TABU SEARCH APPROACH FOR TYPE 1 PROBLEM OF ASSEMBLY LINE BALANCING

Duong Vo Hung
University of Technology – VNU-HCM
(Received 17 June 2003)

ABSTRACT: *In this research, a new approach for solving type 1 problem of assembly line balancing was proposed. This method employed Tabu search algorithm with first improvement approach. The solutions thus obtained have been compared with the results of some of the existing models and methods. The result of this research provided not only optimal number of workstations (n) but also associated range of cycle time (C).*

1. Introduction

From beginning of 1950s, research on the problem of assembly line balancing (ALB) was initiated. ALB problems can be classified under two types [9]: Type 1: required cycle time is specified it is required to design a production line of minimum line length or minimum number of workstations. Type 2: A line length is specified it is required to design a production line with minimum cycle time or minimise maximum total processing time of all tasks assigned to any workstation (minimax) or maximum output. Till date, there are many techniques for solving ALB problems. Bowman [1] has developed two different linear programming models, and required integer solutions. Patterson and Albracht [10] have presented zero-one programming. Held, et al., [7] proposed the use of dynamic programming, the exact solutions of small ALB problems are presented. Suresh, et al., [12] have proposed the use of genetic algorithm-1 (GA1) and genetic algorithm-2 (GA2) for sub-optimal or near optimal solutions. Yasunori, et al., [16] have developed Hopfield neural network, a new method, for solving large line balancing problems. Suresh and Sahu [11] have applied Simulated Annealing (SA) technique to solve ALB problems. Helgeson and Birnie [6] have developed the ranked positional weights method for solving ALB problems. Kilbridge and Wester [6] have proposed a three-step procedure method to solve ALB problems. Moodie and Young [6] have proposed the largest candidate rules for line balancing, with small size problems, the largest candidate rules offers a quick solution compare to ranked positional weights method.

The advantages of computer, now, aided to solve large ALB problems. Arcus [6] has employed COMSOAL, this method needs large number of iterations so it is suited to computer programming. CALB [6] was developed, it can be used for both single-model and mixed-model lines. ALPACA [6] was first implemented in 1967, ALPACA was designed to cope with the complications on the assembly lines.

Tabu search (TS) is a heuristic applied for finding a near optimal solution of combinatorial optimization problems. Up to now, TS has been successfully applied to many different fields in industry [5]. TS prevents being trapped local optima by using flexible memory. Now, TS is employed for solving type 1 of ALB problems.

2. Type 1 problem

Objective function: is minimum number of workstations or minimum length of production line. This can be stated as:

$$\text{Min. } Z = n$$

Where: n: number of workstations.

To minimise number of workstations we must assign as many tasks as possible to a workstation to reduce its idle time [3]. Therefore, the objective function can be written as:

$$Max.\ Z = \sum_{i=1}^{n}\left(\sum_{j=1}^{k_i} t_{ij}\right)^2$$

Where:k_i: is the number of tasks in workstation i.

Maximising the objective function helps to reduce the number of workstations.

3. Background and Implementation

Tabu is a metaheuristic applied for finding a near optimal solution of combinatorial optimization problems. Basically, it consists of several elements called the move, neighbourhood, initial solution, search strategy, memory, aspiration function, and stopping rules.

Move and neighbourhood: move is defined as beginning of fundamental notion. The move is the function which transforms a current solution into another solution. A movable subset of solution is called as neighbourhood.

Initial solution: the Tabu search procedure starts from an initial solution. We must find an initial solution from which the algorithm could start.

Search strategy: at each step, the neighbourhood of a given solution is searched. We follow some principles or rules to find a better solution, to prevent cycling and to stop.

Memory: we have at least two classes of the memory: a short-term memory for the very recent history and a long-term memory for distant history. We have employed a class of short-term memory as the Tabu list which is the basic rule. This list prevents turning back to the solution visited in the previous steps.

Aspiration function: is defined as evaluating the profit in taking a forbidden move. If this profit is acceptable then the Tabu status of the move is dropped and the move can be performed.

Stopping rules: is designed as an instance:

a. find a solution which is close enough to the given lower bound of the goal function.
b. perform maximum of iteration moves with a view to improving the best solution obtained so far.
c. or time limits on the number of iterations.

