An Implementation View on Job Shop Scheduling Based on CPM

MILOŠ ŠEDA
Institute of Automation and Computer Science
Brno University of Technology
Technická 2, 616 69 Brno
CZECH REPUBLIC

Abstract: - The scheduling of manufacturing processes aims to find sequences of jobs on given machines optimal by a selected criterion, e.g. minimal completion time of all operations. With respect to NP-hardness of these problems and the necessity to solve them by heuristic methods, the problem representation and the effectiveness of their procedures is substantial for computations to be completed in a reasonable amount of time. In this paper, we deal with job shop scheduling problem (JSSP) in a disjunctive graph-based representation. Turning all undirected edges into directed ones, the problem is transformed to a problem solvable by the Critical Path Method (CPM). We propose an original implementation of the CPM that makes it possible to decrease its time complexity and thus also the running time of all JSSP iterations.

Key-Words: - manufacturing process, job shop scheduling, NP-hard problem, disjunctive graph, CPM

1 Introduction

Practical machine scheduling problems are numerous and varied [3]. They arise in diverse areas such as flexible manufacturing systems, production planning, logistics, communication, computer design etc. A scheduling problem is to find sequences of jobs on given machines with the objective of minimising some function of the job completion times.

In a simpler version of this problem, flow shop scheduling, all jobs pass through all machines in the same order. A more complex case is represented by a job shop scheduling problem where machine orderings can be different for each job.

Job shop scheduling problem (abbreviated to JSSP or JSS) is one of the hardest combinatorial optimization problems. It belongs to the class of NP-hard problems, consequently there are no known algorithms guaranteed to give an optimal solution and run in polynomial time. That means, classical optimization methods (branch and bound method, dynamic programming) can only be used for small-scale tasks. Therefore, more complex tasks must be solved by heuristic methods [10], [12]. Successful heuristic methods include approaches based on simulated annealing [6], tabu search [11], [14], and genetic algorithms [9]. A very efficient method combines a variable depth search procedure with a shifting bottleneck framework [2], [15].

2 Mathematical Model of JSSP

The classical JSS problem can be described as follows [13]: There are a set of m machines and a set of n jobs. Each job consists of a sequence of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. Each machine can process at most one operation at a time. We assume that any successive operations of the same job are processed on different machines. A schedule is an assignment of the operations to time intervals on the machines.

The problem is to find a schedule which optimises a given objective. Assume that three finite sets J, M, O are given where J is a set of jobs 1, … , n, M is a set of machines 1, … , m, and O is a set of operations 1, … , N.

Consider the following denotations: J_i = the job to which operation i belongs, M_i = the machine on which operation i is to be processed, t_i = the start time for operation i, p_i = the processing time for operation i, $C_{\max}$ = the makespan.

On O, a binary relation $\rightarrow$ is defined that represents *precedence constraints* between operations of the same job.

If $i\rightarrow j$, then $J_i=J_j$ and there is no $k\in\{i,j\}$satisfying $i\rightarrow k$ or $k\rightarrow j$. (Operation i is the predecessor of operation j). Thus, if $i\rightarrow j$, then $M_i \neq M_j$ by the JSSP specifications.

The problem of optimal job shop scheduling is to find a starting t_i time for each operation $i \in O$ such that

$$\max_{i \in O}\left(t_i + p_i\right) \tag{1}$$

is minimised subject to:

$$\forall i \in O \ : \ t_i \geq 0 \tag{2}$$

$$\forall i, j \in O,\ i \rightarrow j \ : \ t_j \geq t_i + p_i \tag{3}$$

$$\forall i, j \in O,\ i \neq j,\ M_i = M_j \ : \ (t_j \geq t_i + p_i) \vee (t_j \geq t_i + p_j) \tag{4}$$

The conditions (3) express *precedence constraints* which represent technological link-up of operations within the same task. The conditions (4) express *machine capacity constraints*, i.e. each machine can process at most one operation at a time.

The described equations cannot be directly used for determining a schedule. We need to eliminate symbols of binary relation $\rightarrow$ and disjunction $\vee$ and try to get a formulation of *integer programming*.

The binary relation can be eliminated easily so that O will be decomposed into subsets of operations that correspond to tasks. Then we will assign to operations in each task numbers creating a sequence of consecutive integers by the operation order.

Denote n_j = the number of operations in job j, and N_j = the total number of operations of the first j jobs.

