2
18	0	4	4				

4.2 Validation studies

The results of the proposed HGA are compared with that of an exact algorithm and the pure GA. The detailed results of 15 instances are given in Tables 11–13. In the tables, the "OCT" column is the optimal solution found by the exact algorithm; "CPU/s" column refers to the average processing time in seconds spent by the algorithms; "Improvement/%" column presents the comparative results of the performance of the HGA with GA.

The branch and bound algorithm embedded in the integer solver software LINGO is used to solve the counterpart of the nonlinear model in Sect. 2.3. For 15 test instances, Warnecke, Tong, Wee-Mag, and Mukherje cannot be solved optimally within 3 days calculation by LINGO.

Each instance is executed ten times using the GA. For comparison, parameters adopted in the GA are set the same values as in the HGA. For the small size instances, both the HGA and GA converge to the same optimal solutions found by the integer solver. For the medium size instances, the HGA gets better solutions for four instances. For the large-size instances, the HGA gains better solutions for all five instances. From the results, the HGA outperforms the GA by 83% for the number of the best solutions and improves the average solutions by 1.13%.

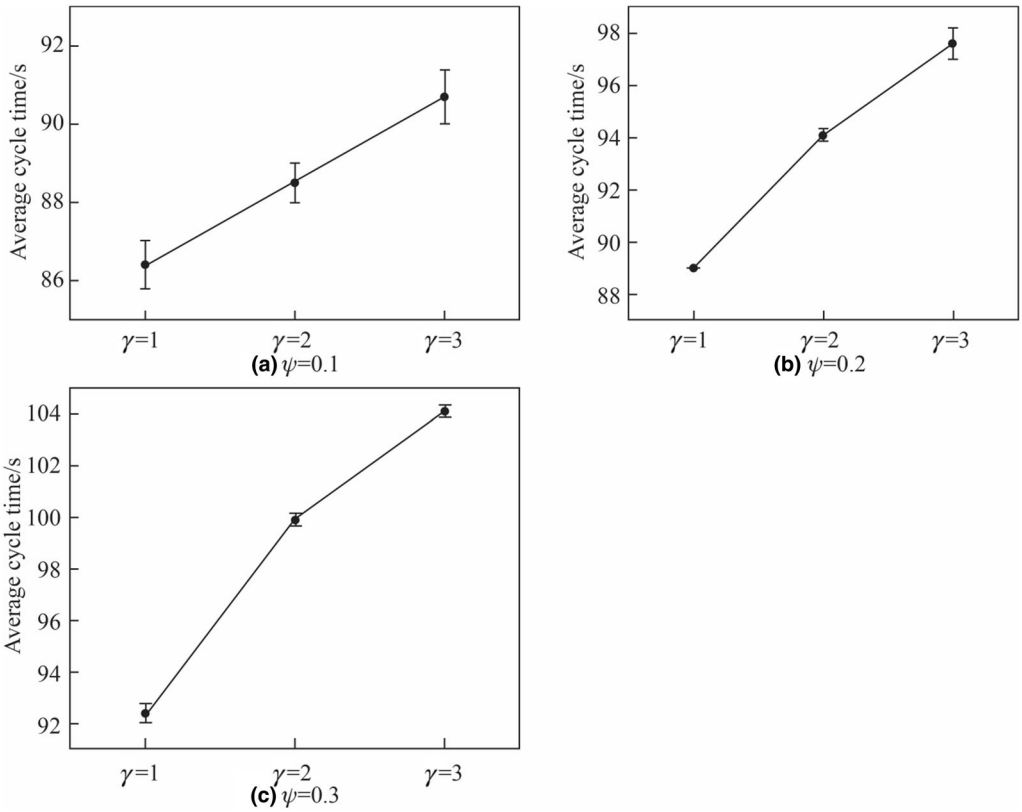

Fig. 14 Variation tendency of the cycle time with different γ for a specific ψ

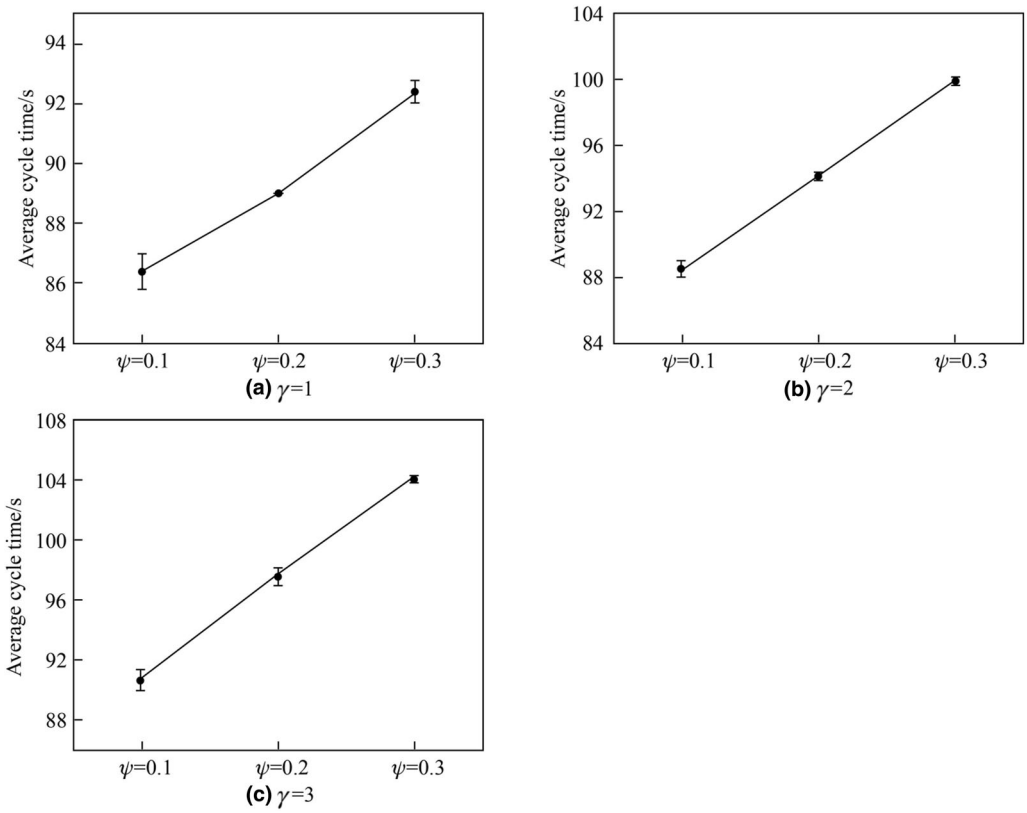

Fig. 15 Variation tendency of the cycle time with different ψ for a specific γ