The Tabu search algorithm is shown in figure 1.

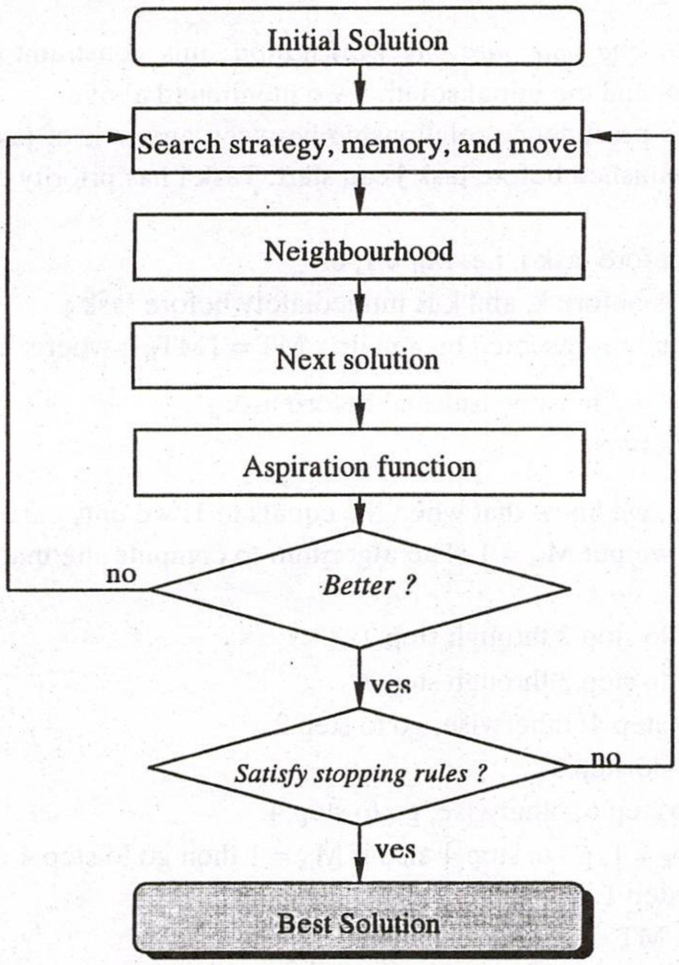

Figure 1. **Tabu Search Algorithm Chart**

4. Avoiding a local optima

In the case of non-improvement, choose the least non-improvement and accept the move. Therefore, the objective value obtained is worst than the current solution. This can be a serious problem if the search algorithm achieves the best solution too soon. It means that the best solution is obtained before the maximum number of iterations are completed. After this, the solutions worst than the best will be searched, until the iteration number equals the maximum number of iterations and the algorithm stops. Thus, it may not be possible to conclude whether the current solution is the optimal.

To avoid this, a memory to store the best solutions that has been obtained is employed. In this case the best objective variable is used to obtain the best objective value and then use the best solution to obtain the solution. When the algorithm stops, the solution stored in the best memory is the best one.

5. Satisfaction of constraints

(i) Cycle time constraint: this constraint is checked whenever a move is performed. We only check the workstation which receives the task moved. For example, if workstation w receives one more task, which has processing time t_w, from another workstation check if:

$$\sum_{j=1}^{k_w} t_{wj} + t_w \leq C$$

If this constraint is satisfied we proceed to check for the next constraint; otherwise, reject the move.

(ii) Only one task is assigned to one and only one workstation: this constraint is always satisfied because of this algorithm and the initial solution we mentioned above.

(iii) Precedence relationship: The precedence relationship between any pair of tasks i and j can be defined as task i must be finished before task j can start. Task i has priority over task j if either:

1) Task i is immediately before task j, i.e. $M_{ij} = 1$, or

2) There exists a task k, i is before k, and k is immediately before task j.

