Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 168 (2006) 826–837
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Balancing assembly lines with tabu search

Sophie D. Lapierre [a,b,†], Angel Ruiz [a,c], Patrick Soriano [a,d,*]

[a] Centre de recherche sur les transports, Université de Montréal, CP 6128, Succursale Centre-ville, Montréal (Qc), Canada H3C 3J7
[b] Département de mathématiques appliquées et génie industriel, École Polytecnique de Montréal, CP 6079, Succursale Centre-ville, Montréal (Qc), Canada H3C 3A7
[c] Département opérations et systèmes de décision, Faculté des sciences de l'administration, Université Laval, Québec (Qc), Canada G1K 7P4
[d] Service dc l'enseignement des méthodes quantitatives de gestion, École des Hautes Études Commerciales, 3000 Chemin de la Côte-Sainte-Catherine, Montréal (Qc), Canada H3T 2A7

Available online 12 October 2004

Abstract

Balancing assembly lines is a crucial task for manufacturing companies in order to improve productivity and minimize production costs. Despite some progress in exact methods to solve large scale problems, softwares implementing simple heuristics are still the most commonly used tools in industry. Some metaheuristics have also been proposed and shown to improve on classical heuristics but, to our knowledge, no computational experiments have been performed on real industrial applications to clearly assess their performance as well as their flexibility. Here we present a new tabu search algorithm and discuss its differences with respect to those in the literature. We then evaluate its performance on the Type I assembly line balancing problem. Finally, we test our algorithm on a real industrial data set involving 162 tasks, 264 precedence constraints, and where the assembly is carried out on a sequential line with workstations located on both sides of the conveyor, with two possible conveyor heights and no re-positioning of the product. We discuss the flexibility of the metaheuristic and its ability to solve real industrial cases.

Keywords: Assembly line balancing; Tabu search; Metaheuristic

1. Introduction

Balancing assembly lines is a difficult combinatorial optimization problem arising frequently in manufacturing. There are two versions of the problem. Assuming a line of identical assembly workstations and a set of tasks to be processed,

† In memory of Sophie D. Lapierre, who left us September 19, 2004. Merci Sophie. Tu es partie bien trop tôt. Tu vas nous manquer!

* Corresponding author. Address: Centre de recherche sur les transports, Université de Montréal, CP 6128, Succursale Centre-ville, Montréal (Qc), Canada H3C 3J7.
E-mail address: patrick@crt.umontreal.ca (P. Soriano).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.07.031

the Type I simple assembly line balancing problem (SALBP-I), as described by Scholl [16], consists in finding an assignment of tasks to workstations such that the required number of workstations is minimized. The problem is constrained by a set of precedence relations between the tasks and by a given *cycle time*, which corresponds to the maximum work time available per workstation. The Type II simple assembly line balancing problem (SALBP-II) consists in allocating tasks to a given number of workstations in order to minimize the cycle time, i.e. the maximum work time of any workstation [12]. Both versions of the problem are NP-hard [10].

The real industrial settings however often differ significantly from the theoretical problem. For instance, it can be observed that the cycle time is generally treated in a more flexible manner. Indeed, since operation times are based on standard measures (average time for an average worker to perform a given task), occasionally one could intentionally overload a workstation and assign to it a fast employee. Also, practical applications often present other specifications such as task incompatibilities [13], two-sided assembly lines [3], etc. Such particularities are very difficult to include in mathematical models and generally increase significantly the complexity of the problem to be solved. This can explain why they are generally not considered in the literature.

The assembly line balancing problem has been extensively studied. Ghosh and Gagnon [7] found more than 150 papers published on this topic in their comprehensive survey. Salveson [15] was the first to present a mathematical formulation in 1955. Early work focused on the development of good heuristics for industrial problems. Arcus' extensive work on industrial balancing problems led him to the development of COM-SOAL, a computerized software using what we refer to as "priority-based" heuristics [2]. Such heuristics simply dispatch available tasks—i.e. those for which all predecessor tasks have already been assigned to a workstation—according to a predefined priority rule such as the "largest task first". Typically, different dispatching rules are tried on the same balancing problem since the best rule depends on the application problem [4]. COMSOAL then tries several random assignments to generate additional balancing solutions. Among all the generated solutions, the best one is usually of relatively good quality.

The research performed in the 70s and 80s focused almost exclusively on the development of exact methods to solve the basic assembly line problem. According to a recent computational study due to Scholl and Klein [19], SALOME (developed by Scholl and Klein [18]) is considered as the best-performing exact algorithm for this problem. However, despite such significant advances at the theoretical level, most algorithms used in practical settings resort to heuristics instead of exact methods (see [7]). This is due to the fact that the latter do not require complex computer coding or software maintenance while providing what is perceived in industry as good enough solutions. Most papers on the balancing problem in industrial settings are extensions of the classical heuristics developed by Arcus.