Evidently:

$$N_0 = 0,\ N_j = \sum_{k=1}^{j} n_k,\ N = \sum_{k=1}^{n} n_k. \tag{5}$$

Using the denotation for total number of operations of the first j–1 jobs, we assign to n_j operations of task j numbers $N_{j-1} + 1,\ \ldots\ ,\ N_{j-1} + n_j$ where $N_{j-1} + n_j = N_j$.

Now we can express equation (3) as follows:

$$\left(\forall j \in J\right)\left(N_{j-1} + 1 \leq i \leq N_j - 1\right):\ t_{i+1} \geq t_i + p_i \tag{6}$$

The makespan is then determined as the maximum of the completion times of the last operations in jobs. Hence, we get:

$$\forall j \in J : C_{\max} \geq t_{N_j} + p_{N_j} \tag{7}$$

Let us define capacity constraints using binary variables $x_{ij} \in \{0,1\}$ as follows:

$$\forall i, j \in O, i \neq j, M_i = M_j :$$

$$x_{ij} = \begin{cases} 1, t_j \geq t_i + p_i, \text{ operation } i \text{ precedes } j \\ 0, t_i \geq t_j + p_j, \text{ operation } j \text{ precedes } i \end{cases} \tag{8}$$

If T is an upper bound of the makespan, then, using x_{ij}, we can replace equation (4) by pairs of inequalities (9) as follows:

$$\forall i, j \in O, i \neq j, M_i = M_j : \begin{cases} t_j \geq t_i + p_i x_{ij} - T\left(1 - x_{ij}\right) \\ t_i \geq t_j + p_j\left(1 - x_{ij}\right) - T x_{ij} \end{cases} \tag{9}$$

Hence, the job shop scheduling problem with makespan objective can be formulated as follows:

Minimise

$$C_{\max} \tag{10}$$

subject to

$$\forall i \in O :\ t_i \geq 0 \tag{11}$$

$$\left(\forall j \in J\right)\left(N_{j-1} + 1 \leq i \leq N_j - 1\right):\ t_{i+1} \geq t_i + p_i \tag{12}$$

$$\forall j \in J : C_{\max} \geq t_{N_j} + p_{N_j} \tag{13}$$

$$\forall i, j \in O, i \neq j, M_i = M_j : \begin{cases} x_{ij} \in \{0,1\} \\ t_j \geq t_i + p_i x_{ij} - T\left(1 - x_{ij}\right) \\ t_i \geq t_j + p_j\left(1 - x_{ij}\right) - T x_{ij} \end{cases} \tag{14}$$

3 Disjunctive Graph-Based Representation

An important feature of heuristic methods is problem representation. In [4], a review of frequently used representations is presented. Here, we briefly describe only one based on disjunctive graphs.

The *disjunctive graph-based representation*: A disjunctive graph is defined as follows:

$$G = \left(V, C \cup D\right) \tag{15}$$

where

V is a set of vertices representing operations. This set contains also two special vertices numbered 0 and N+1 representing the fictitious start and end operations, respectively. The processing time of each operation is denoted as the weight of the vertex. The two fictitious operations 0 and N+1 have operation times of zero.

C is a set of directed *conjunctive* edges. These edges represent pairs of consecutive operations of the same job, as well as edges from the start vertex 0 to the first operation of each job and edges from the last operations of each job to the end vertex N+1.

D is a set of undirected *disjunctive* edges representing pairs of operations to be processed by the same machine.

To determine a schedule we must define an ordering of all operations processed on the same machine. It can be done by turning all undirected (disjunctive) edges into directed ones. Evidently, there are

$$2^{m_1} 2^{m_2} \cdots 2^{m_{|M|}} = e^{(m_1+m_2+\cdots+m_{|M|})\ln 2} \qquad (16)$$

possible disjunctive graphs, where m_i is the number of operations on machine i. Some of them are infeasible because of cycles.

A set S of all directed edges selected from disjunctive edges is called a *complete selection*. A complete selection S defines a feasible schedule if and only if the resulting directed graph is acyclic which guarantees there are no precedence conflicts between operations, see Figure 2. Obviously, the time required to complete all jobs (makespan) is given by the length of the longest weighted path from the start vertex to the end vertex in a directed graph $G(S)=(V,C\cup S)$, where S is an acyclic complete selection. This path is called the *critical path* and is composed of a sequence of *critical operations*.

