


Multi-neighborhood based path relinking for two-sided assembly line balancing problem

Zhaoyang Yang^{1,2}  · Guojun Zhang¹ · Haiping Zhu¹

Published online: 23 September 2015
© Springer Science+Business Media New York 2015

Abstract This paper presents a multi-neighborhood based path relinking algorithm (MN-PR) for solving the two-sided assembly line balancing problem. By incorporating an effective local search into a path relinking framework, the proposed MN-PR algorithm integrates a number of distinguishing features, such as a multi-neighborhood based local search procedure, a dedicated path relinking operator to generate new solutions and a strategy to fix an infeasible solution generated by the path relinking procedure to a feasible one. Our proposed MN-PR algorithm is tested on a set of totally 45 public instances widely used in the literature. Comparisons with other reference algorithms show the efficacy of the proposed algorithm in terms of the solution quality. Particularly, the proposed MN-PR algorithm is able to improve the best upper bounds for one instance with 65 tasks and 326 cycle time. This paper also presents an analysis to show the significance of the main components of the proposed algorithm.

Keywords Two-sided assembly line balancing problem · Local search · Path relinking · Multi-neighborhood search

1 Introduction

The two-sided assembly lines appear commonly in plants which produce large-sized products, such as automobiles and domestic products, replacing traditional one-sided

✉ Zhaoyang Yang
y_26@163.com

¹ State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science and Technology, Wuhan 430074, People's Republic of China

² School of Automotive Engineering, Hubei University of Automotive Technology, Shiyan 442002, People's Republic of China

assembly lines. They use both (left and right) sides of the line in parallel, making it possible to decrease the line length, reduce the throughput time, cut down the cost of tools and fixtures, and lessen material handling.

The two-sided assembly line balancing problem (TALBP) belongs to the class of job scheduling problems (Huang and Yin 2004; Huang and Wang 2006) and has proven to be NP-hard (Scholl and Becker 2006). Thus, there does not exist polynomial time exact algorithm for solving it unless $P = NP$. During the last two decades, a large number of researchers have extensively studied the TALBP problem. For example, Bartholdi first studied the TALBP problem in 1993, considering minimizing the number of stations as the objective based on a simple assignment rule (Bartholdi 1993). Lee et al. (2001) developed a group assignment procedure for TALBP, assigning task groups instead of individual tasks with the objective of maximizing work relatedness and work slackness. Hu et al. developed a station-oriented enumerative assignment procedure based on the Hoffman heuristic to solve the TALBP (Hu et al. 2008). The experimental results verified that the proposed procedure performed well. Özcan and Toklu (2009c) presented a pre-emptive goal programming model for precise goals and fuzzy goals, respectively, for TALBP. The proposed mathematical model aims to minimize the number of mated-stations and the secondary objective is to minimize the number of stations for a given cycle time with zoning constraints.

Meanwhile, metaheuristic based heuristics have shown to be a feasible way to find high quality solutions with a reasonable time for the TALBP. Simaria and Vilarinho proposed an ant colony optimization algorithm to address the mixed-model of TALBP with the precedence, zoning, capacity, side and synchronism constraints and the objective of minimizing the number of workstations (Simaria and Vilarinho 2009). The results showed that the proposed procedure is superior to the procedure of Lee et al. (2001) for a single model for TALBP. Kim et al. (2000) addressed the TALBP using the genetic algorithm (GA). The objective was to minimize the number of stations within a given cycle time with positional constraints. Özcan and Toklu (2009a) proposed a tabu search algorithm for the TALBP with the objective of minimizing the number of stations and the smoothness index. Özcan and Toklu presented a simulated annealing algorithm for the mixed-model of TALBP with the main objective of minimizing the number of mated-stations and the secondary objective of minimizing the number of operators, considering two performance criteria simultaneously: maximizing the weighted line efficiency and minimizing the weighted smoothness index (Özcan and Toklu 2009b). Özbakyr and Tapkan presented a bees algorithm to solve TALBP with and without zoning constraints in order to minimize the number of stations for a given cycle time. The results showed that the proposed bees algorithm outperformed the compared reference algorithms for most of the tested problems (Özbakir and Tapkan 2011). Taha et al. developed a genetic algorithm (GA) to solve TALBP, where a station oriented procedure was proposed to assign tasks to mated-stations and these rules were shown to be effective especially for large problem instances. The results showed that the proposed GA found the best solution for more than 90% of the test problems (Taha et al. 2011). Khorasanian et al. (2013) suggested a performance criterion called assembly line tasks consistency to calculate the average relationship between the tasks assigned to the stations. They proposed a simulated annealing algorithm to solve TALBP with and without considering the relationships between tasks. The

computational results showed that the proposed simulated annealing algorithm outperformed the compared reference algorithms and found five new best solutions for the number of stations performance criterion and ten new best solutions for the number of mated-stations performance criterion for the tested public benchmark instances (Khorasanian et al. 2013).

A new family of assembly line balancing problems, called Time and Space constrained Assembly Line Balancing Problem (TSALBP), considers space limitations in addition to the common constraints in Simple Assembly Line Balancing Problem (SALBP) (Bautista and Pereira 2007). Although the acronym TSALBP is very similar to TALBP, they possess very different natures.