Several metaheuristic approaches have also been developed for this problem. Anderson and Ferris [1], and Rubinovitz and Levitin [14] present two different genetic algorithms to solve the SALBP-I. They both conclude on the good performance of their heuristic, but this is difficult to assess since they did not test them on standard problem sets. Gonçalves and Raimundo [9] present a hybrid heuristic where a genetic algorithm defines priorities to tasks which are then assigned to workstations by a priority rule. Scholl and Voß [17] present basic tabu search (TS) algorithms to solve the assembly line balancing problem of Types I and II. Chiang [6] proposes another TS metaheuristic on the SALBP-I. Although both methods correspond to rather simple versions of TS, they obtain good results on classical data sets. However their results also show that there is still room for improvement.

In this paper we propose a new TS algorithm to solve the Type I standard assembly line balancing problem—i.e. an assembly line for a single product with deterministic task length. This procedure explores two complementary neighborhoods and integrates several advanced features of TS to enhance its efficiency, robustness, and adaptability to real industrial settings. In particular, the

algorithm has been extended to tackle the frequent situation arising when the assembling of large products requires that operations be performed on both sides of the conveyor and at different (i.e. two) possible heights, for instance in the home appliance industry. Of course, re-positioning the product on the line would eliminate this need, but the technology necessary for such added flexibility is often too expensive for the potential benefits. Other common practical constraints like task grouping or task separation are not handled presently by the algorithm, but we believe it could easily be modified to account for them.

The rest of the paper is organized as follows. Section 2 presents the TS algorithm. In Section 3 we analyze the performance of our procedure on the SALBP-I. We then assess the adaptability of the algorithm to practical settings by solving an industrial problem with several complicating characteristics in Section 4. Section 5 concludes the paper.

2. A tabu search procedure for the SALBP-I

TS is a generalized local search procedure that uses information on the history of the search process to overcome the limitations of local optimality. It starts from an initial solution and iteratively moves to a neighbor solution which either improves on the previous solution or not. In the later case, the search process records some information on the move just made to prevent the method from reversing it right away and start cycling (see [8] and [20] for a detailed description of TS). Before presenting our TS algorithm, we will first review existing TS procedures found in the literature. The following notations and definitions will be used throughout the remainder of the paper:

C	cycle time;
t_j	execution time of task $j = 1,\ldots,n$;
n	number of tasks;
m	number of workstations in the assembly line;
S_k	set of tasks which are currently assigned to station $k = 1,\ldots,m$;
$t(S_k)$	total working time of station k given a set of tasks S_k.

2.1. Existing TS procedures

The TS procedures reported in the literature use two different approaches to tackle the SALBP-1 line balancing problem. Scholl and Voß [17] propose an indirect approach that consists basically in solving iteratively several SALBP-II with an increasing number of workstations until a solution that satisfies the target cycle time C is found. They proceed as follows: for a given number of workstations m', $m' < (\sum^n t_j)/C$, they find an initial solution having a cycle time C' larger than C. Then, they apply a TS algorithm in which they compute $\max(t(S_k))$—i.e. the cycle time of the current solution—at each iteration. An improvement is recorded whenever a reduction of this cycle time is achieved. The TS iterates until the target value of C is reached. If the solution cannot be improved enough to meet the target cycle time C, another initial solution is generated with a cycle time C'' slightly smaller than C' and one additional workstation ($m'' = m' + 1$). The TS is then applied again, starting from this new initial solution. This procedure is repeated until a solution with a cycle time smaller or equal to C can be found. With their approach, the authors were able to find better solutions than the best known heuristics on standard test data. However, the method is complex and extensive knowledge about the heuristics used to generate initial solutions is needed in order to implement the proposed search strategies.

Chiang [6] proposes a more direct approach. He starts with an initial solution found with the right cycle time C, and then applies a TS procedure that tries to reduce the number of workstations used. Implementation of this strategy requires the use of a more elaborate objective function since the natural objective function of the SALBP-I does not provide much information. Indeed, the number of workstations used in a given solution is not sufficient to evaluate its potential for improvement nor to compare the quality of two competing solutions. To overcome this difficulty, the author makes the observation that, for a given number

of workstations m, a solution improves if more stations are full. He proposes the use of the following objective function:

$$\max \sum_{k=1}^{m} \left(\sum_{j \in S_k} t_j \right)^2. \quad (1)$$

He shows that this non-linear objective encourages the transfer of tasks from workstations with lesser loads towards those with higher loads, therefore increasing the chances of emptying some workstations. The procedure starts with a solution found with a priority-based heuristic. It then tries to improve on this solution by considering task exchanges between pairs of stations that keep the solutions "feasible", i.e. solutions always satisfy both all precedence constraints and the cycle time restriction, $t(S_k) \leqslant C$, $\forall k$. The method finds rather good solutions when compared to optimal ones on a set of classical test data (also solved in [17]). However, no information is provided on how much improvement the TS procedure provides with respect to the best solutions obtained with priority-based algorithms or to the initial solutions.