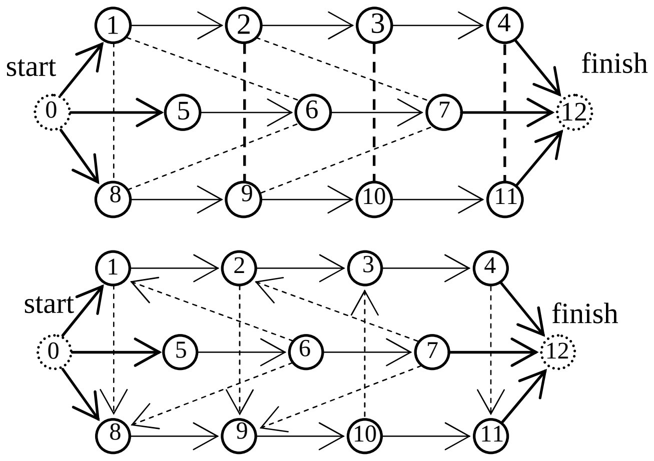

Fig. 1: Disjunctive graph for the JSSP instance and a feasible schedule

Using Critical Path Method (CPM) [7], we easily get the earliest possible start times of operations and the corresponding schedule.

The key operator of the tabu search and simulating annealing methods is one used to construct a neighbourhood of the current solution in which these algorithms search for a solution to be used in the next iteration. In the literature, many sophisticated strategies can be found. For lack of space, we only mention the neighbourhood search strategy of Nowicki and Smutnicki [11], [14]. It is based on modifications of critical blocks that create a critical path evaluated by the CPM. These blocks are given by maximal sequences of consecutive critical operations on the same machine.

For a single (arbitrarily selected) critical path u and critical blocks $B_1, \dots, B_r$ defined for u, it swaps the first (and the last) two operations in blocks $B_2, \dots, B_{r-1}$. In the first block B_1 it swaps only the last two operations, and, via symmetry in the last block B_r, it swaps only the first two operations. These swaps define the set of moves from the processing order p_i. This set of moves is not empty only if the number of blocks is greater than one ($r > 1$) and if there exists at least one block with the number of elements greater than one. The neighbourhood of p_i is then defined as all the processing orders obtained by applying moves from p_i. This strategy implemented within the framework of a tabu search led to the best known results for benchmarks from the OR-Library.

4 CPM and Its New Implementation

Given a network of one project in *A-on-A* graphical representation (activity = arrow (directed edge)) with weights of edges (i,j) determined by activity durations t_{ij}, we can calculate, for each activity, by means of the well known *Critical Path Method* (*CPM*) [1], [7], [8] the *earliest possible start time* (the earliest possible time that the activity can begin); the *earliest possible finish time* (the earliest possible time plus the time t_{ij} needed to complete the activity); the *latest allowable finish time* (the latest time an activity can end without delaying the project) and the *latest allowable start time* (the latest allowable finish time minus the time needed to complete the activity) and *total* (*activity*) *slack* or *total* (*activity*) *float* (the amount of time by which the start of a given activity can be delayed without delaying the completion of the project).

In A-on-A graphical representation, the following notions are referred to start and end vertices of the network graph: $T_i^{(0)}$ represents the earliest possible start time of vertex i, $T_j^{(1)}$ the latest allowable finish time of vertex j. $TS_{ij} = T_j^{(1)} - T_i^{(0)} - t_{ij}$ is the total slack of activity (i,j).

4.1 CPM for topologically sorted network graph

A classical version of the CPM implementation is based on that *topological ordering* [5] of the vertices which means that, for each edge (i, j), i appears before j in the ordering, that is, $i<j$.

Let n_t be the number of the network graph terminal, n_0 the number of the network graph origin, and m the number of edges.

The denotation n_0 is used only for the sake of generality because in literature the origin is numbered by 0 or 1.

Then, a topological ordering of the vertices can be computed as follows:

1. Start from the origin and assign n_0 to it.
2. Leave all edges outgoing from n_0 and assign numbers $n_0+1, \dots, n_0+k_1$ to k_1 vertices that have no input edge.
3. Leave all edges outgoing from the vertices numbered in the previous step and assign numbers $n_0+k_1+1, \dots, n_0+k_1+k_2$ to k_2 vertices that have no input edge.
4. Continue this way until all vertices are numbered.

