Tabu Search With Target Analysis To The Assembly Line Balancing Problems - An Artificial Intelligence Approach

Wen-Chyuan Chiang
Department of Quantitative Methods and Management Information System
College of Business Administration, University of Tulsa
Tulsa, OK 74104
qm_wc@centum.utulsa.edu

Abstract
This paper describes the application of tabu search, a recent heuristic technique for combinatorial optimization problems, to the assembly line balancing problems. Computational experiments with different search strategies have been performed for some assembly line problems from literature. Computational results show that except for few cases tabu search always finds optimal solutions.

Introduction
There are two types of assembly line balancing problems. A Type I problem is to determine the minimum number of workstations required to meet the specified production requirements. A Type II problem is to allocate tasks to workstations in such a way that the maximum time required for assembly at any given station is minimal across all feasible stations. In this paper, we examine the Type I problem and apply tabu search schema to solve it.

The assembly line Balancing problem was first published in a mathematical form by Salveson in 1955. Since then it has been a hot topic for researchers. Master (Master 1966) evaluated the performance of 10 heuristic decision rules by iteratively employing each of the evaluated techniques, increasing the cycle time in one percent increments above the lower bound cycle time until a balance was achieved for the specified number of workstations. Dar-El (Dar-El 1975) investigated 12 heuristic decision rules of Type II problems. Dar-El developed MALB (Dar-El 1973) as a heuristic variant of his earlier optimal-seeking iterative procedure. Dar-El’s general conclusion is that MALB gives consistently superior results to the Arcus (Arcus 1963) or the other techniques investigated. Johnson (Johnson 1988), and Berger et al. (Berger, Bourjolly & Laporte 1992) investigated a branch and bound algorithm to solve Type I problems. Anderson (Anderson 1994), and Leu and Matheson (Leu & Matheson 1994) combined genetic algorithms and heuristic criteria to solve the assembly line balancing problem. Easton (Easton 1990) applied dynamic programming approach and used upper bounds in solving assembly line balancing. Carraway (Carraway 1989) used dynamic programming approach to solve stochastic assembly line balancing problems. Suresh and Sahu (Suresh & Sahu 1994) used simulated annealing to solve stochastic assembly line balancing problems. Shin and Min (Shin & Min 1991) investigated stochastic assembly line balancing problem in just-in-time environment.

In this paper, tabu search is applied to solve type I assembly line balancing problems. Tabu search was introduced by Glover (Glover 1989) as a technique to overcome local optimality. The underlying idea is to forbid some search directions at a present iteration in order to avoid cycling, but to be able to escape from a local optimal point. This strategy can make use of any local improvement techniques. There are many problems that are successfully solved using tabu search (Skorin-Kapov 1990, Knox 1994). In this paper, the application of tabu search to assembly line balancing problem is discussed.

Assembly Line Balancing Problem
The objective of assembly line balancing is to allocate tasks into workstations so that the total idle time across all workstations is minimized.

Lemma 1. In order to minimize the total idle time across all workstations, the number of workstations should be minimized.

Let $T_{ij}$ be the time to finish the jth task in workstation i, $T_j$ be the time to finish task i, $I_{ij}$ be the idle time in station j, n be the number of workstations, m be the total number of tasks, $CT$ be the cycle time, and $k_i$ be the number of tasks assigned to workstation i; then Total idle time across all the workstations =

$$\sum_{i=1}^{n} I_{ij} = \sum_{i=1}^{n} (CT - \sum_{j=1}^{k_i} T_j)$$

$$= n \times CT - \sum_{i=1}^{n} \sum_{j=1}^{k_i} T_j = n \times CT - \sum_{i=1}^{n} T_i$$
From the above formula, we can see that $CT$ and $\sum_{i=1}^{n} T_i$ are constant. Therefore in order to minimize total idle time across all workstations, the number of workstations must be minimized.

In order to encourage as many tasks as possible to be conglomerated into a big workstation with less idle time, a nonlinear objective function is used. The objective function can be written as:

$$\max \sum_{i=1}^{n} (\sum_{j=1}^{k_i} T_j)^3 \quad \text{where} \quad n \text{ the number of workstations and} \quad k_i \text{ is the number of tasks in workstation} \ i.$$ 

Lemma 2. The objective function can be maximized by moving jobs from workstations with less total time to workstations with larger total time.

There are two cases. Case 1, two workstations $s_1$ and $s_2$ can be combined into one workstation. Let $ST_1$ and $ST_2$ be the total time in workstations $s_1$ and $s_2$.

The Objective function after combination

$$= (ST_1 + ST_2)^3 = ST_1^3 + ST_2^3 + 2ST_1ST_2 > ST_1^3 + ST_2^3$$

= Objective function before combination.