2.2. A new TS algorithm

The new TS algorithm we propose is related to that in [6] but differs from it on two major aspects: (1) it incorporates an intensification–diversification framework based on the use of two different and complementary neighborhoods and (2) it uses a redefinition of the solution space and the objective function in order to allow the algorithm to visit infeasible solutions (i.e. where the load of some workstations exceeds the cycle time C) while searching for better solutions.

2.2.1. Neighborhood structures

It is clear that the solution is improved whenever a station becomes full or empty. We agree with [6] that while trying to attain such a reduction in workstations, half-full workstations should be filled up. But we can also improve things if we can reduce the number of half-empty workstations. If the number of stations that are either almost full or nearly empty increases then there is a better chance that a station could be emptied at the next iteration. We therefore built our approach around the definition of two complementary neighborhoods: *Nh1* that focuses on reducing or increasing the half-empty workstations (i.e. inducing a sort of diversification effect in the search) and *Nh2* that intensifies search on near-empty workstations in order to attempt to empty them completely.

Nh1 is a two task exchange process. At each iteration, it selects the workstation whose workload is closest to the 50% mark. It then considers systematically all possible exchanges between each of the tasks in the selected workstation and the rest of the tasks assigned to other workstations. The improvement in the objective function value is computed for each of these possible moves and the best non-tabu exchange (i.e. the one leading to the best objective improvement or the least degradation) is implemented. Although the number of possible exchanges seems to be huge ($|S_k| \times (n - |S_k|)$, a priori), most of them are in fact not feasible because of the precedence requirements. To maximize the probability that an exchange between a given pair of tasks is feasible, we start exploring exchanges with the stations closest to the selected one and extend away towards more distant stations until precedence constraints are violated. This is done in both directions, upstream and downstream. The best feasible exchange is then implemented. Algorithm 1 illustrates *Nh1*.

Algorithm 1. Explore *Nh*1

```
1: Select target workstation Wk_i
2: k ← −1; violate ← 0
3: for l = 1 to 2 do
4:    for each task s ∈ Wk_i do
5:       j ← i − k
6:       repeat
7:          for each task s′ ∈ Wk_j do
8:             if exchange (s → Wk_j,
                  s′ → Wk_i) is feasible then
9:                evaluate exchange
10:            else
11:               violate ← 1
```

12: $j \leftarrow j - k$
13: **until** *violate* $= 1$ **and** $0 \leqslant j \leqslant m$
14: $k \leftarrow 1$
15: *Implement best exchange found*

Nh2 is a task move process whose purpose is to empty underloaded workstations. At each iteration it selects the emptiest workstation and tries to transfer one of its tasks to some other workstation. Here again, the process considers all non-tabu transfers and implements the one leading to the best objective function improvement or the least degradation. Since the number of tasks n is generally much larger than the number of workstations m, the number of possible transfers in *Nh2* is clearly smaller than in *Nh1* ($(m-1) \times |S_k|$, a priori). The same observation regarding the number of feasible moves and how to structure the neighborhood evaluation also applies here. Algorithm 2 illustrates *Nh2*.

Algorithm 2. Explore *Nh*2

1: *Select target workstation* Wk_i
2: $k \leftarrow -1$; *violate* $\leftarrow 0$
3: **for** $l = 1$ to 2 **do**
4: **for** each task $s \in Wk_i$ **do**
5: $j \leftarrow i - k$
6: **repeat**
7: **if** transfer $s \rightarrow Wk_j$ is feasible **then**
8: evaluate exchange
9: **else**
10: *violate* $\leftarrow 1$
11: $j \leftarrow j - k$
12: **until** *violate* $= 1$ **and** $0 \leqslant j \leqslant m$
13: $k \leftarrow 1$
14: *Implement best transfer found*

Since the choice of the station around which the exchanges are centered is made according to a rigid logic, the algorithm may "cycle" between already visited solutions. This situation occurs despite the tabu mechanisms because it lacks "new" moves. In order to prevent this from happening too often we partially randomize the selection process: a random draw decides if the workstation is selected according to the deterministic ranking rule—with a probability of 3/4—or if it is chosen randomly among all non-tabu workstations.