In the program realization of this algorithm, we keep for each vertex a value $C(v)$ that determines the number of unselected vertices adjacent to v. At the beginning, we initialise $C(v)$ by 0 for each vertex v and then, for all end vertices w of directed edges (v, w), we make the assignment $C(w):=C(w)+1$. This process is applied to each vertex v of edges (v, w) satisfying $C(v)=0$. Pruning the edges outgoing from a vertex v numbered in the previous step corresponds to executing the statements modifying the value $C(w)$ of end vertices of edges (v, w) by $C(w) :=C(w)-1$.

The time complexity of the topological sort is $O(|V|+|E|)$ [5].

If a network graph is topologically ordered, then the earliest possible start times are determined as follows:

$$T_{n_0}^{(0)} = 0 \tag{17}$$
$$T_j^{(0)} = \max_{(i,j)\in E} \left\{ T_i^{(0)} + t_{ij} \right\}, \quad j = n_0+1, n_0+2, \dots, n_t$$

The following two equations are used to determine the latest allowable finish times of the vertices:

$$T_{n_t}^{(1)} = T_{n_t}^{(0)} \tag{18}$$
$$T_i^{(1)} = \min_{(i,j)\in E} \left\{ T_j^{(0)} - t_{ij} \right\}, \quad j = n_t-1, n_t-2, \dots, n_0$$

From these equations it is obvious that the time complexity of both the earliest possible start times and the latest allowable finish times calculations is equal to $O(|V||E|)$.

The phase of the topological sort included, the total time complexity of the algorithm is therefore $O(|V|+|E|+|V||E|)$.

4.2 CPM for network with lexicographical ordering of edges

Consider a *lexicographical ordering* of the edges defined in the following way: Let $e_1, e_2 \in E$, $e_1 = (a_1, a_2)$, $e_2 = (b_1, b_2)$, then

$$e_1 \prec e_2 \Leftrightarrow \exists i_0 \in \{1,2\} : \left(a_{i_0} < b_{i_0}\right) \wedge \left(\forall i \left(1 \le i < i_0\right) : a_{i_0} = b_{i_0}\right) \tag{19}$$

In this definition, we do not consider the edges equality because each network graph is simple and thus all edges differ minimally in one vertex. Using this ordering, we can simplify the formulae for the earliest possible start and latest allowable finish times calculations so that the pairs of nested cycles in the algorithm are replaced by simple sequential cycles:

Formally, we express this for the earliest possible start times of the vertices as follows:

$$\forall i \in V : T_i^{(0)} = 0 \tag{20}$$
$$\forall (i,j) \in E : T_j^{(0)} = \max \left\{ T_j^{(0)},\ T_i^{(0)} + t_{ij} \right\}$$

Notice that the cycle $\forall (i,j) \in E$ is processed by the ordering $\prec$ from the "lowest" edge to the "highest" edge.

Next two equations make it possible to determine the latest allowable finish times of the vertices where, in the cycle $\forall (i,j) \in E$, we use the "inverse" ordering from the highest edge to the lowest one:

These formulae imply that, in this type of ordering, the time complexity of the vertices times is $O(|V|+|E|)$. However, time complexity of the lexicographical sort must also be considered. (10) From the most efficient sorting algorithms *MergeSort*, *HeapSort* and *QuickSort*, we selected the last one for the reason of its simple implementation. The time complexity of QuickSort is $O(|E| \log_2|E|)$. As the lexicographical sort is preceded by the topological sort, the total time complexity of this CPM implementation is given by (12) $O(|V|+|E|+|V|+|E|+|E| \log_2|E|)$, which can be simplified to $O(|V|+|E|+|E| \log_2|E|)$.

Now we can write a pseudopascal version of the described CPM implementation for network graphs using a lexicographically ordered list of edges. Here *ne* denotes the number of edges of network graph, i.e., it equals $|E|$. The user-defined type `net_graph` represents an array of records describing edges. Their items are given by SV = starting vertex of edge, EV = ending vertex of edge, t = weight of edge represented by the activity duration and TS = total slack of activity. The user-defined type `ivector` is used for arrays of numbers assigned to vertices.

```
procedure CPMlex(var graph: net_graph; ne, nv0,
nv: integer; var T0,T1: ivector) ;
    var  i: integer ;
    begin   for $i := nv0$ to $nv$ do
                $T0[i] := 0$;
            for $i := 1$ to $ne$ do
                with $graph[i]$ do
                    if $T0[SV] + t > T0[EV]$
                        then $T0[EV] := T0[SV] + t$ ;
                                                { max }
            for $i := nv$ downto $nv0$ do
                $T1[i] := T0[nv]$;
            for $i := ne$ downto 1 do
                with $graph[i]$ do
                    if $T1[EV] - t < T1[SV]$
                        then $T1[SV] := T1[EV] - t$ ;
                                                { min }
            for $i := 1$ to $ne$ do
                with $graph[i]$ do
                    $TS := T1[EV] - T0[SV] - t$
    end ;
{ CPMlex - CPM implementation for network graph
using a lexicographically ordered list of edges }
```