The precedence relationship can be represented by a matrix $MT = \{MT_{ij}\}$, where:

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

According to two priorities above, we know that when M_{ij} equals to 1, we only care M_{jk}, for k = 1 to m. If M_{jk} equals to 1 then we put $M_{ik} = 1$. The algorithm to compute the matrix MT as follows:

```
Step 1:    for i from 1 to m, do step 2 through step 7.
Step 2:    for j from 1 to m, do step 3 through step 7.
Step 3:    if $M_{ij} = 1$, then do step 4; otherwise, go to step 2.
Step 4:    for k from 1 to m, do step 5.
Step 5:    if $M_{jk} = 1$, then do step 6; otherwise, go to step 4.
Step 6:    if $k \geq j$ then let $M_{ik} = 1$, go to step 4 else if $M_{ik} = 1$ then go to step 4 else
           let $M_{ik} = 1$, go to step 1.
Step 7: copy matrix M to matrix MT.
```

This algorithm is very simple but the required computation time may be longer. Therefore, in some existing models Warshall's algorithm is employed for computing matrix MT.

This matrix is used for feasibility check, it means that the precedence relationship constraints will be replaced by feasibility check constraints in the main program.

6. Algorithm for solving type 1 problem

Now, TS algorithm for solving ALB problems was developed by Wen-Chyuan Chiang [3], it means direct method. However, in this research, the type 1 problems can be solved via the type 2 problems, it means indirect method.

Lemma 1: in line balancing problems, if number of workstations n are increased from lower bound to upper bound, then that the first value of n that yields a feasible solution is optimum value of n.

Proof: we know that the first problem minimizes number of workstations n, so when n is increased, it releases the constraints of problem until the first value of n makes the problem feasible. If n is reduced only one unit, it makes the problem infeasible. If n is increased until N, the problem is still feasible, but that value N is larger than n. Hence, the value of n is optimum value.

In line balancing problems, upper bound of n equals to number of tasks and lower bound of n is the smallest integer which is greater than total processing time divide cycle time.

Using lemma 1, we can solve the first problem step by step from lower bound to upper bound until we get the first value of n. When n is assigned a certain value, the problem will be solved by TS algorithm for type 2 problem. It means that we find the minimal value of cycle time C1, then we compare C1 to C. If C1 is less than or equal to C the algorithm stops and that value of n is optimal, otherwise the value of n increases by one unit and solve problem again until the cycle time constraint is satisfied. The feasibility check constraints are satisfied when type 2 problem is solved. (This algorithm is presented in figure 2).