2.2.2. Search strategy and tabu mechanisms

Given that the two neighborhoods are designed to be complementary, the overall search strategy uses both neighborhoods alternatively. The criterion used to determine which one is in use depends on the status of the search process. We always start with *Nh1* and explore it until a certain number of iterations have been performed without improving the best solution found so far. Then, we switch to *Nh2* and, again, explore it until no improvement can be made according to the same criterion. The search continues by alternatively exploring both neighborhoods until a global stopping criterion is met, the maximum allowed computation time in our case.

The algorithm includes two tabu mechanisms which deal with tasks and workstations respectively. A task becomes tabu when transferred from its current workstation. The task remains tabu for *tabu_length*_2 iterations, where *tabu_length*_2 includes a random factor to avoid cycles on the list. A workstation becomes tabu when one of its tasks is transferred to another workstation and remains tabu for *tabu_length*_1 iterations. This means that when a task is moved from a station, we forbid the addition of a new task to this workstation but allow the removal of other tasks from it.

From the technical standpoint, both mechanisms are implemented via *tabu tags* that store the iteration number until which the workstation or task will be considered tabu. Whenever a workstation or task is declared tabu, its associated tabu tag is set to the value given by *current_iteration* + *tabu_length*. When selecting workstations or tasks during the algorithm search, only workstations or tasks having a tabu attribute value lower than the current iteration number are considered.

2.2.3. Solution space and objective function

As is the case for several other strongly constrained problems, the natural solution space of the SALBP-I is difficult to explore effectively with "simple" moves as the ones described here above.

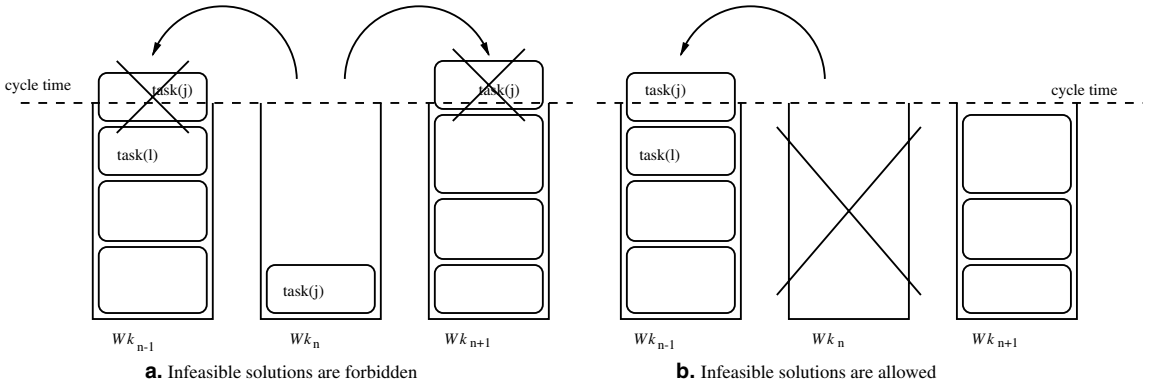

Fig. 1. Enlarging the solution space through cycle time violation. (a) Infeasible solutions are forbidden. (b) Infeasible solutions are allowed.

Indeed, when restricting the search process to feasible solutions, it can become very hard to escape from local optima because of the presence of both precedence and cycle time constraints. Consider the situation illustrated by Fig. 1(a) where the single task in the middle station (Wk_n) cannot be transferred to any of the neighbor stations without violating the cycle time and any task in stations Wk_{n-1} and Wk_{n+1} cannot be moved to respectively Wk_{n-2} and Wk_{n+2} without violating the cycle time or the precedence constraints.

A rather natural and easy way of unlocking the situation is to allow the algorithm to temporarily violate the precedence or the cycle time constraints. Allowing infeasibility in terms of precedence requirements would destroy most of the solution structure and therefore would probably be very difficult to recover from. However, relaxing the cycle time constraint preserves the structure of the solution and can be easily handled through the use of a penalty term in the objective function. As illustrated in Fig. 1(b), task(j) can now be transferred to workstation Wk_{n-1} from where the search process can resume. The algorithm will therefore search the solution space composed of task assignments that always satisfy precedence constraints but may violate the cycle time requirement.

Clearly, when using such a penalization strategy the penalty term should be proportional to the violation amplitude to prevent the search process from straying too deep into infeasibility. We propose the following modified objective function:

$$\max \sum_{k=1}^{m} \left(\sum_{j \in S_k} t_j \right)^2 - p\delta_k, \tag{2}$$

where δ_k is the overload time for workstation k, i.e. the amount of total working time exceeding the cycle time limit, $\delta_k = \max(0, \sum_{j \in S_k} t_j - C)$, and p is a penalty coefficient.

