ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 134 Nr kol. 1554

Wojciech BOŻEJKO[1], Mieczysław WODECKI[2]
[1]Politechnika Wrocławska
[2]Uniwersytet Wrocławski

PARALLEL ALGORITHM FOR SOME SINGLE MACHINE SCHEDULING PROBLEMS

Summary. Problem of scheduling a single machine to minimize total weighted late job can be described as follows: there are n jobs to be processed, each job has an integer processing time, a weight and a due date. The objective is to minimize the total weighted late job, where the late job is performed after its due date. The problem belongs to the class of NP-hard problems. In the paper, we propose sequential and parallel (for SIMD model computing) branch and bound algorithms based on elimination criteria. Finally, the computation results and discussion of the performance of algorithms are presented.

ALGORYTM RÓWNOLEGŁY DLA PEWNEGO JEDNOMASZYNOWEGO PROBLEMU SZEREGOWANIA ZADAŃ

Streszczenie. W pracy zajmujemy się problemem optymalizacji kolejności wykonywania zadań na jednej maszynie, w którym kryterium optymalności jest suma kosztów zadań spóźnionych. Jest on oznaczany przez $n|1||\sum w_i U_i$ i należy do klasy problemów silnie NP-zupełnych. Przedstawiamy algorytm równoległy (dla modelu SIMD) oparty na metodzie podziału i oszacowań, w którym wykorzystano kryteria eliminacyjne.

1. Introduction

The single machine weighted number of late jobs problem known as $n|1||\sum w_i U_i$ is NP-hard (Karp [3]). This problem can be stated as follow: Each of n jobs (numbered 1, ..., n) is to be processed without interruption on a single machine which can handle only one job at a time. For each job i let p_i, d_i w_i be as follows: *processing time*, *due date* and *weight* (which is the penalty incurred if the job is completed after its due date). The objective is to sequence the jobs so that the total penalty is minimized.

In the literature there are known optimal algorithms to solve the above problem which use dynamic programming (Lawler and Moore [4] – pseudopolynomial algorithm which requires $O(n\min\{\sum_j p_j, \max_j\{d_j\}\})$ time, Sahni [8] – when all weights are integer which requires $O(n\min\{\sum_j p_j, \sum_j w_j, \max_j\{d_j\}\})$ time). There are also branch and bound algorithms (Villarreal and Bulfin [10], Potts and van Wassenhowe [5],[6]). However considering that these algorithms are memory and time expensive it is very hard to calculate bigger problems. Therefore, in the last years we can see intensive development of approximate algorithms especially those which use various ideas of local search (tabu search, simulated annealing).

Sections 2 and 3 contain problem description and methods to solve them. Next sections describe branch and bound algorithm and computer simulations of sequential and parallel algorithms.

2. Problem formulation and enumeration scheme

Let $N=\{1,2, \ldots ,n\}$ be the set of all tasks and Π the set of permutations of elements from N. For permutation $\pi\in\Pi$ let us denote $C_{\pi(i)}$, $(C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)})$ to be a due date of the job i in permutation π (i.e. when tasks are executed in order of appearance in π). Then let us denote:

$$U_i = \begin{cases} 0, & \text{when } C_i \le d_i, \\ 1, & \text{otherwise,} \end{cases}$$

to be a *tardiness* of the job and $f_i(C_i)= w_iU_i$ the penalty. The weight (*penalty*) of the permutation is defined by:

$$F(\pi) = \sum_{i=1}^{n} w_i U_i.$$

Therefore, we have to calculate the optimal permutation in our problem (which has minimal weight) in the set of all permutations Π.

For any set $Q\subseteq N$, we define $P(Q) = \sum_{i\in Q} p_i$.

Generation process of permutations from set Π we will present as a search tree. We create this tree as follows. From the root node (zero level), where no jobs have been scheduled, we branch to n different nodes on the first level, each node corresponding to a

specific job being scheduled in the n-th position. Each of these nodes leads to n-1 new nodes on the second level, corresponding to one of the remaining n-1 jobs filling the (n-1)-th position.

