Heuristic Algorithm for Type II Two-sided Assembly Line Rebalancing Problem with Multi-objective

Yahui Zhang, Xiaofeng Hu[a] and Chuanxun Wu

School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract. In practical production, the balance of the assembly lines is always destroyed by many factors, e.g., demand fluctuations, disruptions caused by workstation breakdowns or shutdowns, and changes of the work force. Therefore, this paper addresses the type II rebalancing problem for the two-sided assembly lines (TALRBP-II), which are widely utilized to assemble large-size high-volume products. As we all know that TALBP-II (two-sided assembly line balancing problem type-II) is NP-hard, TALRBP-II is its extension and much more complex. In order to solve this problem, a mathematical model with the objectives of minimizing cycle time and rebalancing cost is proposed. Constraint programming (CP) method is applied to optimize the model and a workstation-oriented heuristic algorithm is designed to solve the problem. An example is presented to illustrate the procedure of the proposed algorithm, and the best solution obtained verify the efficiency of the proposed algorithm.

1 Introduction

Two-sided assembly line (TAL, Figure 1) is a typical production organization form and is widely used in assembling large-sized high-volume products, such as cars, trucks, buses, shovel loader. In the TAL, operators work in parallel on the two sides (left-side (L) and right-side (R)) of the conveyor belt, and a pair of stations (e.g. stations 3 and 4) on the two sides is called as a mated-station or a workstation, and each one is called a companion for the other. TAL balancing problem (TALBP) is one of the most important problems in the field of production planning and management. TALBP is assigning tasks to workstation to optimize one or more objectives with some constraints[1].

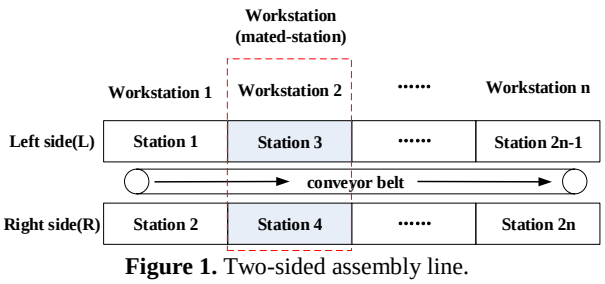

Figure 1. Two-sided assembly line.

TALBPs have received more attention in the literature since Bartholdi [2]. According to the different of objective, TALBP can be categorized into three types [3]: the type I is mainly to minimize the number of stations with a given cycle time, type II is looking for the minimum cycle time with a predetermined number of stations, while type III focus on optimizing smoothness and efficiency of an AL. Especially, TALBP type-II (TALBP-II) is of major importance for the reconfiguration of the ALs [4]. However, many factors are ignored, such as rebalancing cost. In addition, as mentioned in [5], in industrial practices, it is hardly ever to build a new AL to meet all the need arising from changes of production and the majority of AL balance is conducted for reconfiguration instead of first time installation, i.e. rebalancing problem.

Actually, assembly line rebalancing problem (ALRBP) has been mentioned by numerous researchers since Schofield [6], however, related theories and methods research are considerably limited. Gamberini et al.[7-8], Makssoud et al. [9] and Sancı and Azizoğlu [10] proposed heuristic algorithms to deal with the single-model and one-sided ALRBP. Altemeier et al. [11], Grangeon et al.[12] and Yang et al. [13] did research on the mixed-model ALRBP. Zha and Yu [14] and Celik et al. [15] focused on U-lines rebalancing problem. Zhang et al. [16] presented a modified non-dominated sorting genetic algorithm II (MNSGA-II) for a real-life multi-objective two-sided assembly line rebalancing problem (MTALRBP) with modifications of production demand, line's structure and production process. The objectives are to minimize the cycle time and rebalancing costs synchronously. Obviously, there are only one work deals with TALRBP.

In this paper, a workstation-oriented heuristic algorithm is proposed to tackle the TALRBP-II with multi-objectives, including minimizing cycle time and rebalancing cost. The rest of this paper is organized as follows. In Section 2, a mathematical formulation for TALRBP-II is presented. The proposed algorithm is proposed in Section 3, Section 4 is the numerical

* Corresponding author: wshxf@sjtu.edu.cn

example to demonstrate proposed algorithm. Conclusion is presented in Section 5.

2 Mathematical formulation

2.1 Notations

2.1.1 Parameters

I: Set of tasks, I={1, 2, ... , i, ... , M}.