The penalty coefficient p must be chosen carefully. If the penalty term is too large, then the algorithm will almost never consider infeasible solutions. If it is too small, the algorithm will not be able to recover from infeasibility. However, finding an a priori adequate value for p can be quite difficult. In addition, a value that is adequate at one moment of the search process may not be so at another. To circumvent these difficulties, the value of p is changed dynamically according to the search status. Initially, p is set to a large value to discourage infeasibility. We then change its value according to how frequently the algorithm implements infeasible moves. If during the last α itrerations at least β infeasible moves have been selected, then infeasibility is considered too attractive and p is multiplied by a penalty increase factor γ_I. If the current solution is feasible (i.e. no workstation is overloaded) and no infeasible solutions have been implemented during the last α iterations, then infeasibility is considered too expensive and the

Table 1
Characteristics of the proposed TS algorithm

Objective function	Quadratic function (see (2))
Stop criterion	Number of iterations (20,000)
Tabu list	A first tabu list for workstations (length = 5 iterations) and a second tabu list for tasks of length $25 + \tau$ (τ is random $\in \{-5, 5\}$)
Move description	Two types: exchange of two tasks with *Nh1* and transfer of a single task with *Nh2*
Neighborhood switch	After *switch* consecutive iterations without improvement of the best solution so far (*switch* = 50)
Move selection	Best improvement
Strategic oscillation	Allowed. A dynamic penalty is computed in the objective value when a station is overloaded

value of p is therefore multiplied by a penalty decrease factor γ_D. Otherwise, p remains unchanged. This type of automatic parameter adjustment scheme for the penalty function can also be seen as a form of *strategic oscillation*, a TS concept meaning that the exploration of infeasible solutions is alternately encouraged and discouraged.

After a preliminary phase of experiments, we selected the following parameter values for our implementation. We chose to initialize the penalty coefficient with $p = C^2$. We set the penalty increase and decrease factors respectively to $\gamma_I = 1.2$ and $\gamma_D = 0.9$. Finally, we selected the periodicity of the adjustment scheme to $\alpha = 100$ iterations and set the threshold value to $\beta = 10$ infeasible moves.

As can be seen, this adaptive process is quite conservative: the weight of the penalty increases faster than it decreases and it will never decrease as long as the current solution includes an overloaded workstation. The reason for this strategy is that overloading some workstations may enable the algorithm to empty other stations (and therefore close them). However, closing too many stations by overloading some workstations could be dangerous since there might not be enough available space in the remainder of the line to reallocate the overloading tasks. Therefore, we allow overloading only for a limited number of iterations.

In order to provide a brief overview of our TS algorithm, Table 1 presents its most important characteristics and the values used for the remaining parameters. A pseudo-code describing in more details the structure of the algorithm is given in Appendix A. We have implemented and tested this approach on both standard problem sets for the SALBP-I and on a new complex industrial case. The results are presented in the next sections.

3. Computational results on standard problem sets

In order to test the performance of our algorithm with respect to Chiang's best implementation [6]—i.e. best improvement without task aggregation—we have solved the 13 instances of the Arcus 1 and 2 problem sets collected in [21]. All procedures were coded in C++ and the tests were carried out on a Sun workstation UltraSparc 10 (100 MHz).

The first two columns in both sections of Table 2 show the characteristics of each instance, i.e. cycle time C and optimal solution m^*. Then the experimental results obtained respectively by the TS in [6] and by our procedure (identified by LRS for Lapierre, Ruiz and Soriano) are reported. The minimum number of stations is listed under heading m and the running time under cpu. All times are reported in seconds but, since they correspond to different machines and programming languages—the code in [6] is written in C and executed on a VAX 6420 mainframe—care should be taken when comparing them or trying to draw conclusions from them.

LRS found all 13 optimal solutions whereas Chiang's best algorithm reached only 11 out of 13. Despite the evolution of computer speed, the significantly shorter computation times of LRS suggest that our neighborhood strategy is at least as efficient as Chiang's search. However, the small gap between optimal and approximate solutions for these instances does not allow one to clearly assess the improvements provided by the proposed algorithm. In fact, problem sets Arcus 1 and 2 seem to be somewhat too easy to solve: we also got optimal solutions using the simple heuristic COMSOAL [2].