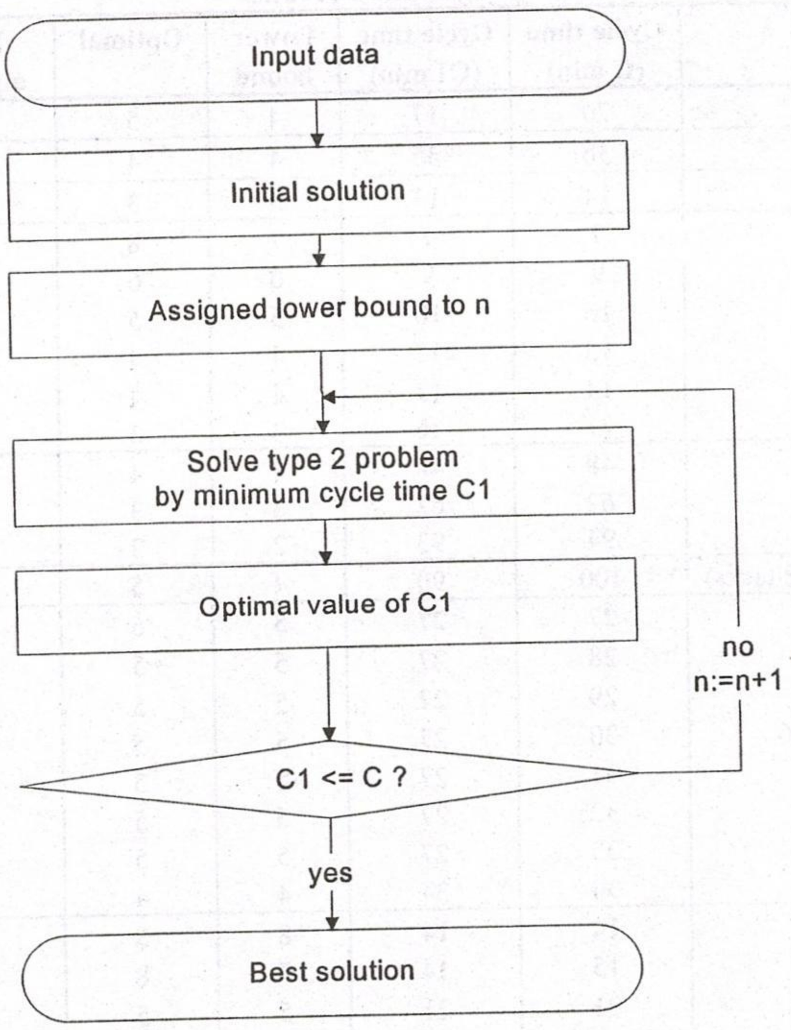

Figure 2: **Algorithm Procedure Chart**

This method may takes a little bit longer time because it has many steps if the optimal value is far from lower bound, but in the real problem the optimal value is often near lower bound. TS, however, is a very fast method for solving ALB problems. Moreover, the idea behind this method is that the solutions give us many informations about value of n, specially in range of cycle time C. Therefore, we easily make decision about production lines and scale of production.

As mentioned above, TS is an improvement algorithm, and hence large the number of iterations chances of obtaining better results will increase. However, large number of iteration may take long computer running time. In this study, maximum number of iterations ranges between 100 to 1000 depending upon the size of problem. Larger the problem size, the larger would be the number maximum iterations. Associated with number of iterations is Tabu size which ranges between 7 to 40 depending upon the number of iterations used.

7. Validation test and results:

In this method, we only consider first improvement approach because best improvement takes longer time and its solutions is not good enough. The result from this study is summarized in table 1.

Table 1: Algorithm's results

Problems	Cycle time (C min)	Cycle time (C1 min)	Lower bound	Optimal	Results of algorithm (n)
Bowman (8 tasks)	20	17	4	5	5
Hopfield (8 tasks)	36	36	4	4	4
Hoffman (9 tasks)	14	13	3	3	3
Jackson (11 tasks)	7	7	7	8	8
	9	9	6	6	6
	10	10	5	5	5
	13	12	4	4	4
	14	12	4	4	4
	21	16	3	3	3
Dar-el (11 tasks)	48	48	4	4	4
	62	62	3	3	3
	94	93	2	2	2
Kilbridge – Wester (12 tasks)	100	90	4	5	5
Kenneth – Ramsing (16 tasks)	27	27	5	6	5
	28	27	5	5	5
	29	27	5	5	5
	30	27	5	5	5
	31	27	5	5	5
	32	27	5	5	5
	33	27	5	5	5
	34	34	4	4	4
Mitchell (21 tasks)	14	14	8	8	8
	15	14	7	8	8
	21	21	5	5	5
	26	21	5	5	5
	35	35	3	3	3
	39	35	3	3	3
Wooden car toy (22 tasks)	11	11	5	5	5
Author's case study 41 tasks	**552**	**465**	**5**	**N/A**	**5**
Kilbridge – Wester (45 tasks)	57	56	10	10	10
	79	79	7	7	7
	92	92	6	6	6
	110	110	5	6	6
	138	138	4	4	4
	184	184	3	3	3
Tonge (70 tasks)	176	N/A	N/A	N/A	N/A
	364	338	11	11	11
	410	**371**	**9**	**9**	**10**
	468	463	8	8	8
	527	527	7	7	7

Remarks

Cycle time C1: cycle time associated with optimal solution

Lower bound: the smallest integer is greater than or equal value of total processing time divides cycle time

8. Evaluation and discussion

In the type 1 problems of ALB, 39 examples were solved to evaluate the validation of algorithm. The performance of this algorithm is presented in table 2

Table 2: Performance of algorithm

Number of cases	Number of optimal solutions obtained	***Percentage of optimality***
39	38	*97.44 %*

This algorithm has yielded high percentage of optimality (97,44%). It obtained only one non-optimal solution in Tong's problem with cycle time 410. However, its solutions include two parts that are optimal number of workstations (n) and the associated cycle time (C1). We know about range of cycle time, any cycle time belongs to that range has the same optimal solution n. For example, in the first case (Bowman with 8 tasks), with cycle time form 17 to 20 minutes, we have the same optimal solution n which equals 5 workstations. This is a contribution of this research.