Case 2, two workstations $s_1$ and $s_2$ cannot be combined into one workstation because of cycle time constraint, if total time in $s_2$ is greater than total time in $s_1$, we can still improve the solution by moving some tasks from $s_1$ to $s_2$ and therefore reduce the size of $s_1$ and increase the chance of combining $s_1$ with other workstations and get rid of workstation $s_1$. Suppose processing time $\Delta$ is moved from $s_1$ to $s_2$, total time for these two workstations after the move is $ST_1 = ST_1 - \Delta$, $ST_2 = ST_2 + \Delta$.

Objective function after combination

$$= ST_1^3 + ST_2^3 = \left(ST_1 - \Delta\right)^3 + \left(ST_2 + \Delta\right)^3$$

$$= ST_1^3 - 2\Delta ST_1 + \Delta^3 + ST_2^3 + 2\Delta ST_2 + \Delta^3$$

$$= ST_1^3 + ST_2^3 + 2\Delta(\Delta + (ST_2 - ST_1)) > ST_1^3 + ST_2^3$$

= Objective function before combination.

Therefore to maximize $\sum_{i=1}^{n} (\sum_{j=1}^{k_i} T_j)^3$ is the same as to minimize number of workstations.

Assembly line balancing problem can be written as

$$\max \sum_{i=1}^{n} (\sum_{j=1}^{k_i} T_j)^3$$

subject to

$$\sum_{j=1}^{k_i} T_{ij} \leq CT \quad \text{where} \quad k_i \text{ is the number of tasks}$$

in workstation $i$ and $CT$ is cycle time

$$T_{ij} \neq T_{il} \text{ if } i \neq k \text{ or } j \neq l$$

$$\sum_{i=1}^{n} k_i = m \quad \text{where} \quad m \text{ is the number of tasks to be assigned}$$

The search of solution for assembly line balancing problem consists of two stages: initial solution construction which generates a feasible initial solution, and tabu search improvement which takes an initial solution and improves it.

Relational Matrix and Warshall Algorithm

There are precedence relationship among tasks, which specifies the order in which the tasks must be performed in the assembly process. Certain tasks must be finished before other tasks can be done. Immediate precedence relationship among tasks can be represented by a relational matrix $M = \{M_{ij}\}$ where

$$\begin{cases} 
1 & \text{if task} \ i \ \text{must be finished immediately before task} \ j \\
0 & \text{otherwise}
\end{cases}$$

Precedence relationship between any pair of tasks $i$ and $j$ can be defined as task $i$ must be finished (not necessarily immediately) before task $j$ can start. Task $i$ is prior to task $j$ if either

1. $i$ is immediately before task $j$, i.e. $M_{ij} = 1$, or
2. There exists a task $k$, $i$ is prior to $k$ and $k$ is immediately before task $j$.

Precedence relationship can also be represented by a matrix $MT = \{MT_{ij}\}$ where

$$MT_{ij} = \begin{cases} 
1 & \text{if task} \ i \ \text{must be finished before task} \ j \\
0 & \text{otherwise}
\end{cases}$$

In fact, precedence relationship is the transitive closure of immediate precedence relationship $M = \{M_{ij}\}$. From graph theory we know that $MT_{ij} = 1$ if and only if there

Chiang 31
exists a path from task i to task j. Warshall (Warshall 1962) developed a very efficient algorithm for calculating transitive closure matrix. Let n be the number of tasks, \( M \) be the matrix representing immediate precedence relationship, and \( M_T = \{ M_T^r \} \) be the matrix representing precedence relationship. The Warshall algorithm can be represented as follows:

Step 1. Copy matrix \( M \) to matrix \( M_T \).
Step 2. For \( i \) from 1 to \( n \) do step 3 to 5
Step 3. For \( j \) from 1 to \( n \) do step 4 to 5
Step 4. If \( M_{j,i} = 1 \) then do step 5, otherwise continue step 3
Step 5. For \( k \) from 1 to \( n \)
set \( M_T^{i,j} \) to be \( M_T^{i,k} \odot M_T^{k,j} \)

where the behavior of operator \( \odot \) can be represented by the following table:

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>non zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>non zero</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Matrix \( M_T \) can be very useful to determine the feasibility of a solution.

### Tabu Search (TS)

The development of Tabu Search can be traced back to the late 1960s and early 1970s. Its contemporary version was proposed by Glover (Glover 1989). The basic idea of TS is to improve a solution using memory-guided rules to obtain good solutions.