Table 2
Performance comparison between Chiang's TS and the proposed TS algorithm (LRS)

Instances		Chiang		LRS	
C	*m**	*m*	cpu	*m*	cpu
Arcus 1 data set (83 tasks)					
5048	16	17	9.91	16	2.65
5853	14	14	46.62	14	3.10
6842	12	12	15.09	12	3.90
7571	11	11	6.77	11	4.3
8412	10	10	6.86	10	4.95
8898	9	9	61.09	9	5.20
10816	8	8	27.90	8	6.40
Arcus 2 data set (111 tasks)					
5755	27	27	70.5	27	2.75
8847	18	19	63.21	18	4.01
10027	16	16	65.82	16	4.94
10743	15	15	54.46	15	3.78
11378	14	14	38.11	14	5.16
17067	9	9	73.95	9	8.31

Therefore, to have a better appraisal of the performance of our method, we tested it on the problem set suggested by [18,19]. This set is built around the largest SALBP-I proposed in the literature which has 297 tasks and 422 precedence constraints. The 26 instances proposed in the set have all the same tasks and precedence relationships but different cycle times which are given under header *C* in Table 3. For each of the instances, the table reports the best solution found (in terms of workstations) by the priority-based heuristic described in [11] that is used to generate the initial solution of the TS algorithm under heading *Init*, the best solution obtained by the TS procedure under *LRS*, the optimal solution under *Opt* as reported by Scholl and Klein, and the gap between the tabu and the optimal solution (when the latter is known) under header *Gap*. Note that whenever the optimal solution is not known, bounds are provided (e.g. [47–48] for instance $C = 1483$). LRS performed the 20,000 iterations on each instance with individual computation times ranging from 13 to 21 seconds.

As can be seen from Table 3, LRS performs very well, improving on the best heuristic solution most of the times. The method finds three optimal solutions and for all other instances it finds solutions within one workstation of the optimal number (for a gap of less than 4%). The performance seems to decrease somewhat as the cycle time increases and, for cycle times of 1991 or longer, no improvement is achieved over the initial solution. However, it should be pointed out that the initial solutions are already excellent which leaves little room for *LRS* to stand out. In particular, for all cycle times of 1991 and over except one, the heuristic solution is just one workstation over the optimum.

Table 3
Results on Scholl and Klein [18,19] data set

C	*Init*	LRS	*Opt*	*Gap*
1394	53	51	50	1
1422	51	50	50	0
1452	50	49	48	1
1483	49	48	[47–48]	$\leqslant 1$
1515	48	47	[46–47]	$\leqslant 1$
1548	47	46	46	0
1584	47	45	44	1
1620	46	44	44	0
1659	45	43	[42–43]	$\leqslant 1$
1699	43	42	[41–42]	$\leqslant 1$
1742	42	41	40	1
1787	41	40	39	1
1834	40	39	38	1
1883	39	38	37	1
1935	38	37	36	1
1991	36	36	35	1
2049	35	35	34	1
2111	34	34	33	1
2177	33	33	32	1
2247	33	32	31	1
2322	31	31	30	1
2402	30	30	29	1
2488	29	29	28	1
2580	28	28	27	1
2680	27	27	26	1
2787	26	26	25	1

4. Testing on a real life industrial application

Industrial assembly line balancing applications differ somewhat from the problem defined in Section 1. Most of the times, these non-standard assembly line balancing problems include practical considerations that cannot be treated directly by standard algorithms or, when it is possible to adapt them, such a modification is often too

expensive. Authors have traditionally focused on the standard problem. Very few papers explicitly present and solve non-standard assembly line balancing problems derived from real applications. To the best of our knowledge, only [3,5] do so and both deal with the automotive industry. The problem described in [3] is similar to the one presented here. Unfortunately, the paper does not present any computational study evaluating the performance of the proposed heuristic.

In order to contribute to the study of real life versions of single product assembly line balancing problems, we present the following instance coming from a real application of a Canadian manufacturer of home appliances, with 162 tasks and 264 precedence constraints (the data set is available at http://www.crt.umontreal.ca/~sophiel/balancing/). The assembly is carried out on a sequential line with workstations located on both sides of the conveyor, with two conveyor heights and no possible re-positioning of the product. The tasks have attributes that must match the conveyor's height and position attributes. Statistics describing the tasks are summarized in Table 4. Notice that only a few tasks have an "indifferent" attribute, a feature that makes our assembly problem even more different from the academic version than the one presented in [3].

This is a harder problem than the standard one because one also has to take into account attributes for each workstation. The design of good heuristics for multi-attributed assembly line balancing problems is not a trivial issue as shown in [11]. The authors provide a multi-attributed version of the COMSOAL heuristic with pseudo-random attribute selection for workstations. Along the same lines, it was rather easy (and quite a natural approach) to adapt the LRS algorithm to this new context. This is done by constraining the potential moves considered in both neighborhood structures to those involving workstations either sharing the same attributes or having compatible ones with respect to the tasks involved. Moreover, other practical constraints like task incompatibility or task grouping may be easily taken into account using the same type of neighborhood filtering strategies based on attributes.

