Discrete Applied Mathematics 3 (1981) 313–318
North-Holland Publishing Company

NOTE

FAST APPROXIMATION ALGORITHM FOR JOB SEQUENCING WITH DEADLINES

G.V. GENS and E.V. LEVNER

Central Economic and Mathematical Institute, Moscow, USSR

Received 11 March 1980
Revised 27 November 1980

The problem under consideration is to schedule jobs on a machine in order to minimize the sum of the penalties of delayed jobs. A "range-and-bound" method is proposed for finding a tight bound $\tilde{P}$ such that $\tilde{P} \le P^* \le 2\tilde{P}$, P^* being the minimal sum desired. The considered scheduling problem, for n jobs and accuracy $\varepsilon > 0$, is solved by a fully polynomial ε-approximation algorithm in $O(n^2 \log n + n^2/\varepsilon)$ time and $O(n^2/\varepsilon)$ space.

1. Introduction

The minimization version of the well-known job sequencing with deadlines problem, MIN-JSD, is as follows. We are given n independent jobs $J_1, J_2, \ldots, J_n$, to be processed on one machine. Associated with each job, J_i, is its processing time, t_i, and deadline, D_i. If the processing of job J_i is not completed by its deadline, D_i, then a penalty p_i is payed. The problem is to find a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of $\{1, 2, \ldots, n\}$, that is, to schedule jobs in such an order as to minimize the total penalty

$$P(\pi) = \sum_{i=1}^{n} p_{\pi(i)} x_{\pi(i)}$$

where $x_{\pi(i)} = [\text{if } t_{\pi(1)} + t_{\pi(2)} + \cdots + t_{\pi(i)} > D_{\pi(i)} \text{ then 1 else 0}]$.

Lawler and Moore [7] have proposed a pseudo-polynomial method to treat it, but the problem is NP-hard (Karp [6]). Sahni [8] has presented an $O(n^2/\varepsilon)$ fully polynomial ε-approximation algorithm for the maximization version of the problem considered. However, the problem MIN-JSD can not be solved through trivial generalization of Sahni's method, since this method employs essentially a tight bound $\tilde{P}$ such that $\tilde{P} \le P^* \le c\tilde{P}$, P^* being the minimal sum desired and c being a constant or a polynomial in n; in the case of MIN-JSD it is a nontrivial question to produce an adequate tight bound $\tilde{P}$.

In a recent paper [2] the authors have built an $O(n^3/\varepsilon)$ fully polynomial algorithm for MIN-JSD, based on the ε-grouping technique introduced by Babat [1], in which there is no need to produce a tight bound $\tilde{P}$.

0166-218X/81/0000–0000/$02.75

In this paper we present a special "range-and-bound" method for finding a tight bound $\tilde{P}$ in MIN-JSD such that $\tilde{P} \le P^* \le 2\tilde{P}$. The $\tilde{P}$ value being found, we apply techniques introduced by Ibarra and Kim [5] and Sahni [8] to build a fully polynomial ε-approximation algorithm solving MIN-JSD in $O(n^2 \log n + n^2/\varepsilon)$ time and $O(n^2/\varepsilon)$ space.

Following the reasoning of Lawler and Moore [7], we can use an integer programming formulation of the MIN-JSD problem, limiting our search to schedules such that:

– jobs which are to be on time, are processed in increasing order of their deadlines, with the tardy jobs following them in arbitrary order;

– there is no idle time between jobs.

Clearly, such schedules contain the optimal one [7].

Order all the jobs by their deadlines, earliest deadline first and let $x_i = [$if the job J_i is to be tardy then 1 else 0$]$. Then problem MIN-JSD can be written as follows:

$$\text{minimize} \quad P(x) = \sum_{i=1}^{n} p_i x_i,$$

$$\text{subject to} \quad \sum_{i=1}^{j} t_i (1 - x_i) \le D_j, \quad 1 \le j \le n,$$

$$x_i = 0 \text{ or } 1, \quad 1 \le i \le n$$

The problem can be rewritten in the following standard form:

$$\text{minimize} \quad P(x) = \sum_{i=1}^{n} p_i x_i,$$

$$\text{subject to} \quad \sum_{i=1}^{j} t_i x_i \ge B_j, \quad 1 \le j \le n,$$

$$x_i = 0 \text{ or } 1, \quad 1 \le i \le n,$$

$$\text{where} \quad B_j = \sum_{i=1}^{j} t_i - D_j, \quad 1 \le j \le n.$$

2. Ranging algorithm for the MIN-JSD problem

The "range-and-bound" method suggested consists of a ranging subroutine $R(p, \varepsilon)$ and a bounding subroutine.

Given a problem instance I and positive numbers p and ε, the ranging subroutine, $R(p, \varepsilon)$, either reports that the minimal penalty, P^*, is $\le p$, or else reports that $P^* > p(1 - \varepsilon)$. The bounding subroutine first finds a bound P^0 such that $P^0 \le P^* < nP^0$, and then uses the ranging subroutine $R(p, \varepsilon)$, with $\varepsilon = 0{,}25$ and p taken successively to $\frac{1}{2}nP^0$, $\frac{1}{4}nP^0, \ldots$, until we find the bound $\tilde{P}$ desired.