The average CPU times for different algorithms are described in Fig. 8. Compared with the processing time of the medium and large-size instances, the average CPU time for small size instances is not obvious in Fig. 8. The processing time of small size instances can be seen clearly in Fig. 9. As shown in Fig. 8, the processing time of LINGO increases rapidly with the increase in the size of instances. It indicates that the solution of large-size instances cannot be obtained by LINGO in an acceptable time span. Though they spend more CPU time on small size instances, the GA and HGA spend less CPU time than LINGO on medium and large-size instances. Compared with the processing time of the GA, 22.22%, 69.35%, and 52.66% increases in average CPU times of the HGA are observed, respectively, for small, medium, and large-size instances. Therefore, the HGA obtains better solutions than the GA without causing excessive time consumption.

Figures 10 and 11 demonstrate the convergence behaviors of the HGA and GA for medium and large-size instances. From Figs. 10 and 11, it can be found that the HGA has better initial performance and converges at an earlier generation. This is because of the use of seeded initial population and the adaptive local search procedure in the algorithm loop.

Concerning robustness, the HGA is more robust than the GA. As shown in Fig. 12, the average standard deviation of the HGA is 0.4 and that of the GA is 0.63 for 15 instances. This is because of the reduced risk of premature convergence using the adaptive local search and the discrete Levy flight in the HGA.