9. Conclusions and recommendations

In this research, one new approach, which applied TS for solving type 1 via type 2 problem of ALB, was developed. This algorithm employed first improvement approach for solving problem. The solution methodology is general enough and can be applied for real life industrial problems because of high percentage of optimality. Moreover, the information from result provides optimal number of workstation (n) and associated with range of cycle time (C1) which help mangers making decision about production lines and changing scale of production because of fluctuation demand.

Further study in this direction is recommended: develop Tabu search based methodology for mixed models, parallel lines or U-shape lines, etc. To consider some parameters that can effect the line balancing such as the variable processing times of tasks, stochastic models etc.

10. Some illustration charts

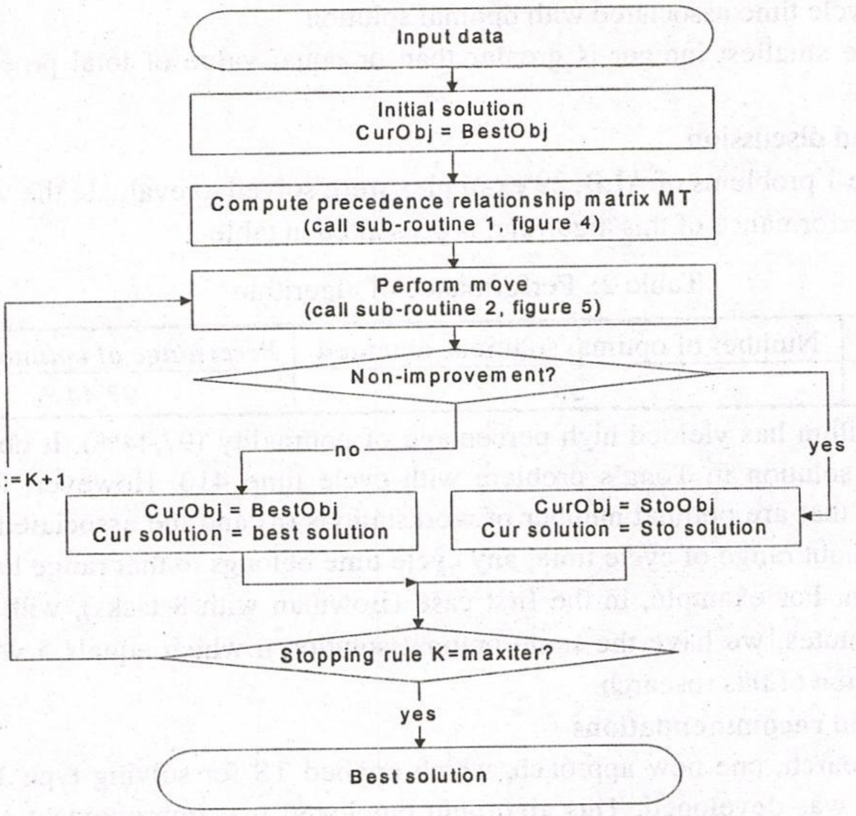
Input data
Initial solution
CurObj = BestObj
Compute precedence relationship matrix MT
(call sub-routine 1, figure 4)
Perform move
(call sub-routine 2, figure 5)
Non-improvement?
no
yes
no K:=K+1
CurObj = BestObj
Cur solution = best solution
CurObj = StoObj
Cur solution = Sto solution
Stopping rule K=maxiter?
yes
Best solution