Some versions of ranging procedures for the knapsack problems were suggested

in [3, 4]. We present in this Section another ranging algorithm incorporating the dynamic programming scheme similar to that of the algorithm FRAME in [8].

We begin with the following observations:

(i) Associated with each vector $x=(x_1,\ldots,x_n)$ may be a pair (P, T), where $P=\sum_{i=1}^{n} p_i x_i$ and $T=\sum_{i=1}^{n} t_i x_i$. We shall deal with sets $S^{(i)}$ of pairs (P, T) ordered in an increasing order of P, so that every pair (P, T) in $S^{(i)}$ corresponds to a schedule of the jobs $1,\ldots,i$. In order to construct an optimal solution, i.e., to determine a vector x^* corresponding to the optimal pair (P^*, T^*), we shall use a standard "backtracing" technique [8].

(ii) If we have two pairs, (P_1, T_1) and (P_2, T_2), with $T_1 \geq T_2$ and $P_1 \leq P_2$, then the pair (P_2, T_2) is called "dominated" and may be discarded.

(iii) If we are interested only in solutions satisfying $P(x) \leq p$ (p being a parameter), then a pair (P, T) with $P>p$ (called "p-redundant") may be discarded.

(iv) If we have two pairs, (P_1, T_1) and (P_2, T_2), such that $0 \leq P_2 - P_1 \leq \delta$, then the pairs are called "δ-close". To discard δ-close pairs from a set S means the following:

(a) partition the interval $[1, p]$ (where p is a parameter given) into $\lceil n/\varepsilon \rceil$ equal subintervals of size no greater than $\delta=\varepsilon p/n$,

(b) if more than one pair from S falls into any one of these subintervals, then discard all such δ-close pairs, except for the only "representative" in each subinterval, namely, the pair with the largest (in this subinterval) T coordinate.

We can now describe the steps of the ranging algorithm.

The ranging algorithm $R(p, \varepsilon)$

Input. $(p_i,\ t_i,\ D_i)$, $1 \leq i \leq n$, such that $D_1 \leq D_2 \leq \cdots \leq D_n$; $\varepsilon>0$, $p>0$, $\delta=\varepsilon p/n$, $B_j=\sum_{i=1}^{j} t_i - D_j$, $1 \leq j \leq n$.

Output. Either the report that the minimal penalty, P^*, is $\leq p$, or else the report that $P^*>p(1-\varepsilon)$.

Step 1. [Initialize]. $S^0 \leftarrow \{(0,0)\}$, $W^0 \leftarrow \emptyset$.

Step 2. [Generate $S^{(1)}, S^{(2)}, \ldots, S^{(n)}$].

For $i=1$ to n

Do $V^{(i)} \leftarrow \emptyset$

For each pair (P, T) in $S^{(i-1)}$

Do If $T+t_i \geq B_i$, then $V^{(i)} \leftarrow V^{(i)} \cup \{(P+p_i, T+t_i)\}$

End

Form $W^{(i-1)}$, the set of pairs (P, T) from $S^{(i-1)}$ such that $T \geq B_i$. Merge $W^{(i-1)}$ and $V^{(i)}$ to obtain $S^{(i)}$, during the merge eliminate δ-close and p-redundant pairs. If $S^{(i)}$ is empty, go to Step 3.

End

Step 3. If any one of $S^{(1)}, \ldots, S^{(n)}$ is empty, then report that $P^*>p(1-\varepsilon)$, otherwise report that $P^* \leq p$.

Let us now prove that if any one of the sets $S^{(1)}, \ldots, S^{(n)}$ in the algorithm $R(p, \varepsilon)$ is empty, then the optimum value, P^*, is $>p(1-\varepsilon)$, otherwise $P^* \leq p$.

If a pair (P, T) is omitted at some point in the execution of the algorithm, then either $P>p$, or else at the end of the ith iteration we shall have in $W^{(i-1)} \cup V^{(i)}$ its "representative", (P', T'), such that $P \geq P' - i\varepsilon p/n$. If $P' \leq p$, the pair (P', T') is placed into $S^{(i)}$, otherwise it is discarded as p-redundant.

If a $S^{(i)}$ is empty, this means that, for any pair (P, T), either $P>p$, or else its representative at the ith iteration, (P', T'), had $P'>p$. So, for any feasible solution, we have

$$P \geq P' - i\varepsilon p/n > p(1-\varepsilon),$$

and, hence, the minimum, P^*, is $>p(1-\varepsilon)$.

If $S^{(n)}$ is non-empty: a $(P_j, T_j) \in S^n$, then (P_j, T_j) corresponds to a feasible solution of MIN-JSD, and $P_j \leq p$ (as p-redundant pairs are discarded). So, $P^* \leq P_j \leq p$. □

Complexity analysis of $R(p, \varepsilon)$. Since $|S^{(k)}| \leq \lceil n/\varepsilon \rceil$, the time and space required to generate $S^{(n)}$ is $O(\sum_{k=1}^{n} |S^{(k)}|) = O(n^2/\varepsilon)$. Given $\varepsilon = 0.25$, the algorithm $R(p, \varepsilon)$ in $O(n^2)$ time and space will either report that $P^* \leq p$, or else report that $P^* > \frac{3}{4}p$.