J: Workstations (mated-stations), J = {1, 2, ... , j, ... , N}.

(j,k): A station of workstation j and its operation direction k, k=1indicates the left side and k=2 indicates the right side.

x_{ijk}^0: 1, if task i is assigned to station (j,k) in the initial balancing; 0, otherwise.

y_{ij}^0: 1, if task i is assigned to workstation j in the initial balancing; 0, otherwise.

$P(i)$*: Set of immediate predecessors of task i.

$P(i)$: Set of all predecessors of task i.

P_0: Set of tasks that have no immediate predecessors; i.e., P_0={$P(i)$*=∅,for i=1,2,···M}.

$S(i)$*: Set of immediate successors of task i.

$S(i)$: Set of all successors of task i.

t_i: Processing time of task i.

μ: A very large positive number.

α,β :Coefficient of transferred tasks between workstations and within workstations, respectively.

A_L: Set of tasks that must be performed on the left side of the line, $A_L \subset I$.

A_R: Set of tasks that must be performed on the right side of the line, $A_R \subset I$.

A_E: Set of tasks that can be performed on either side of the line, $A_E \subset I$.

$C(i)$: Set of tasks whose operation directions are opposite to task i's operation direction, C(i)= A_L, $i \subset A_R$; C(i)=A_R, $i \subset A_L$; $C(i)=\Phi$, $i \subset A_E$.

$K(i)$: Set of indicating the preferred operation directions of task i, $K(i)$={1}, $i \subset A_L$; $K(i)$={2}, $i \subset A_R$; $K(i)$={1, 2}, $i \subset A_E$.

2.1.2 Variables

x_{ijk}: 1, if task i is assigned to station (j,k); 0, otherwise.

y_{ij}: 1, if task i is assigned to workstation j; 0, otherwise.

t_i^s: Start time of task i.

t_i^f: Finish time of task i, $t_i^f = t_i^s + t_i$.

ct: Cycle time

z_{ip}: Indicator variable. z_{ip}=1, if task i is assigned earlier than task p in the same station; z_{ip} =0, if task p is assigned earlier than task i in the same station.

2.2 Mathematics model

$$\text{Min} \quad ct \tag{1}$$

$$\text{Min} \ \alpha \sum_{i\in I}\sum_{j\in J}\left|y_{ij}-y_{ij}^0\right| + \beta \sum_{i\in I}\sum_{j\in J}\sum_{k=1}^{2}\left(\left|x_{ijk}-x_{ijk}^0\right|-\left|y_{ij}-y_{ij}^0\right|\right) \tag{2}$$

$$\sum_{j\in J}\sum_{k\in K(i)} x_{ijk} = 1, \forall i \in I \tag{3}$$

$$\sum_{g\in J}\sum_{k\in K(i)} g x_{hfk} - \sum_{j\in J}\sum_{k\in K(i)} j x_{ijk} \le 0, \forall i \in I - P_o, h \in P(i)^* \tag{4}$$

$$t_i^f \le ct, \forall i \in I \tag{5}$$

$$t_i^f - t_h^f + \mu\left(1-\sum_{k\in K(h)} x_{hjk}\right) + \mu\left(1-\sum_{k\in K(i)} x_{ijk}\right) \ge t_i, \quad \forall i \in I - P_0, h \in P(i), j \in J \tag{6}$$

$$t_p^f - t_i^f + \mu(1-x_{pjk}) + \mu(1-x_{ijk}) + \mu(1-z_{ip}) \ge t_p, \quad \forall i \in I, p \in \left\{r \mid r \in I - \left(P(i)^* \cup S(i)^* \cup C(i)\right), i < r\right\} \quad j \in J, k \in K(i) \cup K(p) \tag{7}$$

$$t_i^f - t_p^f + \mu(1-x_{pjk}) + \mu(1-x_{ijk}) + \mu z_{ip} \ge t_i, \quad \forall i \in I, p \in \left\{r \mid r \in I - \left(P(i)^* \cup S(i)^* \cup C(i)\right), i < r\right\} \quad j \in J, k \in K(i) \cup K(p) \tag{8}$$