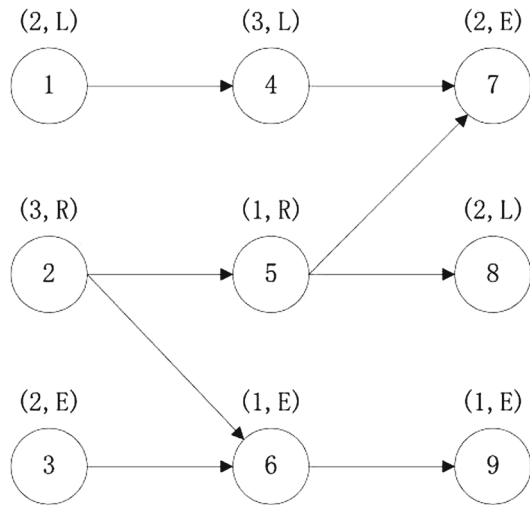
Recently, the general path relinking (PR) framework (Glover 1997; Glover et al. 2000) has attracted increasing attention in the community of combinatorial optimization, and shows outstanding performances in solving a number of hard problems, such as unconstrained binary quadratic optimization (Wang et al. 2012), multiple-level warehouse layout (Zhang and Lai 2006), capacitated clustering (Deng and Bard 2013), and multi-depot periodic vehicle routing (Rahimi-Vahed et al. 2013). In this paper, we introduce an effective path relinking algorithm for solving the TALBP which relies on both a solution relinking procedure and a local search procedure. To the best of our knowledge, this is the first path relinking algorithm for solving the TALBP problem. Assessed on a set of 45 benchmark instances commonly used in the literature, the proposed algorithm proves highly effective compared to the state-of-the-art algorithms in the literature, by improving the best solutions for one instance, and obtaining comparable or even competitive results compared with four state-of-the-art reference algorithms in the literature.

The rest of the paper is organized as follows. In Sect. 2, we describe in detail the TALBP problem. In Sect. 3, our proposed MN-PR algorithm is given and each main component of the algorithm is described in details. In Sect. 4, we show our computational results and comparisons with the current best performing algorithms in the literature. In Sect. 5, we investigate the significance of some key ingredients of our MN-PR algorithm, before concluding the paper in Sect. 6.

2 Problem description

Different from the single assembly line, a two-sided assembly line has a mated station for each cycle time. Thus, there are two operators working at the opposite sides of the line simultaneously performing different tasks of the same individual product. The tasks are performed on the mated-stations according to certain sequence dependence of tasks and may have restrictions on the operation directions. Some assembly operations can only be performed on a specified side, while others could be performed at either side of the line. Therefore, the tasks are classified into three types: Left (L), Right (R), and Either (E) tasks. A task can only be assigned to a station if the following two constraints are both satisfied: (1) the sum of the task time and the total task times of the tasks performed before that task in the same station is less than or equal the cycle time; (2) the sum of the task time and the finishing time of its predecessor in the opposite-side of that mated-station is less than or equal the cycle time.

Fig. 1 An example of the TALBP problem



The task precedence relationships can be defined by a precedence diagram, where the operation time t and the operation direction (R , L or E) are shown for each task. For example, Fig. 1 shows the precedence diagram of the tasks needed to assemble a product on a two-sided assembly line. In this diagram, each node represents a task and each directed arrow between nodes i and j indicates that task i immediately precedes task j . There is a label (t_i, d_i) above each node i . The first element of this label indicates the time required for performing task i , and the second element designates the side on which task i can be performed. For the second element, there are three types named R , L , and E . The type $R(L)$ for a task implies that the task can only be performed on the right (left) side of the line. If a task has the type E , it can be performed on either side of the line. The TALBP studied in this paper follows the general assumptions of the TALBP with deterministic operation times and without assignment restrictions except the precedence constraints.

The objective of the TALBP is thus to optimize the numbers of the mated-station and stations, which are respectively denoted as NM and NS . Note that NM is the main objective while NS is a side objective when the main objective NM ties.

3 Multi-neighborhood based path relinking for TALBP

In this section, we present our multi-neighborhood based path relinking algorithm (MN-PR) for solving the TALBP. Our proposed MN-PR algorithm integrates several distinguishing features which ensure its effectiveness, including two complementary neighborhoods, a path relinking operator to generate new solutions and an infeasible solution fixing strategy.

3.1 Search space and evaluation function

The solution of the TALBP can be represented as a sequence of all the tasks. By running a procedure, called *feasible solution building procedure* as described in Sect. 3.3, we can obtain a unique solution from a given sequence. Thus, for the TALBP, the search space Ω explored by our MN-PR algorithm is composed of all feasible task sequences. The size of the search space Ω is bounded by $O(n!)$, where n is the number of tasks in the instance. We say a sequence is feasible if for any task, all its preceding tasks are before it in the sequence.

To evaluate the quality of a candidate solution $s \in \Omega$, we adopt an evaluation function which is defined by considering the two objectives simultaneously: (i) minimizing the number of mated-stations, (ii) minimizing the number of stations. Formally, it is stated as:

$$f = NM + \frac{NS}{2 * NM + 1} \quad (1)$$

where NS and NM are the numbers of stations and mated stations in the solution, respectively.

3.2 Main framework

Path relinking algorithms are known to be highly effective for solving a large number of constraint satisfaction and optimization problems. By combining the more global relinking procedure and the more intensive local search, the path relinking framework offers a useful balance between intensification and diversification as a means of exploiting the search space.

In principle, our multi-neighborhood based path relinking algorithm (MN-PR) repeatedly alternates between a path relinking operator that is used to generate new offspring solutions and a local search procedure that optimizes the newly generated offspring solutions. As soon as an offspring solution is improved by local search, the population is accordingly updated based on two criteria: solution quality and population diversity.

The general architecture of the MN-PR algorithm is described in Algorithm 1. Notice that we use a simplified description of the PR framework here compared with the traditional path relinking, which can be considered as a hybrid evolutionary algorithm where the recombination operator is replaced by a path relinking operator. The proposed MN-PR algorithm is composed of four main components: population initialization, a local search procedure, a path relinking operator and a population updating rule.

Starting from an initial random population, MN-PR uses the local search procedure to optimize each individual to reach a local optimum (lines 4–6). Then, the path relinking operator is employed to generate new offspring solutions (line 8), whereupon a new round of local search is again launched to optimize the objective function. Subsequently, the population updating rule decides whether such an improved solution

should be inserted into the population and which existing individual should be replaced (line 10). In the following subsections, the main components of our MN-PR algorithm are described in details.