5 Conclusions

In this paper, we presented a mathematical model of the job shop scheduling problem. Based on a mixed integer programming formulation, it could be used for computation in such optimization tools as GAMS and LINDO. Obviously, because of NP-hardness of the model, they can only get an optimal solution for small JSSP instances. Therefore, other representation schemes, more suitable for computations by approximation or heuristic methods, must be searched. As frameworks of these methods are sufficiently known, we focused on the key neighbourhood operator of the probably best known algorithm proposed by Nowicki and Smutnicki to disjunctive graph-based representation of JSSP.

Since JSSP in the disjunctive graph-based representation is based on the CPM calculation step, the efficiency of this step plays a significant role here. In this paper, a new implementation of the CPM was proposed. After the usual topological sort of vertices, an additional step (lexicographical sort of edges) is included. It was shown that for a network graph $G=(V,E)$ the time complexity of this version of CPM is equal to $O(|V|+|E|+|E|\log_2|E|)$, and thus it is lower than the time complexity of the classical CPM implementation using topological ordering of vertices.

Further investigation will include fuzzy versions of these problems where two cases of uncertainties can be obtained - uncertain due dates and uncertain processing times.

Acknowledgments
The results presented have been achieved using a subsidy of the Ministry of Education, Youth and Sports of the Czech Republic, research plan MSM 0021630518 "Simulation modelling of mechatronic systems".

References:

[1] A. Azaron, C. Perkgoz and M. Sakawa, A Genetic Algorithm for the Time-Cost Trade-off in PERT Networks, *Applied Mathematics and Computation*, Vol. 168, 2005, pp. 1317-1339.

[2] E. Balas and A. Vazacopoulos, Guided Local Search with Shifting Bottleneck for Job Shop Scheduling, *Management Science*, Vol. 44, 1998, pp. 262-275.

[3] J. Blazewicz, K.H. Ecker, G. Schmidt and J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin, 1996.

[4] R. Cheng, M. Gen and Y. Tsujimura, A Tutorial Survey of Job-Shop Scheduling Problems Using Genetic Algorithms – I. Representation, *Computers & Industrial Engineering*, Vol. 30, No. 4, 1996, pp. 983-997.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2001.

[6] A. El-Bouri, N. Azizi, S. Zolfaghari, A Comparative Study of a New Heuristic Based on Adaptive Memory Programming and Simulated Annealing: The Case of Job Shop Scheduling, *European Journal of Operational Research*, 2007, 17 pp., in press

[7] S.E. Elmaghraby, *Activity Networks: Project Planning and Control by Network Models*, John Wiley & Sons, New York, 1977.

[8] S.M.T.F. Ghomi and E. Teimouri, Path Critical Index and Activity Critical Index in PERT Networks, *European Journal of Operational Research*, Vol. 141, 2002, pp. 147-152.

[9] J. Goncalves, J. de Magalhaes-Mendes and M. Resende, A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem, *European Journal of Operational Research*, Vol. 167, 2005, pp. 77-95.

[10] Z. Michalewicz and D.B. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, Berlin, 2002.

[11] E. Nowicki and C. Smutnicki, A Fast Taboo Search Algorithm for the Job Shop Problem, *Management Science*, Vol. 42, 1996, pp. 797-813.

[12] C.R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford, 1993.

[13] R. Vaessens, E. Aarts and J. Lenstra, Job Shop Scheduling by Local Search, *INFORMS Journal on Computing*, Vol. 8, 1996, pp. 302-317.

[14] J.P. Watson, A. Howe and L. Whitley, Deconstructing Nowicki and Smutnicki's i-TSAB Tabu Search Algorithm for the Job-Shop Scheduling Problem, *Computers & Operations Research*, Vol. 33, 2006, pp. 2623-2644.

[15] H. Wenqi and Y. Aihua, An Improved Shifting Bottleneck Procedure for the Job Shop Scheduling Problem, *Computers & Operations Research*, Vol. 31, 2004, pp. 2093-2110.