$$x_{ij1} = \{0,1\}, \forall i \in A_L, j \in J \tag{9}$$

$$x_{ij2} = \{0,1\}, \forall i \in A_R, j \in J \tag{10}$$

$$x_{ijk} = \{0,1\}, \forall i \in A_E, j \in J, k = 1,2 \tag{11}$$

$$z_{ip} = \{0,1\}, \forall i \in I, p \in \left\{r \mid r \in I - \left(P(i)^* \cup S(i)^* \cup C(i)\right), i < r\right\} \tag{12}$$

$$t_i^f \ge t_i, \forall i \in I \tag{13}$$

Eqs. (1) and (2) are objective functions, represent minimizing cycle time and the rebalancing cost, i.e., the number of reassigned tasks, respectively. Eqs. (3) ~ (5) indicate that the constraints of tasks cannot be split, precedence relationship, and cycle time. Eqs. (6) ~ (8) are constraints specific to the two-sided assembly line, indicating that the start time of task i is decided by the maximum value between the completion time of the last immediate predecessors task h on its mated station and the previous task p on the station itself, i.e., $t_i^s \ge \max\{t_h^f, t_p^f\}$. Eqs. (9)~(13) define each variable.

2.3 Multi-objective programming

Constraint programming (CP) [17] is one of classical multi-objective decision (MOD) methods. Compared with other MOD methods, it has the advantages of obtaining Pareto solution sets efficiently, lessening restriction of additional parameters and avoiding uniform dimensions. The principle is selecting one of the multiple objectives as the only objective and the other objectives are considered as constraints. In this paper,

the proposed mathematical model can be translated to the new model as follow.

$$\text{Min} \quad ct \tag{1}$$

Subject to：

Eqs.（3）~（13）

$$\sum_{i\in I}\sum_{j\in J}\alpha\left|y_{ij}-y_{ij}^{0}\right|+\sum_{i\in I}\sum_{j\in J}\sum_{k=1}^{2}\beta\left(\left|x_{ijk}-x_{ijk}^{0}\right|-\left|y_{ij}-y_{ij}^{0}\right|\right)\le\varepsilon_R,$$
$$\forall\varepsilon_R\in\left[\underline{\varepsilon_R},\overline{\varepsilon_R}\right] \tag{14}$$

ε_R is the maximum rebalancing cost accpeted on the AL, $\underline{\varepsilon_R}$ and $\overline{\varepsilon_R}$ are its upper and lower bound.

3 The proposed algorithm

3.1. The main procedure of WOHA

Assume that the number of workstations is fixed to *N*, and the theoretical minimum cycle time can be indicated as $\overline{ct}=\sum_{i=1}^{M}t_i/2N$. Take the nearest integer of $\overline{ct}$ as the target cycle time (Tct), $Tct=\left\lceil\sum_{i=1}^{M}t_i/2N\right\rceil$. The main procedure of the workstation-oriented heuristic algorithm (WOHA) is presented below.

Step 1. Set the initial cycle time *ct* =*Tct*.

Step 2. Select the initial workstation *j* = 1.

Step 3. Get the solutions set SU_j of transferred tasks for *j*:

(1) Identify the set TT_j of tasks that can transfer into *j*, and the set TO_j of tasks that can transfer out of *j* without violating the precedence constraints.

(2) Build the solutions set SU_j of the tasks combination of set TT_j, TO_j, i.e. $SU_j = TT_j \cup TO_j$.

Step 4. Screen solutions set： select the solutions with the cycle time and idle time constraints to get the possible solutions set PS_j.

①The cycle time constraint means the operation time of tasks which contain inevitable wait time(*IWT*), could not be greater than *Tct*, as shown in Eq. (15). S_{jk} represents the set of tasks assigned to station (*j*, *k*).

$$\sum_{i\in S_{jk}}(t_i+IWT_i)\le Tct,\forall j\in J,k=1,2 \tag{15}$$

For a task *i*, the inevitable wait time (*IWT*) is the maximum value of zero and the difference between the processing time of the task and the total operation time of its matched tasks. It can defined as follows:

$$IWT_i=\max\left\{0,t_i-\sum_{l\in MT_i}t_l\right\} \tag{16}$$

MT_i represents the matched tasks of task *i*, which can be processed with task *i* synchronously, i.e., they are not predecessor or successor of task *i* and can be assigned to different direction from task *i*.

② The idle time constraint means the idle time contained in the possible solution of *j* must be less than the maximum idle time allowed by the assembly line (*MIDAL*). It can be calculated by the formulation:

$$MIDAL=2N*Tct-\sum_{i=1}^{M}t_i \tag{17}$$

Step 5. Rank all the possible solutions in *PS*.