Several stopping criteria are possible for the above MN-PR algorithm, such as a maximum number of iterations, the number of times of updating the population, and so on. In this work, our MN-PR algorithm terminates when the maximum number of population updatings is reached.

Algorithm 1 Pseudo-code of our MN-PR algorithm for TALBP

```

1: Input: problem instance
2: Output: the best solution found so far
3:  $P = \{s^1, \dots, s^p\} \leftarrow \text{Population\_Initialization}()$ 
4: for each  $s^i \in P$  do
5:    $s^i \leftarrow \text{Local\_Search}(s^i)$ 
6: end for
7: while stop condition is not met do
8:   for two random individuals  $s^a$  and  $s^b \in P$  do
9:      $s^0 \leftarrow \text{Path\_Relinking}(s^a, s^b)$ 
10:     $s^0 \leftarrow \text{Local\_Search}(s^0)$ 
11:     $P \leftarrow \text{Pool\_Updating}(P, s^0)$ 
12:   end for
13: end while

```

3.3 Building a feasible solution from a sequence

Since our solution is represented as a sequence of all the tasks, we need to build a feasible solution from a given sequence. Specifically, we use a side assignment rule proposed by [Taha et al. \(2011\)](#) to generate a feasible solution from a sequence. The main idea is that tasks are assigned to stations according to a station-oriented heuristic. For a given sequence, a new mated station is firstly opened. Then, tasks are selected according to their positions in the sequence and their preferred operation direction to fill this mated station as much as possible while considering the cycle time. This procedure is repeated until all the tasks are assigned. Interested readers are referred to [Taha et al. \(2011\)](#) for more details.

3.4 Population initialization

In our MN-PR algorithm, the individuals of the initial population are randomly generated. To generate our initial population, we apply the following procedure as described in [Algorithm 2](#). In this manner, each of the generated individuals in the population is a feasible solution.

Algorithm 2 Initial population generation procedure

```

1: Input: problem instance
2: Output: initial population
3: for ( $i=1$  to  $p$ ) do
4:   Initialize a sequence  $s \leftarrow \emptyset$ 
5:    $TWP \leftarrow$  tasks without predecessors
6:   while  $TWP \neq \emptyset$  do
7:     Randomly choose a  $task_j \in TWP$  and assign it at the end of sequence  $s$ 
8:     Delete  $task_j$  from  $TWP$ 
9:      $TWP \leftarrow$  tasks whose predecessors have all been assigned
10:  end while
11:  if ( $s$  is different from all the other sequences in the population) then
12:    Call feasible solution building procedure to get a solution for sequence  $s$ 
13:    Add this solution  $s$  into the population
14:  end if
15: end for

```

3.5 Local search procedure

A simple descent local search is employed in our proposed MN-PR algorithm. At each iteration, the best neighboring solution is selected to be compared with the current solution. If the objective value of the best neighboring solution is better than that of the current solution, this neighboring solution is used to replace the current solution. This procedure is repeated to iteratively improve the current solution until no improving solutions can be found in the current neighborhood.

In our MN-PR algorithm, our local search method employs two complementary neighborhoods. Given a sequence as the starting point of our local search, the first neighborhood is first explored. When no improving solution can be found using the first neighborhood, we switch to the second neighborhood and optimize the objective function further until it also reaches the local optimum solution too.

In local search, a neighborhood is typically defined by a move operator mv , which transforms a solution X to generate a neighboring solution X' , denoted by $X' = X \oplus mv$. Let $M(X)$ be the set of all possible moves which can be applied to X , then the neighborhood N of X is defined by $N(X) = \{X' : X' = X \oplus mv, mv \in M(X)\}$.

To move from one solution to another in the search space, our local search procedure employs two different neighborhood structures:

The first neighborhood is defined by moving a task from its original position in the sequence to another position. Note that the new position of the moved task should be feasible such that its new position is between all its predecessors and all its successors. This neighborhood move is called *insert* move, denoted as N_1 . Figure 2 gives an example of this neighborhood, where task 5 can be inserted before task 3 and task 4, as well as being inserted after task 6.

Our second neighborhood is called *swap* move neighborhood, denoted as N_2 . It consists of swapping the positions of two tasks in the given sequence. In order to get a feasible solution after this neighborhood move, we impose a restriction that the new positions of the swapped tasks are both between all its predecessors and all its successors. In addition, for the purpose of reducing the size of the search space, we

Fig. 2 Neighborhood structure N_1

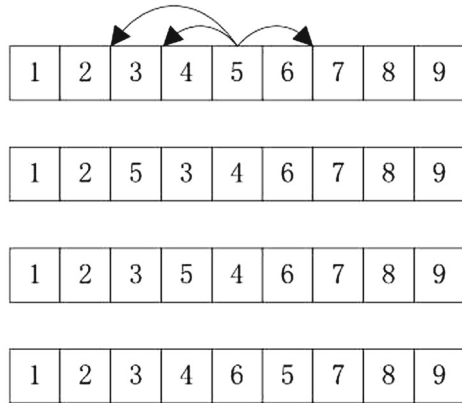
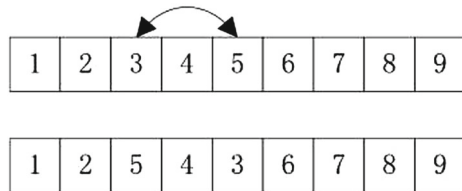


Fig. 3 Neighborhood structure N_2



restrict the distance between the positions of the two swapped tasks to be no more than 3, which can also be verified by the fact in our experiments that long distance swap moves usually deteriorate the solution quality especially for high quality solutions. Figure 3 illustrates an example of this *swap* neighborhood.

According to our experiments, these two neighborhoods are complementary to each other and can enhance the search effectiveness by combining them together, although the *insert* neighborhood has been widely used in the literature and can be considered as the basic neighborhood in our local search procedure.