To solve this problem we therefore generate initial solutions using the heuristic in [11] and then apply our *adapted* TS algorithm. The initial solution contains all the required workstations, with their attributes and the set of tasks allocated to them. The TS algorithm then attempts to improve the initial solution by emptying (i.e. closing) workstations. Note that the original attributes given to the open workstations cannot be modified by the TS procedure.

We carried out several tests using the same initial solution with four versions of our TS in order to evaluate the benefits derived from each of the major components of the procedure. Each version performed the allotted 20,000 iterations on every instance. Computation times ranged from 7 to 13 seconds. The first version (*TS1*) explores only neighborhood *Nh1* while the second version (*TS2*) is based on *Nh2* alone. The third version (*TS3*) implements the intensification–diversification framework that switches from one neighborhood to the other according to the search status. Finally, the fourth version (*TS4*) is the complete version described in Section 2 which, in addition to *TS3*, allows workstation overloading during the search and performs strategic oscillation. Table 5 shows the average number of workstations obtained by each version of TS over 10 trial runs. We also report, in column *Init*, the average values of the initial solutions.

Finally, we decided to evaluate the relative difficulty of solving the real life version of SALBP-I with height and position attributes versus the basic one. In order to evaluate how tasks and workstations attributes define the solution structure with respect to the standard problem, we decided to solve the problem using the same tools—heuristics and our last version of LRS, *TS4*—but neglecting the attribute information. The results for the standard version of the problem are shown in Table 5 under header *SALBP-I*. From Table 5, it can be seen that if *TS2* alone hardly reduces the number of stations, *TS1* always succeeds in improving the original solution. However, *TS3* improves on both *TS1* and *TS2*, confirming that the combination of the two neighborhoods *Nh1* and *Nh2* generates some interesting synergies that improve the overall search process. Table 5 also

Table 4
Task characteristics of our industrial benchmark test

Position	Height			
	Up	Down	Up or down	Total
Front	50	26	3	79
Back	38	35	6	79
Front or Back	3	0	1	4
Total	91	61	10	162

confirms the important contribution of the strategic oscillation feature to the overall performance of the algorithm: *TS4* is able to generate better solutions than *TS3* in most cases. If we compare the two versions of the benchmark test—the basic SALBP-I and the one with attributes—the solutions obtained for the former require less workstations then those of the latter. This is to be expected, since the presence of attributes imposes additional constraints on the problem. Hence, the basic version is a relaxation of the version with side and height attributes.

An interesting observation that can also be made when analyzing the results reported in Table 5 is that the performance of priority-based heuristics decreases significantly when passing from the basic SALBP-I to real life settings in which tasks/workstations attributes need to be considered. Since the LRS algorithm seems to be much less sensitive to these changes, the practical interest of the TS increases with the complexity of the assembly line. Of course, the absolute quality of the solutions provided by the LRS algorithm cannot be assessed accurately since no exact method is available to solve real life assembly line balancing problems such as the one presented here. Nevertheless, these results are quite promising and should encourage further research in this direction to better support the manufacturer's needs.

5. Conclusion

In this paper we presented a novel tabu search algorithm for solving both simple assembly line balancing problems of Type I and non-standard versions of this problem coming from real life applications. Computational results on both standard benchmark instances and real life problems from industry have shown that the method is efficient when compared to the existing ones. On standard problems collected from the literature, our new TS provides results within 1 workstation of the optimal solution. However, existing data sets are not very difficult to solve: the simplest heuristics find optimal solutions regularly. Clearly, more complex test problems are needed in order to evaluate more adequately existing and new algorithms. Along this line, we present a non-standard problem taken from an industrial application. The flexibility of metaheuristics allowed us to easily adapt our TS algorithm to the new specifications. In this richer and more complex problem, our TS outperforms the most popular priority-based heuristics and allows us to confirm the efficiency and adaptability of this type of solution approach for complex problems such as this one.

Acknowledgement

Financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds québécois de la recherche sur la nature et les technologies (FQRNT, previously known as FCAR). This support is hereby gratefully acknowledged.

Appendix A

Algorithm 3. Pseudo-code of the LRS tabu search algorithm

1: Initialize
2: $it \leftarrow 0$
3: $it\_nh \leftarrow 1$
4: $it\_p \leftarrow 0$
5: $nh \leftarrow 1$
6: $p \leftarrow C^2$
7: Generate heuristic initial solution
8: **while** $it < 20000$ **do**
9: Select target workstation
10: Local search(nh)

Table 5
Comparison of the average number of workstations obtained with different TS variants on a SALBP-I with attributes and the basic SALBP-I