In order to rank all the possible solutions, we first given an evaluation function *CL*:

$$CL=x*\left(\overline{ct}-3\right)+\left(1-x\right)*Tct \tag{18}$$

And then, we will get different rank rules for the possible solution according to the comparison between the possible solution and *CL*:

① The first level rule: the possible solution ≥ *CL*, then it prefers to choose the solution which has the minimum number of moved tasks; and if all the possible solution has the same number of moved tasks, rank them using station time in the descending order.

② The second level rule: If possible solution <*CL*, then it prefers to choose the solution which has the maximum station time; and if all the possible solution has the same station time, rank them using the number of moved tasks in the ascending order.

③ The first level rule always has a higher priority than the second level rule.

Note: The mentioned number of moved tasks here not just refer to current workstation, but involved all the workstations which have already been assigned tasks currently. It is obvious that the less number of transferred tasks of current workstation does not mean that the final total number of moved tasks will be less.

Step 6. Select the first possible solution *r* in the order of the possible solutions got in the step 5.

Step 7. Assign tasks of *r* according to the single workstation tasks assignment procedure (*SWTA*) and get the completion time of *j*.

Step 8. If the completion time of *j* ≤ *Tct*, save it as a feasible solution for *j*, and go to Step 9, otherwise, Remove it from *PS*, *PS*=*PS*-{*r*}, go to Step 6.

Step9. Set *j*=*j*+1, if *j* ≤ *n*, go to Step 3.

Step10. If all the tasks are assigned, terminate and record ct as a solution value. Otherwise, set *ct* =*ct*+*θ*, and go to Step 2.

3.2 Procedure of SWTA

Step1. Categorize the tasks of workstation *j* into three set *IL*, *IR*, *NT*, according to their initial position. The *IL* represents the tasks that be assigned to the left station of *j* in the initial balanced line, while the *IR* is the opposite. The *NT* stands for the new tasks moved into *j*.

Step2. If *NT* is not empty, go to step 3. Otherwise, go to step 4.

Step3. Tasks pre-assignment: if the task is not E (L or R) type, assign it to set *IL* or *IR*, respectively; otherwise, rank the E type tasks according to their operation time and compute the completion time of each side which includes the inevitable wait time. Record them as *LLT* and *RRT*, separately. Then assign the E-type tasks as the order to the smaller side in turn, and if the *LLT*=*RRT*, assign the task to the L side. Repeat it unless the E type tasks have been assigned completely.

Step4. Update information of tasks on current workstation, and get the potential tasks exchange solutions (*PTES*) between the left and right side of the workstation according to the number of exchange tasks in the descending order.

Step5. Select the first solution *h* of *PTES*.

Step6. Mark the tasks of left station as *L*, the tasks of right station as *R*.

Step7. Get the assignment sequence of tasks according to the weights and the related rules.

Weight 1: processing time of task * (the number of its immediate successors on current workstation +1). Rank all the remaining tasks according weight 1 in the descending order.

Weight 2: task *ID*. Rank all the remaining tasks according weight 2 in ascending order

Weight 3: the number of mated tasks.Rank all the remaining tasks according weight 3 in ascending order.

Weight 4: random.

Step8. Assign tasks sequentially, and calculate the completion time of *j* and the generated idle time between tasks after a task is assigned each time. If the completion time is greater than *Tct*, delete solution *h* from *PTES*, *PTES*= *PTES*-{*h*}, go to step 5; otherwise, go to step 9.

Step9. If the idle time generated exceeds *MITALAP* and *MITCWAP*, delete solution *h* from *PTES*, *PTES*= *PTES*-{*h*}, go to step 5; otherwise go to step 8.

MITALAP is the expression of the maximum idle time allowed by the assembly line at present, it can be obtained by the difference between *MIDAL* and the total idle time contained in the previous workstations.

MITCWAP means the maximum idle time allowed by current workstation at present and can be computed as following:

$$MIDCWAP = 2*Tct - \sum_{i \in w_j} (t_i + IWT_i) \qquad (19)$$

w_j: the set of tasks assigned to workstation *j*.

Step10. If tasks contained in solution *h* have been assigned completely, stop and save it as a feasible solution for workstation *j*.