Each node π from k-th level ($k = 0,1,2,...,n$) is characterized by a set of *fixed jobs* $S_\pi^* = (\pi(k+1), \pi(k+2), ..., \pi(n))$ and *free jobs* $S_\pi = (\pi(1), \pi(2), ..., \pi(k))$ where $|S_\pi| = k$, $S_\pi^* \cup S_\pi = N$ and $S_\pi^* \cap S_\pi = 0$. Producing new permutation β from π (new node on (k-1)-th level of tree) consists in fixing on k-th position in β one of the free jobs from the set S_π, i.e. change positions of fixed job with the job which is on k-th position in π and include it in the set of fixed jobs S_β^*. Remaining jobs on the same positions are in both permutations. Obviously in each successor of permutation β the job fixed on k-th position in β will still remain.

$$\begin{array}{cccccccccc} 1 & 2 & & k-1 & k & k+1 & & n-1 & n & \leftarrow \textit{position} \end{array}$$
$$\pi = \underbrace{(\pi(1), \pi(2), ..., \pi(k-1), \pi(k)}_{S_\pi}, \underbrace{\pi(k+1), ..., \pi(n-1), \pi(n))}_{S_\pi^*}$$

3. Eliminations criteria

Each node π in the search tree is characterized by a set S_π of unscheduled jobs. We define a precedence relation, that this, a partial order on the job's set S_π, such that an optimum sequence exists that satisfies this precedence relation.

This relation will be denoted by $\mathcal{R}$, and $i\mathcal{R}j$ means that job i must be performed before job j, ($i,j \in S_\pi$). In this case job j is called *descendant* of job i and job i is called an *ascendant* of job j. Sets $A_i = \{j \in S_\pi: i\mathcal{R}j\}$ and $B_i = \{j \in S_\pi: j\mathcal{R}i\}$ will, respectively, denote indices of the descendants and the ascendants of job $i \in S_\pi$. In the following, we restricted ourselves to schedules which satisfy the precedence constrains. By determining so many of these precedence relation $\mathcal{R}$, we present theorems on so-called *eliminations criteria*.

Theorem 1 [1]. If for a job $l \in S_\pi$ we have:

$$d_l \geq P(S_\pi),$$

then we only have to consider schedules whereby l comes last among the jobs in S_π.

Theorem 2 [9]. If for two jobs $i,j \in S_\pi$ we have:

$$d_i \geq P(S_\pi \setminus A_j),$$

then we only have to consider schedules whereby i precedes j.

Theorem 3 [2]. If for two jobs $i,j \in S_\pi$ we have:

$$d_i \leq d_j, \quad w_i \geq w_j, \quad p_i \leq p_j,$$

then we only have to consider schedules whereby i precedes j.

Theorem 4. If jobs $i,j \in S_\pi$ and:

$$P(B_j) + p_j \geq \max\{d_l : l \in S_\pi \text{ and } w_l / p_l \geq w_j / p_j,$$

then we only have to consider schedules whereby i precedes j.

These theorems are checked in every node of a search tree.

4. Lower bound

Let π be a node of the tree H on k-th level, $S = \{\pi(1),\ldots,\pi(k)\}$ - the set of free jobs and $S^* = \{\pi(k+1),\ldots,\pi(n)\}$ - the set of fixed jobs. A lower bound LB of the costs of all possible schedules generated from π can be defined as follow:

$$LB(\pi) = F(S^*) + LB(S),$$

where:

$$F(S^*) = \sum_{i=k+1}^{n} f_{\pi(i)}\left(C_{\pi(i)}\right),$$

is the cost of executing fixed jobs from the set S^* and $LB(S)$ is the lower bound of executing free jobs from the set S. We will calculate $LB(S)$ using two methods.

A Lower Bound from the greedy method.

Lower bound $LBG(S)$ of free jobs execution costs can be calculated as follows:

Algorithm LBG

$LBG(S) := 0; W := S; p := P(W);$
Execute k times:
if there exists $i \in W$ such as $d_i \geq P(W)$
then $W := W \setminus \{i\}$ and $p := p - p_i$
else
$$LBG(S) := LBG(S) + \min_{i \in W}\{f_i(\beta)\} \text{ and } p := p - \max_{i \in W}\{p_i\}$$

It is easy to prove that for any permutation γ of jobs from the set S, $F(\gamma) \geq LBG(S)$.