TS introduces a memory structure that forbids or penalizes certain moves that would return to a recently visited solution. In assembly line balancing problem, the flexible memory is defined as follows:

```c
int tabu[MAX_JOBS][MAX_STATIONS]
int tabusize
```

The above two dimensional array \( tabu \) is used to check if a move from a solution to its neighborhood is allowed. If \( tabu[i][s] \) is 0, then job \( i \) is free to move from its current workstation to another workstation \( s \). Otherwise, say \( tabu[i][s] \) is 6, job \( i \) cannot move to workstation \( s \) in the next 6 iterations. After a job \( i \) moved from workstation \( s \) to another workstation, the value of \( tabu[i][s] \) is assigned to a value called \( tabu \) size, which means that job \( i \) cannot go back to workstation \( s \) in the next \( tabu \) size iterations.

After each iteration, all nonzero values in flexible memory \( tabu \) are reduced by 1. When an entry \( tabu[i][s] \) is reduced to 0, a job is allowed to move back to workstation \( s \) again.

The following example can be helpful to understand this flexible memory. Suppose there are 6 jobs assigned to 3 workstations:

| jobs 1 and 2 are in workstation 1 | jobs 3 and 4 are in workstation 2 | jobs 5 and 6 are in workstation 3 |

The initial values of all entries in \( tabu \) are all set to be 0 and \( tabu \) size is 3. In iteration 1, it is decided to exchange jobs 1 and 3. Figure 1 shows the flexible memory \( tabu \) after the exchange. Both \( tabu[1][1] \) and \( tabu[3][2] \) are set to be 3 because job 1 cannot go back to workstation 1 and job 3 cannot go back to workstation 2 in the next 3 iterations.

### Figure 1 Solution and \( tabu \) memory after iteration 1

Suppose in iteration 2, it is decided to move job 2 to workstation 3, \( tabu[2][1] \) are set to be 3 because job 2 cannot go back to workstation 1 in the next 3 iterations. After iteration 2, \( tabu[1][1] \) and \( tabu[3][2] \) are reduced by 1 which means that job 1 cannot return to workstation 1 and job 3 cannot return to workstation 2 in the next 2 iterations. Solution and \( tabu \) memory after iteration 2 are shown in Figure 2.

### Figure 2 Solution and \( tabu \) memory after iteration 2
Suppose in iteration 3, if we could move job 1 from workstation 2 to workstation 1, we could get a solution that is better than the best solution we had found so far. However according to taboo flexible memory, job 1 cannot go to workstation 1 for the next two iterations. If we strictly follow taboo search methodology, we could miss an optimal solution. An additional rule called aspiration can solve this problem.

Aspiration Criterion

When a move can lead to a solution better than the best solution obtained so far, this move is allowed even if it is in taboo. This rule is called an aspiration criterion (Glover 1989). In the above situation, job 1 is allowed to move to workstation 1 even if this move is still in taboo. Solution and tabu memory after iteration 3 are shown in Figure 3.

Aspiration criterion is a very important rule in taboo search. It allows a move to get out of taboo status temporarily and therefore makes the quality of result solution less dependent on taboo size. Usually the greater the taboo size is, the less chance for solution to be trapped in local optima. However using greater taboo size could also eliminate many opportunities to find better solution if aspiration criterion were not used.

Intensification and Diversification

Besides the above described components, taboo search requires some additional rules to make it more intelligent to find better solutions. The use of flexible memory has been limited to a short term horizon, i.e. to remember the most recent moves to avoid being trapped to local optima.

The intensification scheme in Tabu search uses long term memory to guide its search of solutions. According to Glover (Glover 1989), it can be used to encourage solutions to satisfy such properties and discourage solutions that violate them. We would like to narrow the neighborhood in the search process to favor solutions with properties that occurred often in good solutions previously visited.

In assembly line balancing, the idea is to allocate as many jobs as possible to each workstation so that the number of workstations can be minimized. The rule of intensification in the case of assembly line balancing problem can be stated as follows:

When a job j moves to workstation s and makes s to reach its full capacity, this move is believed to be good and job j is fixed to workstation s in the next few iterations, unless a solution which is better than the best solution found so far can be found by moving job j to another workstation.

The diversification scheme is another strategic pursuit of solutions with varying characteristics which provides an essential counterbalance to the intensification component of taboo search. (Glover 1989) In assembly line balancing problem, diversification can be achieved by introducing a penalty function into the objective function. Let switch(j) be the number of times job j switches from one workstation to another. The penalty function for moving job j to workstation s can be defined as

\[
\text{penalty function} = \begin{cases} 
0 & \text{if the move can improve current solution} \\
\text{switch}(j) \times 10 & \text{otherwise}
\end{cases}
\]

The change of objective function

\[
= \text{new objective function value} - \text{old objective function value} - \text{penalty function}
\]

Since in assembly line balancing, we are trying to maximize objective function, in each improvement step, we search the neighborhood to find a move which has the maximal change of objective function. When a job switched too many times, its chance to be selected as next move is reduce and therefore the chances for other jobs are increased so that the search region is forced to those areas that have not been searched before.

References