4 Numerical example

This section presents a numerical example to explain and verify the effectiveness of the proposed algorithm. For this purpose, the example of P24 [18] is considered. Figure 2 is the precedence diagram of tasks in P24 and the Figure 3 is an initial solution of P24 obtained by Hoffman algorithm [19], of which the cycle time is fixed with 22 units of times. The target CT is 18. The proposed algorithm is programmed in C# and executed on a PC with Intel Core(TM) i5, 3.20GHz, 8 G RAM. One of the iterations of the procedure (i.e., one workstation) is given below.

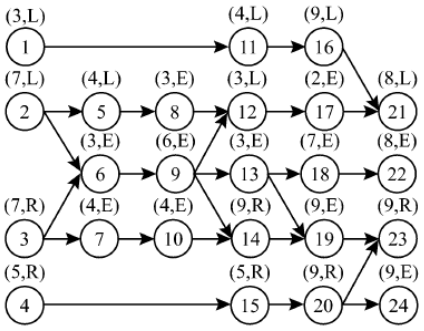

Figure 2. The precedence relations among tasks of P24.

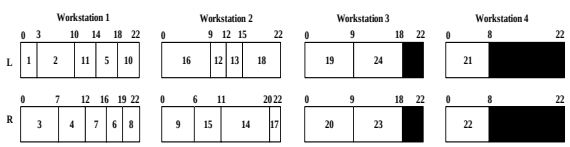

Figure 3. The initial balanced assembly line of P24.

At first, choose workstation 1 as current workstation.

Second, get the set *R* of possible solutions of transferred tasks for this workstation.

(1) divide all the tasks into two groups: group-1 contains the tasks on the current workstation, i.e., {1,2,3,4,5,6,7,8,10,11}; the other rest tasks belong to group-2.

(2) Get and record the tasks which can move into / out workstation 1 from this two group, separately. At last, tasks which can be moved into are {4, 6, 8, 10, 11} and can be moved out are {9, 15, 16}.

(3) Calculate the inevitable wait time for all tasks on workstation 1. Take task 1 for example, its mated tasks are {3, 4, 5, 6, 7, 8, 10}, there is no doubt that its inevitable wait time is 0. The other tasks' are 0, too. So, its workstation time is equal to the processing time of all the tasks on it, 44.

(4) Get all the possible solutions of transferred tasks. Eventually, we got 101 possible solutions, there were 5 solutions only had one moved tasks, 25 solutions involved two transferred tasks and 71 solutions own three transferred tasks.

Third, screen all the possible solutions according to the difference between the workstation time and target CT, as well as the difference between the idle time contained in the possible solution and the maximum idle time which is allowed by the AL. Finally, there are 27 possible solutions are retained : {10,11}, {10,7}, {6,4}, {8,4},
{4,11},{4,10},{4,11,1},{6,11,1},{6,10,11},{6,10,7},{6,4,11},{6,4,10},{8,11,1},{8,5,11},{8,10,11},{8,10,5},{8,10,7},{8,4,11},{8,4,5},{8,4,10},{8,6,11},{8,6,5},{8,6,10},{8,6,4},{10,11,1},{10,7,11}.

Fourth, rank all the possible solutions, according to Eq.(18).

Fifth, assign tasks of the first solution ({10,11}, which means moving out taks 10 and 11).

(1) Classify tasks according to their operation directions to three groups: LL, RR, EE(LE+RE). LL involves all the L type tasks ({1,2,5}); RR involves all the R type tasks({3,4,7,6,8}), EE involves all the E type

tasks({6,7,8}) no matter which side it had been assigned to.

(2) Search all the *PTES*.

There is no doubt that only E type tasks can be transferred between left and right side, so we just need search from group EE({6,7,8}), and *PTES* contains: {0}, {7}, {6}, {8}, {7,6}, {7,8}, {6,8}, {7,6,8}.

Note: the potential tasks exchange solutions only contain the transferred tasks; {0} means no tasks transfer; all the solutions have been already sequenced as their priority, so we just need execute them in turn in the next step.

(3) Choose the final executed solution. In our case, we choose the second solution ({7}, which means transfer task 7 from right to left).

(4) Remark the types of tasks according to the side to which they are assigned at percent. In our case, tasks have been assigned to the left station are {1,2,5,7}, so remarked them as L type, while {3,4,6,8} are remarked as R type.

Note, there are no tasks moved into, so we do not need assign the moved into tasks simply.

(5) Get the allocation sequence of tasks according to the weights and their rules, and allocate them sequentially until the last one is allocated.