3.6 Path relinking operator

The path relinking operator aims to generate new solutions by creating paths connecting two high-quality (parent) solutions, and it is composed of two main operations. The first one is to construct a path connecting two parent solutions, where the parent solutions located at the beginning and the end of the path are respectively called the initiating solution and the guiding solution, while other solutions are called intermediate ones. Another operation is to choose one solution as the reference solution from the constructed path for further improvement. In order to describe our path relinking procedure, we first give some primary definitions, denoting the initiating solution by s^a and the guiding solution by s^b :

- *LCS*: The longest common subsequence of s^a and s^b
- *NC*: The set of tasks not included in the *LCS* of solutions s^a and s^b
- *HD*: The Hamming moving distance from s^a to s^b

In this work, we present two path relinking operators respectively called the greedy path relinking (GPR) operator and the random path relinking (RPR) operator. For the GPR operator, the procedure of creating a path connecting two parent solutions is described in Algorithm 3, where a solution sequence (i.e., a path) of length $HD + 1$ ($s(0), s(1), s(2), \dots, s(HD)$) is generated in a step by step way by starting from $s(0)$. Note that $s(m)$ differs from $s(m - 1)$ by the relative position of only one task, $m = 1, 2, \dots, HD$. In addition, $s(0)$ and $s(HD)$ correspond respectively to the initiating solution s^a and the guiding solution s^b , while other solutions are intermediate (or path) solutions. Moreover, at each step, a best *insert* move is always chosen to generate the next intermediate solution. In Algorithm 3, $s(r + 1) \leftarrow s(r) \oplus l^*$ (line 19) means that the task in the l^* th position of $s(r)$ is moved to the position as the guiding solution s^b and this generated solution is denoted as $s(r + 1)$.

Algorithm 3 Pseudo-code of constructing a path from s^a to s^b with a greedy path relinking operator

```

1: Input: A pair of solutions ( $s^a, s^b$ )
2: Output: Path solutions  $s(0), s(1), \dots, s(r)$  from  $s^a$  to  $s^b$ 
3: Find the  $LCS(s^a, s^b)$  of  $s^a$  and  $s^b$ 
4:  $s(0) \leftarrow s^a, s \leftarrow s^a, r \leftarrow 0, NC \leftarrow \emptyset$ 
5: for each  $task_i \in s^a$  do
6:   if ( $task_i \notin LCS$ ) then
7:      $NC \leftarrow task_i$ 
8:   end if
9: end for
10: while  $NC \neq \emptyset$  do
11:    $min = \infty$ 
12:   for each  $task_i \in NC$  do
13:     Find  $task_i$ 's position  $l$  in  $s(r)$  and tentatively move it to the position as in  $s^b$ 
14:     if  $f(s(r) \oplus l) < min$  then
15:        $min = f(s(r) \oplus l)$ 
16:        $j = task_i; l^* = l$ 
17:     end if
18:   end for
19:   Move task  $j$  to its new position as in  $s^b$ :  $s(r + 1) \leftarrow s(r) \oplus l^*$ 
20:   Remove task  $j$  from  $NC$ :  $NC \leftarrow NC \setminus \{j\}$ 
21:    $r \leftarrow r + 1$ 
22: end while

```

After the creation of the path, we choose one solution from this path such that the chosen solution is far enough from the initiating and guiding solutions and has a good objective value (Wang et al. 2012). Specifically, we construct a candidate solution list (CSL) that consists of the intermediate solutions having a distance of at least $\xi \cdot HD$ (where ξ is a predetermined parameter value between 0 and 1.0 and is empirically set to be 0.4 in our experiments) from both the initiating and guiding solutions. Then, the solution having the best objective value in CSL is chosen as the reference solution.

For the RPR operator described in Algorithm 4, the intermediate solutions are generated from s^a to s^b in a random way. The only difference with the GPR operator

is that at each step it chooses a random move towards the guiding solution s^b regardless of the objective function. This process is repeated until NC becomes empty.

The reference solution selection strategy is the same as the GPR operator. However, one should notice that the intermediate solutions on the generated path from s^a to s^b may not be feasible solutions for the RPR operator. In this case, we just use the infeasible solution fixing strategy as described in Sect. 3.7 to fix this solution into a feasible one.

Finally, with a consideration of efficiency (see Sect. 5.2 for a comparison between the GPR and RPR operators), our MN-PR algorithm employs the RPR operator as its path relinking operator, and the GPR operator is designed just for the purpose of comparison and analysis.

Algorithm 4 Pseudo-code of constructing a path from s^a to s^b with a random path relinking operator

```

1: Input: A pair of solutions  $(s^a, s^b)$ 
2: Output: Path solutions  $s(0), s(1), \dots, s(r)$  from  $s^a$  to  $s^b$ 
3: Find the LCS of  $s^a$  and  $s^b$ 
4:  $s(0) \leftarrow s^a, s \leftarrow s^a, r \leftarrow 0, NC \leftarrow \emptyset$ 
5: for each  $task_i \in s^a$  do
6:   if ( $task_i \notin LCS$ ) then
7:      $NC \leftarrow task_i$ 
8:   end if
9: end for
10: while  $NC \neq \emptyset$  do
11:   Randomly pick a  $task_i$  from  $NC$ 
12:   Find  $task_i$ 's position  $l$  in  $s(r)$ 
13:   Move  $task_i$  to its new position as in  $s^b$ :  $s(r+1) \leftarrow s(r) \oplus l$ 
14:   Remove  $task_i$  from  $NC$ :  $NC \leftarrow NC \setminus \{task_i\}$ 
15:    $r \leftarrow r + 1$ 
16: end while

```

3.7 Infeasible solution fixing strategy

One should notice that not all the path solutions $s(1), \dots, s(r)$ constructed from s^a to s^b are feasible ones, since the precedence constraint of the TALBP is very strong and the generated intermediate solutions may not be feasible. For this reason, we use an infeasible fixing strategy fix those infeasible solutions into feasible ones.