Figure 3: **Tabu search algorithm procedure for the first improvement**

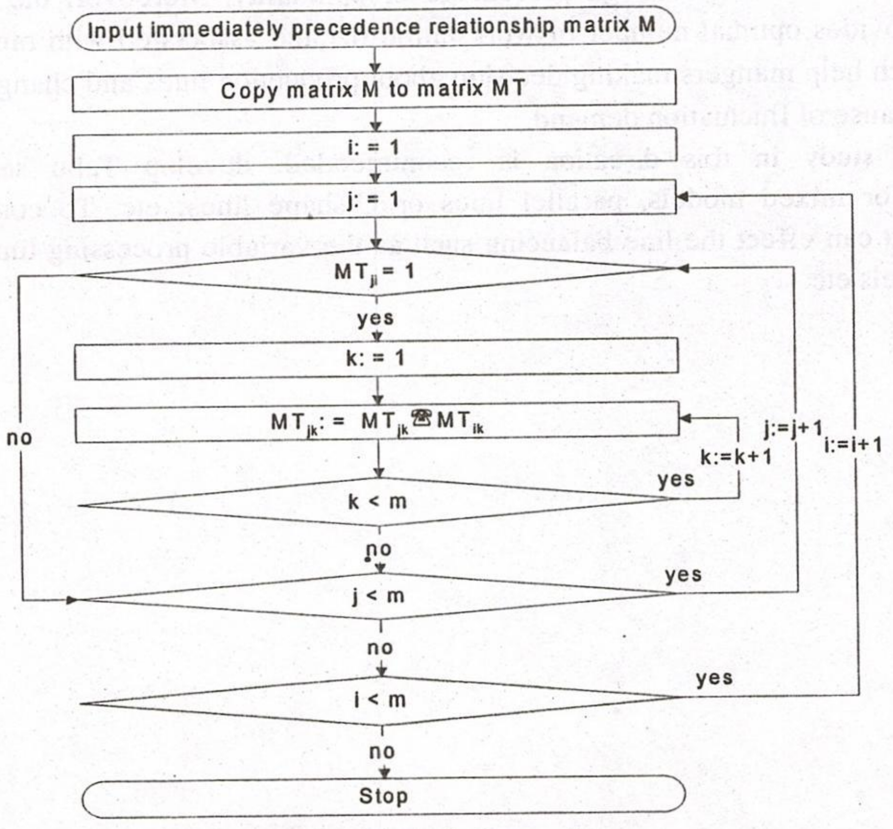

Figure 4: **Sub-routine 1 Warshall procedure**

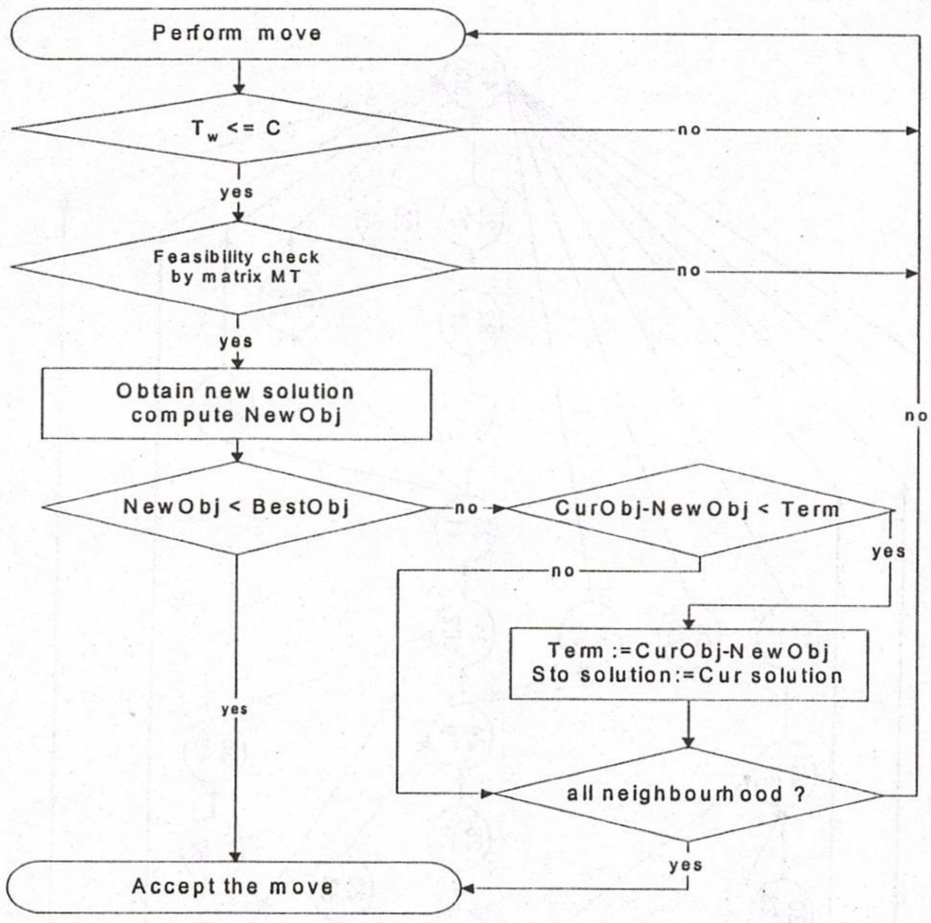

Figure 5: **sub-routine 2 performing move for the first impovement**

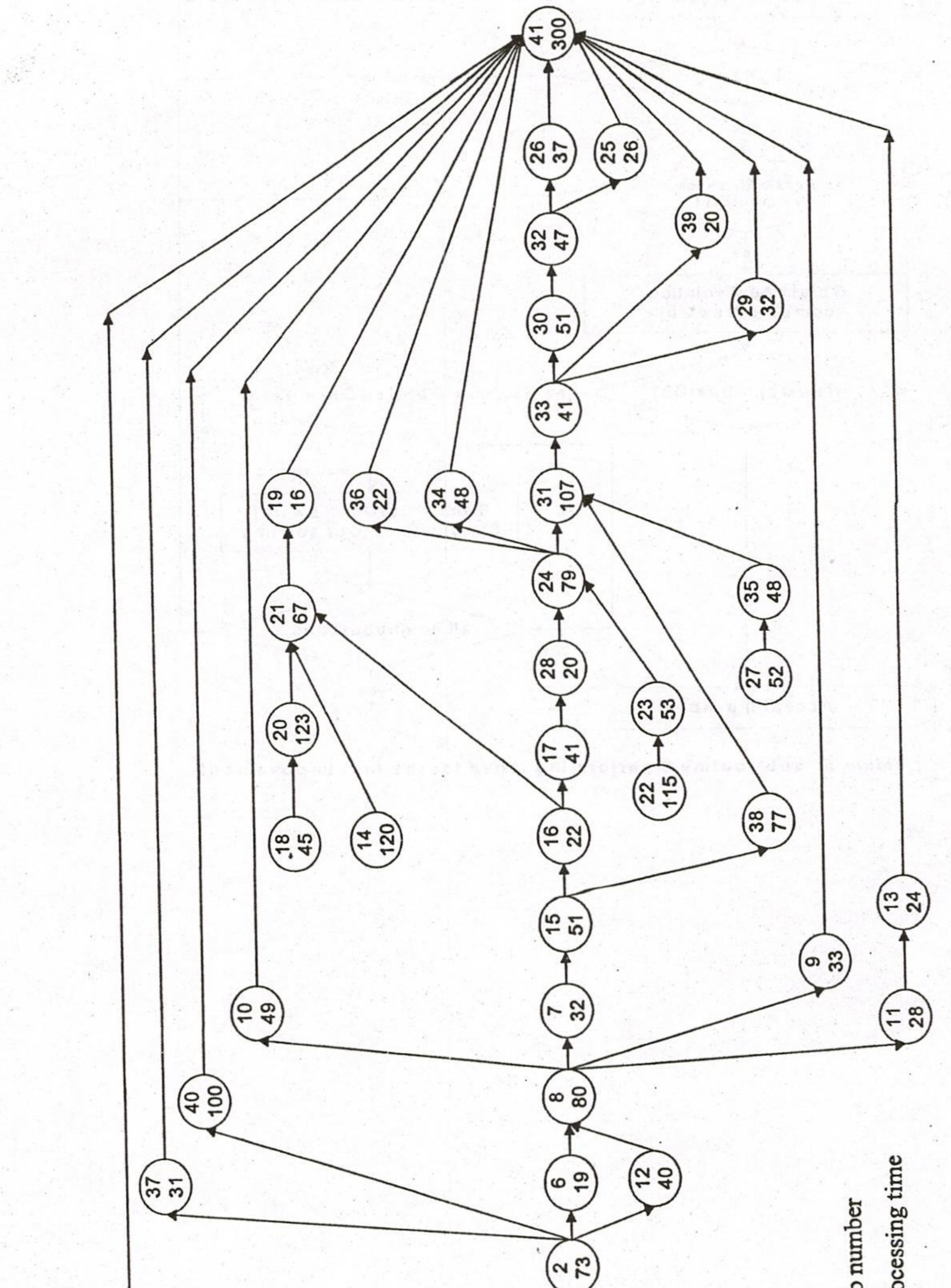

Figure 6: **Author's case study - Precedence relationship diagram**

ỨNG DỤNG GIẢI THUẬT TABU CHO BÀI TOÁN CÂN BẰNG DÂY CHUYỀN SẢN XUẤT DẠNG 1

Đường Võ Hùng
Khoa Quản lý Công nghiệp, Trường Đại học Bách khoa – ĐHQG-HCM

TÓM TẮT: Trong nghiên cứu này, một hướng giải mới cho bài toán cân bằng dây chuyền sản xuất dạng 1 đã được phát triển. Giải thuật này ứng dụng thuật toán TABU với cách chọn lời giải tốt hơn đầu tiên làm lời giải tiếp theo. Giải thuật được kiểm chứng và so sánh với những lời giải của những phương pháp khác đã được công bố trên các tạp chí khoa học. Kết quả của giải thuật không những chỉ cung cấp thông tin về số trạm làm việc tối ưu (n) mà còn khoảng thời gian chu kỳ (C) tương ứng.

REFERENCES