Use weights one by one according to the order. At first, we get the value of weight 1 of all tasks involved on this workstation, and then rank them in the descending order, as shown in Table 1.

Table 1. Weight 1 of tasks for the first workstation.

Order	Weight 1	Task ID
1	21	2,3
2	8	5
3	5	4
4	4	7
5	3	1,6,8

It is clear that, tasks could not get their unique order just using weight 1, so we need weight 2. The result is showed in Table 2. And then we can allocate them as this order, the procedure is depicted in Figure 4. We should note that, at the process of allocating tasks to workstation, if the order violates the precedence relation among tasks, it should change.

Table 2. The unique order of tasks for the first workstation.

Order	Weight 1	Tasks ID
1	21	2
2	21	3
3	8	5
4	5	4
5	4	7
6	3	1
7	3	6
8	3	8

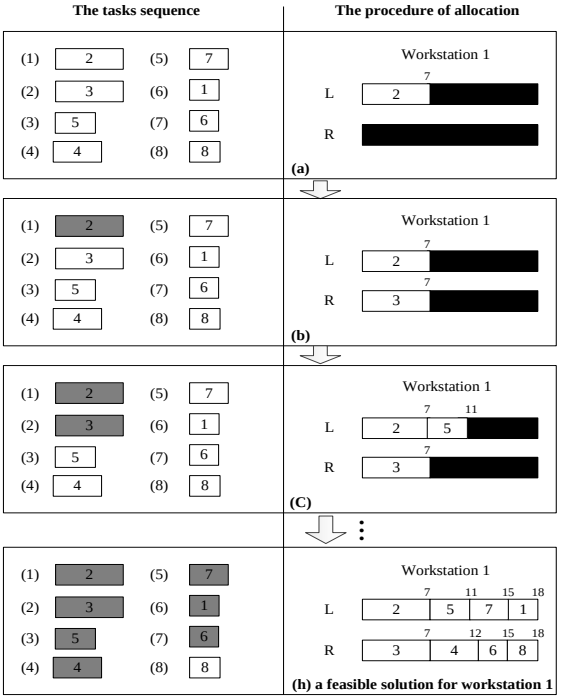

Figure 4. The procedure of tasks' allocation according to weight.

Do the above process to the other workstations successively and the final rebalance solution is showed in Figure 5. The task adjustment is shown in Table 3 and the total rebalancing cost is 11(α,β are assumed be 1 and 2).

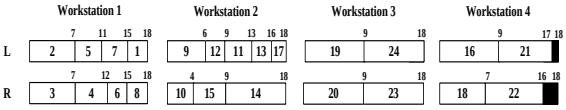

Figure 5. A rebalance solution of P24.

Table 3. Tasks adjustment in rebalancing assembly line.

Task	Initial	Rebalanced solution	Type of tasks' move	Cost
1	(1,1)	(1,1)	-	0
2	(1,1)	(1,1)	-	0
3	(1,2)	(1,2)	-	0
4	(1,2)	(1,2)	-	0
5	(1,1)	(1,1)	-	0
6	(1,2)	(1,2)	-	0
7	(1,2)	(1,1)	WW	1
8	(1,2)	(1,2)	-	0
9	(2,2)	(2,1)	WW	1
10	(1,1)	(2,2)	BW	2
11	(1,1)	(2,1)	BW	2
12	(2,1)	(2,1)	-	0
13	(2,1)	(2,1)	-	0
14	(2,2)	(2,2)	-	0
15	(2,2)	(2,2)	-	0
16	(2,1)	(4,1)	BW	2
17	(2,2)	(2,1)	WW	1
18	(2,1)	(4,2)	BW	2
19	(3,1)	(3,1)	-	0

20	(3,2)	(3,2)	-	0
21	(4,1)	(4,1)	-	0
22	(4,2)	(4,2)	-	0
23	(3,2)	(3,2)	-	0
24	(3,1)	(3,1)	-	0

WW- task move within workstation; BW-task move between workstation.

5 Conclusion

In this paper, a mathematical model for multi-objective two-sided assembly line rebalancing problem type II (TALRBP-II) is proposed to minimize cycle time and rebalancing cost. Constraint programming (CP) method is applied to optimize the model and a workstation-oriented heuristic algorithm is designed to solve the problem. An example is presented to illustrate the procedure of the proposed algorithm, and the best solution obtained verify the efficiency of the proposed algorithm.