If a solution is infeasible, it must violate some of the task precedence constraints. For example, if task A is the predecessor of task B, but task B is in front of task A in the sequence, an infeasible solution is generated. In this case, we need to remove this conflict by moving task A to a position after task B or moving task B to a position before task A. However, if there are several conflicts in a sequence, it is possible that a new conflict occurs while solving a previous conflict. In order to avoid this situation and improve the fixing efficiency, two rules are considered as follows:

1. Each time the task involved in the largest conflict (defined as the largest distance of the two conflicting tasks) is selected to move;

2. If a task is once moved to remove a conflict, this task cannot be moved any more.

With these two rules, our infeasible solution fixing strategy can quickly fix an infeasible solution into a feasible one, based on which our local search can then further optimize it to a local optimum.

3.8 Population updating

In our MN-PR algorithm, we use a simple population updating criterion to decide if a newly generated offspring solution should be inserted into the population and which solution should be replaced if yes. Specifically, when an offspring s^0 is obtained by the path relinking procedure, we optimize s^0 by the local search procedure. If the improved s^0 is better than the worst solution in the population according to the objective value and is not the same as any solution in the population, the worst solution in the population is replaced by the improved solution s^0 . This population updating strategy can help to maintain an elite population with certain population diversities.

4 Computational results

In this section, experiments are carried out to evaluate the performance of our proposed MN-PR algorithm.

4.1 Test instances

To ensure a fair comparison, we test our proposed algorithm on the same set of TALBP instances as those used by the reference algorithms in the literature. The benchmark problems include: P12 and P24 are taken from [Kim et al. \(2000\)](#), P16, P65, and P205 are taken from [Lee et al. \(2001\)](#), and P148 is taken from [Bartholdi \(1993\)](#) and modified by [Lee et al. \(2001\)](#).

4.2 Experimental protocol and reference algorithms

Our algorithm is programmed by using the C# programming language of Visual Studio 2010 and the experiments are conducted on a PC with an Intel(R) Core(TM) i3-2310M CPU 2.10GHz and 4GB of RAM. For each instance, our MN-PR algorithm is independently run for 5 times and each run is limited to be 100 times of population updating. The following results report the best results of these 5 runs.

Our results are compared with four state-of-the-art algorithms for solving the TALBP in the literature, which include: a tabu search algorithm (TSA) proposed by [Özcan and Toklu \(2009a\)](#), a bee colony algorithm (BA) proposed by [Özbakir and Tapkan \(2011\)](#), a genetic algorithm (GA) proposed by [Taha et al. \(2011\)](#) and a simulated annealing algorithm (mSA) proposed by [Khorasanian et al. \(2013\)](#).

4.3 Results and comparisons with reference algorithms

The computational results of our MN-PR algorithm and comparisons with four high-performance reference algorithms are presented in Table 1. The first two columns give the problem name and the cycle time (CT) for each instance. In the third column, the lower bounds of both the number of stations (NS) and mated stations (NM) are presented. The next four columns illustrate the computational results of the four reference algorithms where the TSA and BA algorithms only report their NS values, while the results of our MN-PR algorithm are provided in the last two columns. In addition, the last three rows provide the summarized comparison between our MN-PR algorithm and the four reference algorithms, which respectively represent the number of instances for which our MN-PR algorithm can get better, equal and worse results than the corresponding reference algorithms. In the table, “x” denotes that the reference algorithm does not report the result for the corresponding instance.

From Table 1, one observes that when our MN-PR algorithm is compared with TSA, the results are very close with each other for the small-sized problems. However, our MN-PR outperforms TSA for large instances. For all the ten P205 problems, our MN-PR algorithm can obtain better results than TSA. In total, our MN-PR algorithm can get 13 better results than TSA, while no worse results are obtained. As for the BA algorithm, our MN-PR algorithm also obtains better results. Specifically, our MN-PR algorithm can get 6 better results while one worse result is obtained. When compared with GA, our MN-PR algorithm can get better or equal results for all the 45 tested instances. Particularly, our MN-PR algorithm can get better NS and NM values for 5 and 4 instances, respectively. Finally, compared with mSA, the best performing algorithm in the literature for TALBP, although our MN-PR algorithm obtains worse NS and NM values for 4 and 3 instances, our MN-PR algorithm can get better NS value for one instance (P65_326) and equal results for all the remaining instances.

It is worthy to notice that our MN-PR algorithm improves the upper bound for the NS value of instance P65_326. Two different best solutions for instance P65_326 are shown in Figs. 4 and 5. Interested readers are referred to Lee et al. (2001) for detailed information about instance P65_326.

5 Analysis and discussion

Now we turn our attention to analyzing some important features of the MN-PR algorithm, including the importance of the proposed path relinking operator and the multi-neighborhood strategy.

5.1 Importance of multi-neighborhood strategy

In this section, we try to figure out which neighborhood is the most essential one in our MN-PR algorithm. The experiments are performed by comparing the original MN-PR algorithm with simplified versions of our MN-PR algorithm. Specifically, we consider two variants of the MN-PR algorithm respectively with neighborhoods N_1 and N_2 .