To further compare HGA and GA, a statistical test is carried out. The computational results are compared with each other in terms of Gap. Gap is the percentage difference between the optimal cycle time ("OCT" column in Tables 11–13) and the best cycle time ("Min" column in Tables 11–13) and is calculated as Gap = (Min − OCT)/OCT. The lower gap value means a better performance of the algorithm. With the HGA and GA converging to the same optimal solutions for the small size instances and only one instance having the OCT solution among the large size instances, the statistical test is implemented for the medium size instances.

Because the normality of Gap values is violated, the Kruskal-Wallis test is performed. Table 14 shows the results of the Kruskal-Wallis test. Both p values are zero, thereby indicating that there is a statistically significant difference among these two algorithms. The result suggests that the HGA outperforms the GA statistically.

4.3 Illustrative example

A specific example of Gunther-35 is solved by the proposed algorithm. The effect of γ and ψ on the solutions is demonstrated. Its combined precedence graph is shown in Fig. 13, and the nominal task times of 35 tasks for three models are shown in Table 15. For product mix (0.2, 0.3, 0.5), six workstations are available, along with the comparison between the solutions of three different γ (1, 2, 3) and three different ψ (0.1, 0.2, 0.3).

Each Gunther-35 with different combinations of parameters is solved ten times. The interval plots are adopted to show the variation tendency of the average cycle time under different parameters. The average cycle time, as described in Fig. 14, becomes larger with larger γ for a given ψ. As shown in Fig. 15, the larger ψ makes the average cycle time become larger for a specific γ. γ is the number of uncertain tasks considered and ψ represents the task time interval. Both parameters reflect the uncertainty level considered in the problem. Therefore, it can be concluded that the production efficiency will be sacrificed to hedge against uncertainty owing to the cycle time related to the production efficiency of the assembly line.

5 Conclusions

Mixed-model assembly lines are widely used in industries at present. With its practical benefits, the consideration of uncertain task times in the mixed-model assembly balancing problem is important. In this paper, MMALBP-II with interval uncertainty is considered and the robust optimization method is used. The robust model of this problem and its counterpart are formulated. The proposed models are NP-hard. Therefore, an efficient HGA is developed to overcome the computational difficulties in solving large-size problems. Experimental results show that the proposed algorithm is effective and efficient to find solutions for large-size problems. It is also found that the production efficiency will be sacrificed to hedge against uncertainty. For future research, the mixed-model assembly line balancing and sequencing with interval uncertainty could be considered simultaneously.

Acknowledgements This work is supported by the National Science and Technology Major Project of Ministry of Science and Technology of China (Grant No. 2013ZX04012-071) and the Shanghai Municipal Science and Technology Commission (Grant No. 15111105500). The authors also want to express their gratitude to the reviewers. Their suggestions have improved this work.