Instance C	SALBP-I with attributes					SALBP-I	
	TS4	*TS3*	*TS2*	*TS1*	*Init*	*TS4*	*Init*
75	**27.8**	28.2	32	28.2	32	26	27
80	**24.6**	27	28.8	27.2	29.2	24	25.4
85	**24**	24.4	27.6	24.4	27.8	23	23.8
90	**23.2**	23.4	26.2	23.6	26.4	22	22.2
100	**21.2**	**21.2**	23.8	21.4	23.8	19	20.2
110	20.3	**20.2**	22.4	**20.2**	22.8	18	18.2
125	18.2	18.2	20.2	**17.8**	20.2	16	16.2
135	**15.8**	**15.8**	18.4	16.4	18.8	14.2	15
150	15.2	15.2	16	**15**	16.4	13	13.2
165	**14**	**14**	15.2	**14**	15.4	12	12.2
175	**13.2**	13.4	15.4	13.6	16	11	11.6
185	**11.7**	11.8	13.6	11.8	13.8	11	11
200	**11**	**11**	12.6	**11**	12.6	10	10

11: **if** improvement is reached **then**
12: $it\_nh \leftarrow 0$
13: **if** $it\_nh > switch$ **then**
14: **if** $nh = 1$ **then**
15: $nh \leftarrow 2$
16: else
17: $nh \leftarrow 1$
18: $it\_nh \leftarrow 0$
19: Update tabu lists
20: Update infeasible move counter
21: **if** $it\_p > \alpha$ **then**
22: Update penalty (p)
23: $it\_p \leftarrow 0$
24: $it \leftarrow it + 1$
25: $it\_nh \leftarrow it\_nh + 1$
26: $it\_p \leftarrow it\_p + 1$

References

[1] E.J. Anderson, M.C. Ferris, Genetic algorithms for combinational optimization: The assembly line balancing probelm, ORSA Journal on Computing 6 (1994) 161–173.

[2] A.L. Arcus, COMSOAL: A computer method of sequencing operations for assembly lines, International Journal of Production Research 4 (1966) 259–277.

[3] J.J. Bartholdi, Balancing two-sided assembly lines: A case study, International Journal of Production Research 31 (1993) 2447–2461.

[4] F.F. Boctor, A multiple-rule heuristic for assembly line balancing, Journal of the Operational Research Society 46 (1995) 62–69.

[5] C. Boutevin, M. Gourgand, S. Norre, Stochastic algorithms for the line balancing problem in automotive industry, in: Proceedings on CD-ROM, IFAC'2002, Barcelona, Spain, 21–26 July 2002.

[6] W.C. Chiang, The application of a tabu search metaheuristic to the assembly line balancing problem, Annals of Operations Research (1998) 209–227.

[7] S. Ghosh, R.J. Gagnon, A comprehensive literature review and analysis of the design, balancing and scheduling of assembly lines, International Journal of Production Research (1989) 637–670.

[8] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.

[9] J.F. Gonçalves, J. Raimundo de Almeida, A hybrid genetic algorithm for assembly line balancing, Journal of Heuristics 8 (2002) 629–642.

[10] A.L. Gutjahr, G.L. Nemhauser, An algorithm for the line balancing problem, Management Science 11 (1964) 308–315.

[11] S.D. Lapierre, A. Ruiz, Balancing assembly lines with multi-attributed tasks, Journal of the Operational Research Society, forthcoming.

[12] A.A. Master, An experimental investigation and comparative evaluation of production line balancing techniques, Management Science 16 (1970) 728–745.

[13] K. Park, S. Park, W. Kim, Heuristic for an assembly line balancing problem with incompatibility, range, and partial precedence constraints, Computers & Industrial Engineering 32 (1997) 321–332.

[14] J. Rubinovitz, G. Levitin, Genetic algorithm for assembly line balancing, International Journal of Production Economics 41 (1995) 343–354.

[15] M.E. Salveson, The assembly line balancing problem, Journal of Industrial Engineering 6 (1995) 18–25.
[16] A. Scholl, Balancing and sequencing of assembly lines, second ed., Physica, Heidelberg, 1999.
[17] A. Scholl, S. Voß, Simple assembly line balancing—Heuristic approaches, Journal of Heuristics 2 (1996) 217–244.
[18] A. Scholl, R. Klein, SALOME: A bidirectional branch-and-bound procedure for assembly line balancing, INFORMS Journal on Computing 9 (1997) 319–334.
[19] A. Scholl, R. Klein, Balancing assembly lines effectively—A computational comparison, European Journal of Operational Research 114 (1999) 50–58.
[20] P. Soriano, M. Gendreau, Fondements et applications des methodes de recherche avec tabous, RAIRO Recherche Opérationnelle 31 (1997) 133–159.
[21] F.B. Talbot, J.H. Patterson, An integer programming algorithm with network cuts for solving the assembly line balancing problem, Management Science 30 (1984) 85–99.