A Lower Bound from the Assignment Problem.

Let

$$T_i(q) = \min\{P(Q) : Q \subset S \setminus \{B_i \cup A_i \cup \{i\}\},$$

where $q = |Q|$. Next

$$t_{ij} = P(B_j) + p_i + T_i\left(j - |B_i| + 1\right),\ |B_i| \le j \le |S \setminus A_i|,$$

and

$$c_{ij} = \begin{cases} f_i(t_{ij}) & \text{for } |B_i| \le j \le |S \setminus A_i|, \\ \infty & \text{for } 1 \le j \le |B_j| \text{ and } |S \setminus A_i| < j \le k. \end{cases}$$

A lower bound $LBAP(S)$ of the free jobs execution from the set S is equal to the optimal solution of the following assignment problem:

$$\text{Minimize: } \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}x_{ij},$$

$$\text{for } x_{ij} \in \{0,1\},\quad \sum_{i=1}^{k} x_{ij} = 1,\ j = 1,2,\ldots,k \quad \text{and} \quad \sum_{j=1}^{k} x_{ij} = 1,\ i = 1,2,\ldots,k.$$

In paper [7] the lower bound for $n|1|\sum w_iT_i$ problem is calculated similarly.

Let $LB(S) = \max\{LBG(S), LBAP(S)\}$.

Theorem 5. If

$$LB(\pi) \ge F(\pi^*),$$

then permutation π can be eliminated.

5. Branching rule

We will set jobs from the *candidate set* K on k-th position in permutation π ($|S_\pi|=k$):

$$K(\pi) = S \setminus \bigcup_{i=1}^{k} B_i.$$

For each job $l \in K$ let us denote the following indication:

$$\Delta(l) = f_q\left(C_l - p_l + p_q\right) + \sum_{j=r+1}^{k-1} f_{\pi(j)}\left(C_{\pi(j)} - p_l + p_q\right) + f_l\left(C_q\right) - f_l\left(C_l\right) - \sum_{j=r+1}^{k-1} f_{\pi(j)}\left(C_{\pi(j)}\right) - f_q\left(C_q\right),$$

where $l = \pi(r)$ and $q = \pi(k)$.

Theorem 6. If β is a permutation generated from π by fixing of the job $l \in K(\pi)$ on the position k, then:

$$F(\beta) = F(\pi) + \Delta(l).$$

Therefore, expression $F(\pi)+\Delta(l)$ is a permutation weight generated from π by fixing free job s on k-th position. While the algorithm progresses we will choose jobs that after fixing will generate permutation – direct successor which has the smallest possible weight (i.e. which has the smallest $\Delta(i)$, $i \in K(\pi)$).

6. Branch and bound algorithm

The starting point of the algorithm (the root of solutions tree H) is a permutation π_0, and the set of free jobs $S_{\pi_0} = N$. Let us assume $\pi^* \leftarrow \pi_0$ as the best solution and let upper bound $UB=F(\pi^*)$. The tree level is $h = 0$.

Let π be a permutation (node) on h-th level of the tree H. The set of free jobs $S_\pi = |k|$, $k = n\text{-}h$.

STEP 1: {*Lower bound*}
If lower bound $LB(\pi) \geq F(\pi^*)$ **then** go to STEP 4

STEP 2: {*Upper bound*}
If $F(\pi) < UB$ **then** $UB \leftarrow F(\pi)$, $\pi^* \leftarrow \pi$;

STEP 3: {*Calculations*}
If $K(\pi) = \emptyset$ **then** go to STEP 4
Select a job $l \in K(\pi)$, such that:

$$\Delta(l) = \min_{i \in K(\pi)} \{\Delta_i\},$$

Generate new permutation β (node in H) by selecting the job l on the k-th position. Let $h \leftarrow h+1$; $k \leftarrow n-h$; $\pi \leftarrow \beta$;
Go to STEP 1.