Table 1 Computational results of our MN-PR algorithm and comparisons with the reference algorithms for the public TALBP instances

Prob.	CT	LB	TSA (Özcan and Toklu 2009a)		BA (Özbakir and Tapkan 2011)		GA (Taha et al. 2011)		mSA (Khorasanian et al. 2013)		MN-PR	
			NS	NS	NS	NM	NS	NM	NS	NM	NS	NM
P9	3	3[6]	6	6	6	3	6	3	x	x	6	3
	4	3[5]	5	5	5	3	5	3	x	x	5	3
	5	2[4]	4	4	4	2	4	2	x	x	4	2
	6	2[3]	3	3	3	2	3	2	x	x	3	2
	4	4[7]	x	7	7	4	7	4	7	4	7	4
	5	3[5]	6	6	6	3	6	3	6	3	6	3
P12	6	3[5]	5	5	5	3	5	3	5	3	5	3
	7	2[4]	4	4	4	2	4	2	4	2	4	2
	8	2[4]	4	4	4	2	4	2	x	x	4	2
	15	3[6]	x	6	6	4	6	4	6	4	6	4
	16	3[6]	6	6	6	4	6	4	x	x	6	4
	18	3[5]	x	6	6	3	6	3	6	3	6	3
P16	19	3[5]	5	5	5	3	5	3	x	x	5	3
	20	3[5]	x	5	5	3	5	3	5	3	5	3
	21	2[4]	5	5	5	3	5	3	x	x	5	3
	22	2[4]	4	4	4	2	4	2	4	2	4	2
	18	4[8]	8	8	8	4	8	4	x	x	8	4
	20	4[7]	8	8	8	4	8	4	x	x	8	4
P24	24	3[6]	6	6	6	3	6	3	x	x	6	3
	25	3[6]	6	6	6	3	6	3	6	3	6	3

Table 1 continued

Prob.	CT	LB	TSA (Özcan and Toklu 2009a)		BA (Özbakir and Tapkan 2011)		GA (Taha et al. 2011)		mSA (Khorasanian et al. 2013)		MN-PR	
			NS	NS	NS	NS	NS	NM	NS	NM	NS	NM
P65	30	3[5]	5	5	5	3	5	3	5	5	3	3
	35	2[4]	4	4	4	2	4	2	4	4	2	2
	40	2[4]	4	4	4	2	4	2	4	4	2	2
	326	8[16]	17	17	17	9	17	9	17	17	9	16
	381	7[14]	15	14	14	8	14	8	14	14	7	14
P148	435	6[12]	13	12	12	7	13	7	12	12	6	12
	490	6[11]	11	11	11	6	11	6	11	11	6	11
	544	5[10]	10	10	10	5	10	5	10	10	5	10
	204	13[26]	26	26	26	14	27	14	26	26	13	26
	255	11[21]	21	21	21	11	21	11	21	21	11	21
	306	9[17]	18	17	17	9	18	9	17	17	9	18
	357	8[15]	15	15	15	8	15	8	15	15	8	15
	408	7[13]	13	13	13	7	13	7	13	13	7	13
	459	6[12]	12	12	12	6	12	6	12	12	6	12
	510	6[11]	11	11	11	6	11	6	11	11	6	11
P205	1133	11[21]	24	22	22	13	22	13	22	22	11	22
	1322	9[18]	21	20	20	10	20	10	18	18	9	19
	1510	8[16]	18	17	17	9	17	9	16	16	8	16
	1699	7[14]	17	16	16	8	15	8	14	14	7	15
	1888	7[13]	16	14	14	7	14	7	13	13	7	14
	2077	6[12]	14	12	12	6	12	6	12	12	6	12

Table 1 continued

Prob.	CT	LB	TSA (Özcan and Toklu 2009a)		BA (Özbakir and Tapkan 2011)		GA (Taha et al. 2011)		mSA (Khorasani et al. 2013)		MN-PR	
			NS	NM[NS]	NS	NS	NS	NM	NS	NM	NS	NM
	2266	6[11]	13		11		11	6	11	6	11	6
	2454	5[10]	12		11		10	5	10	5	10	5
	2643	5[9]	11		10		10	5	10	5	10	5
	2832	5[9]	10		10		9	5	9	5	9	5
#Better			13		6		5	4	1	0		
#Equal			28		38		40	41	29	31		
#Worse			0		1		0	0	4	3		

Bold values indicate the best result among all the reference algorithms

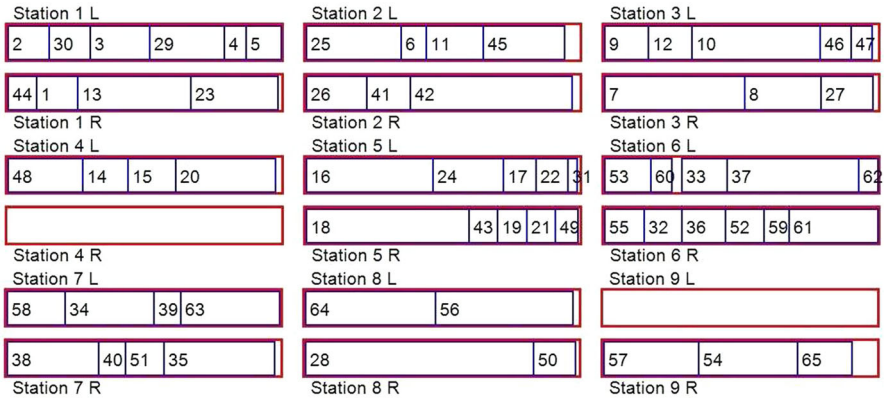


Fig. 4 A solution of $NS = 16$, $NM = 9$ for instance P65_326

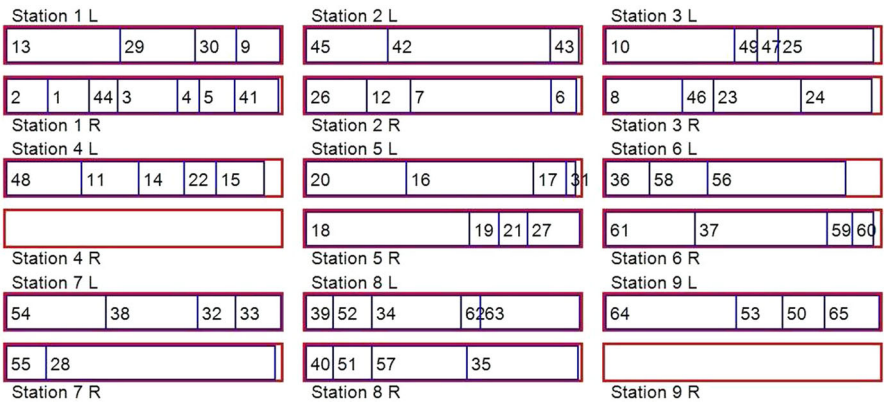


Fig. 5 Another solution of $NS = 16$, $NM = 9$ for instance P65_326

Table 2 shows the results of the above mentioned variants of the MN-PR algorithm for 22 large size instances, since small size problem instances are very easy to solve for all the variants of our algorithms. For each variant of our MN-PR algorithm, the best NS and NM values as well as the computational time to reach this best solution are reported.