For future work, we might develop a new algorithm or extend the proposed algorithm for the more realistic two-sided assembly line rebalancing problem by considering constaints of stochastic processing time, mixed models, space and resources, etc.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (Grant No. 51475303)

References

1. X.F. Hu, E.F. Wu, J.S. Bao, Y. Jin. (2010). A branch-and-bound algorithm to minimize the line length of a two-sided assembly line. Eur. J. Oper. Res. **206**(3):703–7.
2. J. Bartholdi. (1993). Balancing two-sided assembly lines: a case study. Int. J. Prod. Res. **31**(10):2447–61.
3. E.F. Wu, Y. Jin, J.S. Bao, X.F. Hu. (2008). A branch-and-bound algorithm for two-sided assembly line balancing. Int. J. Adv. Manuf. Technol. **39**:1009–1015.
4. Z. Li, Q.Tang, L.P. Zhang. (2016). Minimizing the Cycle Time in Two-Sided Assembly Lines with Assignment Restrictions: Improvements and a Simple Algorithm. Math. Prob. Eng. vol. **2016**, Article ID 4536426, 15 pages.
5. E. Falkenauer. (2005) Line balancing in the real world. The International Conference on Product Lifecycle Management, Lumiere Univeristy of Lyon, France, July 1, pp 360-370.
6. N. A. Schofield. (1979). Assembly line balancing and the application of computer techniques. Comput. Ind. Eng. **3**(1), 53–69.
7. R. Gamberini, A. Grassi, B. Rimini. (2006). A New Multi-objective Heuristic Algorithm for Solving the Stochastic Assembly Line Re-balancing Problem. Int. J. Prod. Res. **102**(2):226-243.
8. R. Gamberini, E. Gebennini, A. Grassi, A. Regattieri. (2009). A multiple single-pass heuristic algorithm solving the stochastic assembly line rebalancing problem. Int. J. Prod. Res. **47**(8): 2141-2164.
9. F.Makssoud, O. Battaïa , A. Dolgui , K. Mpofu , O. Olabanji. (2015). Re-balancing problem for assembly lines: new mathematical model and exact solution method. Assembly. Autom. **35**(1): 16-21.
10. E. Sancı, M. Azizoğlu. (2017). Rebalancing the assembly lines: exact solution approaches [J]. Int. J. Prod. Res. **55**(20): 5991–6010.
11. S. Altemeier, M. Helmdach, A. Koberstein, W. Dangelmaier. (2009). Reconfiguration of Assembly Lines under the Influence of High Product Variety in the Automotive Industry-A Decision Support System. Int. J. Prod. Res. **48** (21): 6235–6256.
12. N. Grangeon, P. Leclaire, S. Norre. (2011). Heuristics for the re-balancing of a vehicle assembly line. Int. J. Prod. Res. **49**(22): 6609-6628.
13. C. J. Yang, J. Gao, L. Y. Sun. (2013). A Multi-objective Genetic Algorithm for Mixed-model Assembly Line Rebalancing. Comput. Ind. Eng. **65**(1):109-116.
14. J. Zha, J. J. Yu. (2014). A Hybrid Ant Colony Algorithm for U-Line Balancing and Rebalancing in Just-in-Time Production Environment. J. Manuf. Syst. **33** (1): 93–102.
15. E. Celik, Y. Kara, Y. Atasagun. (2014). A new approach for rebalancing of U-lines with stochastic task times using ant colony optimisation algorithm. Int. J. Prod. Res. **52**(24): 7262-7275.
16. Y.H. Zhang, X.F. Hu, C.X. Wu. (2017). A modified multi-objective genetic algorithm for two-sided assembly line re-balancing problem of a shovel loader. Int. J. Prod. Res. (4) :1-21.
17. Y.Y. Haimes, L.S.Lasdon, D.A. Wismer. (1971). On a Bi-criterion Formulation of the Problems of Integrated System Identification and System Optimization. IEEE. T. Syst. Man. Cy-S. **1**:296-297.
18. Y.K. Kim, Y.H. Kim, Y.J. Kim. (2000). Two-sided Assembly Line Balancing: A Genetic Algorithm Approach. Prod. Plan. Control. **11**(1):44–53.
19. T.R. Hoffmann. (1963). Assembly line balancing with a precedence matrix. Manage. Sci. **9**: 551–562.