STEP 4: {Backtrack}
If π is the root of the tree (π^* is an optimal solution) **then EXIT.**
If permutation π was generated from β by fixing free job $l \in K(\pi)$ **then:**
$h \leftarrow h-1$; $k \leftarrow n-h$; $K(\pi) \leftarrow K(\pi) \setminus \{l\}$, **goto STEP 3.**

■

The quality of solutions calculated by branch and bound algorithm depends also on starting point. Below we present heuristic algorithm which calculates this solution.

Algorithm AH (heuristic algorithm).

Enumerate jobs, such as $d_1 \leq d_2 \ldots \leq d_n$. Let t to a current time.
Set $W \leftarrow \emptyset$; $t \leftarrow 0$;
for $i := 1$ **to** n **do**
begin

if $t+p_i \le d_i$ **then**
Set: $W \leftarrow W \cup \{i\}$, $t \leftarrow t + p_i$;
else
Create set: $Q = \{l \in W : \sum_{j \in S} p_j - p_l + p_i \le d_i\}$;
if $Q \neq \varnothing$ **then**
Create set: $V = \{j \in Q : w_j > w_i\}$;
if $V \neq \varnothing$ **then**
Set $k \in V$ such as $w_k = \max\{w_l : l \in V\}$ and
set $W \leftarrow W \setminus \{k\} \cup \{i\}$; $t \leftarrow t - p_k + p_i$;
end;

Execute jobs from the set W (in order of appearance), next execute the remaining jobs in any order.

The AH algorithm requires $O(n\log n)$ time.

7. Algorithm Parallelization

Parallel algorithm was implemented for SIMD model of parallel processors without shared memory. Each processor has its own local memory with short time of access; Communication between processors is very slow (comparing to local-memory access).

The main idea of parallel algorithm is to make concurrent search process on solution's tree *H*. Each processor has set of vertex to search and local value of upper bound *UB*. If every processor had the newest value of the best upper bound in every moment, the speedup (comparing to sequential algorithm) would be the greatest. But broadcasting of upper bound costs – the time of communication between processors is very long. That's why frequency of communication between processors (broadcasting of the newest value of upper bound) have to be low. In our implementation the processor is getting a new value of *UB* when it wants to broadcast its own π^* and (independently) after some period of time (or number of iterations *Broadcast_iter*).

Comparison results of some *Broadcast_iter* for parallel algorithm can be seen in Table 2.

Scheme of parallel algorithm

for each processor
begin
Heap : heap; {local for each processor}
while *Heap* <> NULL
begin
π := Get(*Heap*);

{Upper Bound}
if $F(\pi) < UB$ then
begin
$UB \leftarrow F(\pi)$, $\pi^* \leftarrow \pi$;
broadcast π^* to other processors;
end;
Select a job $l \in K(\pi)$, such that $\Delta(l) = \min_{l \in K(\pi)} \{\Delta_i\}$,
Generate new permutation β (node in H) by selecting the job l on the k-th position. Let $h \leftarrow h+1$; $k \leftarrow n-h$;.
Put(*Heap*, β);
end;
end.

8. Computer simulations

Test problems were generated as follows [7]. For each job i, an integer processing time p_i was generated from the uniform distribution [1, 100] and, for weighted tardiness problems, an integer weight was generated from the uniform distribution [1, 10]. Problem hardness is likely to depend on the relative range of due dates (RDD) and on the average tardiness factor (TF). Having computed $P = \sum_{i=1}^{n} p_i$ and selected values of RDD and TF from the set {0.2, 0.4, 0.6, 0.8, 1.0}, an integer due date d_i from the uniform distribution [P(1 - TF – RDD/2), P(1 – TF + RDD/2)] was generated for each job i. Five problems were generate for each of the 25 pairs of values of RDD and TF, yielding 100 problems for each value of n.

Table 1

Number of iterations

	number of processors		
n	1	2	4
20	**1542**	2287	2930
25	**5654**	6052	7468
30	22985	18006	**15913**
35	91587	**66321**	79854
40	293122	226892	**224632**

The algorithm was implemented in Ada95 language and run on Sun Enterprise 4x400MHz computer under Solaris 7 operating system. Tasks of Ada95 language were executed in parallel as system threads.

There are average results of iteration number of algorithm for each number of processors in Table 1. As we can see parallel algorithm makes less number of iterations than

the sequential algorithm for large problem size (n>25). In parallel algorithm the number of iterations is counted as the sum of iterations for every processor – so the speedup we can get, may be greater than P, where P is number of processors.