From Table 2 one finds that the original MN-PR algorithm outperforms other variants. The MN-PR algorithm with neighborhood N_2 performs worse than the other two variants of our MN-PR algorithm, showing that neighborhood N_1 is the basic neighborhood in our algorithm. One also observes that with both neighborhoods, our MN-PR algorithm can obtain better results the other two variants with just one neighborhood respectively for 5 and 11 instances, indicating the importance of combining the two neighborhoods.

From the experiments we can observe that each neighborhood is important in our MN-PR algorithm. The combination of these different neighborhoods make the algo-

Table 2 Comparison among the MN-PR algorithms with neighborhood N_1 , N_2 and $N_1 + N_2$ in terms of the best solution found on the 22 large instances

Prob.	CT	LB	MN-PR with N_1		MN-PR with N_2		MN-PR with $N_1 + N_2$	
		NM[NS]	NM[NS]	t(s)	NM[NS]	t(s)	NM[NS]	t(s)
P65	326	8[16]	9[17]	24.3	9[17]	14.3	9[16]	49.5
	381	7[14]	8[14]	32.6	8[15]	12.4	7[14]	55.9
	435	6[12]	7[13]	21.5	7[13]	12.5	7[12]	46.3
	490	6[11]	6[11]	24.8	6[11]	15.2	6[11]	40.3
	544	5[10]	5[10]	22.7	5[10]	13.1	5[10]	35.9
P148	204	13[26]	14[27]	321.9	14[26]	130.5	13[26]	459.9
	255	11[21]	11[21]	359.7	11[22]	91.3	11[21]	420.1
	306	9[17]	9[18]	275.8	9[18]	81.1	9[18]	401.7
	357	8[15]	8[15]	335.8	8[15]	91.8	8[15]	451.5
	408	7[13]	7[13]	299.9	7[13]	84.9	7[13]	451.1
	459	6[12]	6[12]	256.8	6[12]	87.1	6[12]	367.5
	510	6[11]	6[11]	295.3	6[11]	82.6	6[11]	453
	205	1133	11[21]	11[22]	264.7	12[23]	130	11[22]
1322	9[18]	10[19]	302.5	10[19]	153.8	10[19]	352.6	
1510	8[16]	9[17]	311.3	9[17]	142.6	8[16]	460.1	
1699	7[14]	8[15]	276	8[15]	124.3	8[15]	433.5	
1888	7[13]	7[14]	199.1	7[14]	109.2	7[14]	320.8	
2077	6[12]	6[12]	280.7	7[13]	120	6[12]	381.7	
2266	6[11]	6[11]	211.9	6[12]	104.4	6[11]	441.3	
2454	5[10]	5[10]	270.4	6[11]	122	5[10]	454.7	
2643	5[9]	5[10]	218.3	5[10]	109.9	5[10]	337.3	
2832	5[9]	5[9]	288.6	5[10]	102.2	5[9]	373.9	

Bold values indicate the best result among all the reference algorithms

rithm more powerful and each neighborhood is meaningful in the proposed MN-PR algorithm.

5.2 Comparison between the Path Relinking Operators

The relinking operator is one of the fundamental ingredients of our MN-PR algorithm. In order to analyze the influence of the relinking operator on the performance of the MN-PR algorithm, we compare our GPR and RPR relinking operators, and the corresponding MN-PR algorithms are respectively denoted by GPR and RPR. The experiment is carried out on the same set of benchmarks as mentioned in Sect. 5.1. The computational results are summarized in Table 3. The first two columns give the instance name and the cycle time. Column 3 gives the lower bound. Computational results of GPR and RPR are respectively listed in the following columns, including the best NS and NM values, as well as the computing time to reach the best solution ($t(s)$ in seconds).

Table 3 Comparison between the MN-PR algorithms with GPR and RPR operators

Prob.	CT	LB	MN-PR with GPR		MN-PR with RPR	
		NM[NS]	NM[NS]	t(s)	NM[NS]	t(s)
P65	326	8[16]	9[17]	38.5	9[16]	49.5
	381	7[14]	8[14]	34.9	7[14]	55.9
	435	6[12]	7[13]	35.9	7[12]	46.3
	490	6[11]	6[11]	38.4	6[11]	40.3
	544	5[10]	5[10]	33.4	5[10]	35.9
P148	204	13[26]	14[27]	314.3	13[26]	459.9
	255	11[21]	11[21]	346.2	11[21]	420.1
	306	9[17]	9[18]	308.4	9[18]	401.7
	357	8[15]	8[15]	335.3	8[15]	451.5
	408	7[13]	7[13]	332.6	7[13]	451.1
	459	6[12]	6[12]	334.9	6[12]	367.5
	510	6[11]	6[11]	321.9	6[11]	453
P205	1133	11[21]	12[23]	334.1	11[22]	416.5
	1322	9[18]	10[20]	296.2	10[19]	352.6
	1510	8[16]	9[17]	309.6	8[16]	460.1
	1699	7[14]	8[15]	301.9	8[15]	433.5
	1888	7[13]	7[14]	278.1	7[14]	320.8
	2077	6[12]	6[12]	327	6[12]	381.7
	2266	6[11]	6[12]	298.4	6[11]	441.3
	2454	5[10]	5[10]	351.8	5[10]	454.7
	2643	5[9]	5[10]	308.7	5[10]	337.3
	2832	5[9]	5[10]	298.6	5[9]	373.9

Bold values indicate the best result among all the reference algorithms

In comparison with GPR, RPR obtains better results for nine instances in terms of *NS* or *NM* values and no worse results are obtained, which implies that the RPR operator is more robust than the GPR operator. On the other hand, the computing time of RPR is comparable to that of GPR. Therefore, it can be concluded that the search capability of the random relinking operator is better than the greedy relinking operator, which can be explained by the fact that the random relinking operator may bring more diversification into the search and it is more appropriate to be integrated with the intensive local search procedure.