References

1. Salveson ME (1955) The assembly line balancing problem. J Ind Eng 29(10):55–101
2. Baybars I (1986) A survey of exact algorithms for the simple assembly line balancing problem. Manag Sci 32(8):909–932
3. Boysen N, Fliedner M, Scholl A (2007) A classification of assembly line balancing problems. Eur J Oper Res 183(2):674–693
4. Battaïa O, Dolgui A (2013) A taxonomy of line balancing problems and their solution approaches. Int J Prod Econ 142(2):259–277
5. Simaria AS, Vilarinho PM (2004) A genetic algorithm based approach to the mixed-model assembly line balancing problem of type II. Comput Ind Eng 47(4):391–407
6. Zhang WQ, Xu WT, Liu G et al (2017) An effective hybrid evolutionary algorithm for stochastic multiobjective assembly line balancing problem. J Intell Manuf 28(3):783–790
7. Dong JT, Zhang LX, Xiao TY (2018) A hybrid PSO/SA algorithm for bi-criteria stochastic line balancing with flexible task times and zoning constraints. J Intell Manuf 29(4):737–751
8. Van Hop N (2006) A heuristic solution for fuzzy mixed-model line balancing problem. Eur J Oper Res 168(3):798–810
9. Soyster AL (1973) Convex programming with set-inclusive constraints and applications to inexact linear programming. Oper Res 21(5):1154–1157
10. Bertsimas D, Sim M (2004) The price of robustness. Oper Res 52(1):35–53
11. Al-e-hashem SMJM, Aryanezhad MB, Malekly H et al (2009) Mixed model assembly line balancing problem under uncertainty. In: Paper presented at the 2009 international conference on computers and industrial engineering, pp 233–238
12. Hazır Ö, Dolgui A (2013) Assembly line balancing under uncertainty: robust optimization models and exact solution method. Comput Ind Eng 65(2):261–267
13. Gurevsky E, Hazır Ö, Battaïa O et al (2013) Robust balancing of straight assembly lines with interval task times. J Oper Res Soc 64(11):1607–1613
14. Nazarian E, Ko J (2013) Robust manufacturing line design with controlled moderate robustness in bottleneck buffer time to manage stochastic inter-task times. J Manuf Syst 32(2):382–391
15. Moreira MCO, Cordeau JF, Costa AM et al (2015) Robust assembly line balancing with heterogeneous workers. Comput Ind Eng 88:254–263
16. Hazır Ö, Dolgui A (2015) A decomposition based solution algorithm for U-type assembly line balancing with interval data. Comput Oper Res 59:126–131
17. Pereira J, Álvarez-Miranda E (2018) An exact approach for the robust assembly line balancing problem. Omega 78:85–98
18. Tasan SO, Tunali S (2008) A review of the current applications of genetic algorithms in assembly line balancing. J Intell Manuf 19(1):49–69
19. Saavedra-Moreno B, Salcedo-Sanz S, Paniagua-Tineo A et al (2011) Seeding evolutionary algorithms with heuristics for optimal wind turbines positioning in wind farms. Renew Energ 36(11):2838–2844
20. Osaba E, Carballedo R, Diaz F et al (2014) On the influence of using initialization functions on genetic algorithms solving combinatorial optimization problems: a first study on the TSP. In: IEEE Conference on Evolving and Adaptive Intelligent Systems
21. Oman S, Cunningham P (2001) Using case retrieval to seed genetic algorithms. Int J Comput Intell Appl 1(1):71–82
22. Sewell EC, Jacobson SH (2012) A branch, bound, and remember algorithm for the simple assembly line balancing problem. INFORMS J Comput 24(3):433–442
23. Hoffmann TR (1963) Assembly line balancing with a precedence matrix. Manag Sci 9(4):551–562
24. Goldberg DE, Korb B, Deb K (1989) Messy genetic algorithms: motivation, analysis, and first results. Complex Syst 3(3):493–530
25. Rubinovitz J, Levitin G (1995) Genetic algorithm for assembly line balancing. Int J Prod Econ 41(1–3):343–354
26. Leu YY, Matheson LA, Rees LP (1994) Assembly-line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria. Decis Sci 25(4):581–606
27. Cheng RW, Gen M, Tsujimura Y (1999) A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Comput Ind Eng 36(2):343–364
28. Triki H, Mellouli A, Hachicha W et al (2016) A hybrid genetic algorithm approach for solving an extension of assembly line balancing problem. Int J Comput Integr Manuf 29(5):504–519
29. Yun Y, Chung H, Moon C (2013) Hybrid genetic algorithm approach for precedence-constrained sequencing problem. Comput Ind Eng 65(1):137–147
30. Miltenburg J (2002) Balancing and scheduling mixed-model U-shaped production lines. Int J Flex Manuf Syst 14(2):119–151
31. Li HP, Zhang SQ, Zhang C et al (2017) A novel unsupervised Levy flight particle swarm optimization (ULPSO) method for multispectral remote-sensing image classification. Int J Remote Sens 38(23):6970–6992
32. Li L, Zhang Z, Guan C et al (2018) Multi-objective optimization for partial disassembly line balancing with goal-driven discrete cuckoo search. J Comput Aid Des Comput Graph 30(4):681–694