Table 2

Comparing of number of iterations and execution time for one random problem instance (n=12, RDD=TF=1.0, 4 and 1 processors)

4 – processors implementation		
Broadcast_iter	iterations	time (sec.)
1	208118	37
10	200434	14
100	205357	14
500	206889	9
1000	196128	9
5000	207699	9
10 000	205983	11
20 000	206750	11

1 – processor implementation		
-	247046	29

As we have said before, the frequency of communication between processors (broadcasting of the newest value of upper bound) have to be low. The best value for our computer was 1000 iteration, but *Broadcast_iter* value strongly depends on machine and communication speed.

9. Conclusions

This paper gives a practical sequential and parallel branch and bound algorithm for the total weighted late job problem. Preliminary calculation let us suppose that increase number of processors cause to make possible to solve larger instances (more than 50 jobs), especially when we will use stronger lower bounds and additional elimination criteria.

LITERATURA

1. Elmaghraby S.E.: The One-Machine Sequencing Problem with Delay Costa, Journal of Industrial Engineering, 19(1968) pp.105-108.
2. Emmons H.: One-Machine Sequencing to Minimize Certain Functions of Job Tardiness, Operations Research, 17(1969), pp. 701-705.

3. Karp R.M.: Reducibility among Combinatorial Problems, Complexity of Computations, R.E. Millerand J.W. Thatcher (Eds.), Plenum Press, New York, 1972, pp.85-103.
4. Lawler E.L., Moore J.M.: A Functional Equation and its Applications to Resource Allocation and Sequencing Problems, Management Sci., 16(1969), pp.77-84.
5. Potts C.N., Van Wassenhove L.N.: A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, Operations Research, 33(1985), pp.177-181.
6. Potts C.N., Van Wassenhove L.N.: Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs, Management Science, vol. 34, No.7, 1988, pp.843-858.
7. Rinnoy Kan A.H.G., B.J. Lageweg, Lenstra J.K.: Minimizing total costs in one-machine scheduling, Operations Research, 25(1975) pp.908-927.
8. Sahni S.K.: Algorithms for Scheduling Independent Jobs, J.Assoc. Comput. Match., 23(1976), pp.116-127.
9. Shwimer J.: On the n-job, one-machine sequencing-independent scheduling problem with tardiness penalties: A branch and bound solution, Management Sci., 18 B(1972) pp.301-313.
10. Villareal F.J., Bulfin R.L.: Scheduling a Single Machine to Minimize the Weighted Number of Tardy Jobs, IE Trans., 15(1983), pp.337-343.

Recenzent: Prof. dr hab. inż. Tadeusz Sawik

Streszczenie

W pracy rozpatrujemy problemu kolejnościowy polegający na uszeregowaniu n zadań na jednej maszynie. Maszyna ta, w dowolnej chwili, może wykonywać co najwyżej jedno zadanie. Dla zadania i niech p_i, w_i, d_i będą odpowiednio: *czasem wykonania, wagą funkcji kosztów* oraz *wymaganym terminem zakończenia* (*linią krytyczną*). Jeśli ustalona jest kolejność wykonywania zadań i C_i jest *terminem wykonania* zadania i ($i=1,2,...,n$), to $U_i=1$, gdy $C_i>d_i$, a 0 w przeciwnym przypadku nazywa się *spóźnieniem*, natomiast w_iU_i jest *kosztem* spóźnienia zadania. Rozważany problem polega na wyznaczeniu takiej kolejności wykonywania zadań, która zminimalizuje sumę kosztów spóźnień $\sum_{i=1}^{n} w_iU_i$. Należy on do klasy problemów silnie NP-zupełnych. Przedstawiamy algorytm równoległy (dla modelu obliczeń SIAD) oparty na metodzie podziału i ograniczeń, w którym wykorzystano kryteria eliminacyjne. Pozwalają one na znaczne zmniejszenie zbioru rozwiązań dopuszczalnych. Przy większej liczbie procesorów, w rozsądnym czasie, można tym algorytmem rozwiązywać średnich rozmiarów przykłady.