6 Conclusion

In this paper, we propose a multi-neighborhood based path relinking algorithm for solving the two-sided assembly line balancing problem. The proposed MN-PR algorithm includes a number of distinguishing features, such as a multi-neighborhood based local search strategy, a dedicated path relinking procedure to generate new solutions

and a strategy to fix an infeasible solution created by the path relinking procedure to a feasible one.

We test the proposed MN-PR algorithm on a set of 45 benchmark instances commonly used in the literature. Computational results show that our algorithm is highly effective in comparison with the state-of-the-art algorithms in the literature. For most instances, our proposed MN-PR algorithm is able to find the optimal or near optimal solutions in a reasonable time. Specifically, it improves the best known *NS* value for instance P65_326.

We studied some essential ingredients of the proposed algorithm which shed light on the following points. First, the multi-neighborhood strategy plays a crucial role in the high performance of the MN-PR algorithm. Second, the random path relinking (RPR) operator is generally better than the greedy path relinking (GPR) operator for TALBP.

There are several directions to extend this work. One immediate possibility is to introduce new diversification strategies in the path relinking procedure, such as exterior path relinking, to increase the diversity of the algorithm. The other possibility is to develop more advanced local search procedures, such as tabu search (Lü and Huang 2009; Huang et al. 2013). Finally, it is important to consider problem specific knowledge to enhance the performance of the present MN-PR algorithm.

Acknowledgments This paper is dedicated to memorizing the first author's supervisor, Prof. Wenqi Huang, for his lifelong insightful instruction. Special thanks are given to Prof. Zhipeng Lü for his helpful comments and suggestions to improve this paper significantly. The authors greatly acknowledge the financial supports from the National Natural Science Foundation of China (NSFC) with the Grant Number 51275191, and the Fundamental Research Funds for the Central Universities of HUST with the Grant Number 2014TS033.

References

- Bartholdi J (1993) Balancing two-sided assembly lines: a case study. *Int J Prod Res* 31(10):2447–2461
- Bautista J, Pereira J (2007) Ant algorithms for a time and space constrained assembly line balancing problem. *Eur J Oper Res* 177(3):2016–2032
- Deng Y, Bard J (2013) A reactive GRASP with path relinking for capacitated clustering. *J Heuristics* 17(2):119–152 2011
- Glover F (1997) A template for scatter search and path relinking. *Lect Notes Comput Sci* 1363:1–51
- Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. *Control Cybern* 39:653–684
- Huang W, Yin A (2004) An improved shifting bottleneck procedure for the job shop scheduling problem. *Comput Oper Res* 31(12):2093–2110
- Huang W, Wang L (2006) A local search method for permutation flow shop scheduling. *J Oper Res Soc* 57(10):1248–1251
- Huang W, Fu Z, Xu R (2013) Tabu search algorithm combined with global perturbation for packing arbitrary sized circles into a circular container. *Sci China Inform Sci* 56(9):1–14
- Hu X, Wu E, Jin Y (2008) A station-oriented enumerative algorithm for two-sided assembly line balancing. *Eur J Oper Res* 186(1):435–440
- Khorasanian D, Hejazi S, Moslehi G (2013) Two-sided assembly line balancing considering the relationships between tasks. *Comput Ind Eng* 66(4):1096–1105
- Kim Y, Kim Y, Kim Y (2000) Two-sided assembly line balancing: a genetic algorithm approach. *Prod Plan Control* 11(1):44–53
- Lee T, Kim Y, Kim Y (2001) Two-sided assembly line balancing to maximize work relatedness and slackness. *Comput Ind Eng* 40(3):273–292

- Lü Z, Huang W (2009) Iterated tabu search for identifying community structure in complex networks. *Phys Rev E* 80(2):026130
- Özbakir L, Tapkan P (2011) Bee colony intelligence in zone constrained two-sided assembly line balancing problem. *Expert Syst Appl* 38:11947–11957
- Özcan U, Toklu B (2009a) A tabu search algorithm for two-sided assembly line balancing. *Int J Adv Manuf Technol* 43:822–829
- Özcan U, Toklu B (2009b) Balancing of mixed-model two sided assembly lines. *Comput Ind Eng* 57:217–227
- Özcan U, Toklu B (2009c) Multiple-criteria decision-making in two-sided assembly line balancing: a goal programming and a fuzzy goal programming models. *Comput Oper Res* 36:1955–1965
- Rahimi-Vahed A, Crainic T, Gendreau M, Rei W (2013) A path relinking algorithm for a multi-depot periodic vehicle routing problem. *J Heuristics* 19(3):497–524
- Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *Eur J Oper Res* 168(3):666–693
- Simaria A, Vilarinho P (2009) 2-ANTBAL: an ant colony optimization algorithm for balancing two-sided assembly lines. *Comput Ind Eng* 56(2):489–506
- Taha R, El-Kharbotly A, Sadek Y, Afia N (2011) A Genetic Algorithm for solving two-sided assembly line balancing problems. *Ain Shams Eng J* 2:227–240
- Wang Y, Lü Z, Glover F, Hao J (2012) Path relinking for unconstrained binary quadratic programming. *Eur J Oper Res* 223(3):595–604
- Zhang G, Lai K (2006) Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem. *Eur J Oper Res* 169(2):413–425