[1] Bowman, E. H., Assembly line balancing by Linear Programming. *Operation Research*, Vol. 8, No. 3, 385-389 (1960).

[2] Buffa, E. S. and Sarin R. K., *Modern Production / Operation Management*, Eighth Edition, John Willey & Sons, Inc (1987).

[3] Chiang, W. C., The Application of a Tabu Search Metaheuristic to The Assembly Line Balancing Problem. *Annals of Operation Research*, Vol. 77, 209-227 (1998).

[4] Dar-el, E. M., Solving Large Single Model Assembly Line Balancing Problems - A Comparative Study. *AIIE Transactions*, Vol. 7, No. 3, 302-310 (1975).

[5] Glover, F., Tabu Search: A Tutorial. *Interfaces*, Vol 20, No. 4, 74 – 94 (1990).

[6] Groover, M. P., *Automation, Production System and Computer Integrated Manufacturing*. Prentice Hall, Inc. (1992).

[7] Held, M., Karp, R. M., and Shareshian, R., Assembly line balancing - Dynamic Programming with precedence constraints. *Operation Research*, Vol. 11, No. 3, 442-459 (1963).

[8] LINGO, *User's guide.* LINDO Systems Inc. (1995).

[9] Mastor, A. A., An Experimental Investigation and Comparative Evaluation of Production Line Balancing Techniques. *Management Science*, Vol. 16, No. 11, 728-746 (1970).

[10] Patterson, J. H., and Albracht, J. J., Assembly line balancing: Zero-One Programming with Fibonacci Search. *Operation Research*, Vol. 23, No. 1, 166-172 (1975).

[11] Suresh, G. and Sahu, S., Stochastic assembly line balancing using Simulated Annealing. *International Journal of Operation Research*, Vol. 32, No. 8, 1801-1810 (1994).

[12] Suresh, G., Vinod V. V., and Sahu, S., A Genetic Algorithm for assembly line balancing. *Production Planning and Control*, Vol. 7, No. 1, 38-46 (1996).

[13] Tonge, F. M., A Heuristic Programming for Assembly Line Balancing, *The Rand Corporation - Santa Monica - California* (1960).

[14] Warshall, S., A Theorem of a Boolean Matrix. *Journal of ACM*, Vol. 9, 11-12 (1962).

[15] Wild, R., *Mass-Production Management/The Design and Operation of Production Flow lines Systems.* John Wiley & Son (1990).

[16] Yasunori, H., Ikuko, N., *Tohru*, W. and Hidekatu, T., Line Balancing Problems Using a Hopfield Network. *Japan - USA Symposium on Flexibility Automation - A Pacific Rim Conference* - Kobe Japan, 1369 - 1375 (1994).