3. "Range-and-Bound" algorithm

Before considering the "range-and-bound" algorithm, we find a preliminary bound P^0 for MIN-JSD satisfying $P^0 \leq P^* \leq nP^0$.

Initialize every x_i to 0. Order (temperarily) all the jobs according to non-decreasing penalties: $p_{i_1} \leq p_{i_2} \leq \cdots \leq p_{i_n}$. Then set $x_{i_1} = 1$, $x_{i_2} = 1$, $x_{i_3} = 1, \ldots$ in that order until all the n constraints of the problem MIN-JSD are satisfied. Let k^* be the smallest number of such x_i's.

Clearly, p_I, where $I = i_{k^*}$, is the optimal solution value of the following problem:

$$\text{minimize} \quad \max_{1 \leq i \leq n} p_i x_i,$$

$$\text{subject to} \quad \sum_{i=1}^{j} t_i x_i \geq B_j, \quad 1 \leq j \leq n;$$

$$x_i = 0 \text{ or } 1, \quad 1 \leq i \leq n.$$

It is evident that $p_I \leq P^* \leq k^* p_I \leq np_I$, and we can take the p_I value as the P^0 value desired. The running time needed to find the P^0 value is clearly $O(n^2)$.

We are now ready to present our "range-and-bound" algorithm, R&B-MIN-JSD.

Algorithm R&B-MIN-JSD

Input. (p_i, T_i, D_i), $1 \leq i \leq n$, such that $D_1 \leq D_2 \leq \cdots \leq D_n$; a bound P^0 satisfying $P^0 \leq P^* \leq nP^0$.

Output. The tight bound, $\tilde{P}$, such that $\tilde{P} \leq P^* \leq 2\tilde{P}$.

Step 1. [Initialize]. Set $p \leftarrow \frac{1}{2}nP^0$.

Step 2. [Binary search]. Use the ranging algorithm $R(p, \varepsilon)$ with $\varepsilon = 0.25$.
If the algorithm $R(p, 0.25)$ reports that $P^* \leq p$, then set $p \leftarrow \frac{1}{2}p$ and go to Step 2.
If the algorithm $R(p, 0.25)$ reports that $P^* > \frac{3}{4}p$, then set $p \leftarrow \frac{3}{2}p$.

Step 3. [Find $\tilde{P}$]. Use the ranging algorithm $R(p, \varepsilon)$ with $\varepsilon = 0.25$.
If the algorithm $R(p, 0.25)$ reports that $P^* \leq p$, then set $\tilde{P} \leftarrow \frac{1}{2}p$ and Stop.
If the algorithm $R(p, 0.25)$ reports that $P^* > \frac{3}{4}p$, then set $\tilde{P} \leftarrow \frac{3}{4}p$ and Stop.

As far as $P^0 \leq P^* \leq nP^0$, and the execution of Step 2 either halves this interval, or leads to Stop, Step 2 can be executed at most $\log_2 n$ times, each time with a possible execution of the algorithm $R(p, 0.25)$. Hence, the total time of the algorithm R&B-MIN-JSD is $O(n^2 \log n)$ and space $O(n^2)$.

4. Approximation algorithm for the MIN-JSD problem

We can now explain our interest of the calculation of $\tilde{P}$: the existence of a tight bound $\tilde{P}$ enables us to apply fully polynomial approximation schemes suggested by Ibarra and Kim [5] and Sahni [8], and to reduce the time bound $O(n^3/\varepsilon)$ obtained in [2] to $O(n^2 \log n + n^2/\varepsilon)$.

In this paper we dwell on the Sahni approach, introducing a minor refinement to the "partitioning" algorithm presented in [8], in order to consider a case of minimization. The modification proposed is the use of a revised rule to discard "unnecessary" pairs in the basic dynamic programming procedure.

At the end of the $(i-1)$-th iteration a list $S^{(i-1)}$ of combinations of penalty and time, (P, T), is formed. To perform iteration i we produce a candidate pair $(P + p_i, T + t_i)$ for each pair $(P, T) \in S^{(i-1)}$ provided $T + t_{i-1} \geq B_i$. The revised rule is to merge the candidate pairs only with those pairs $(P, T) \in S^{(i-1)}$ which have $T \geq B_i$. As for the "dominated" and "δ-close" pairs ($\delta = \varepsilon\tilde{P}/n$), they are discarded in the usual way (see [8]).

The partitioning algorithm mentioned has time and space complexity $O(n^2/\varepsilon)$ [8], and the $\tilde{P}$ computation complexity is $O(n^2 \log n)$ time and $O(n^2)$ space.

A concluding remark. It is certainly possible that the time and space complexity presented here can be improved upon. An open question is: Can an ε-approximation algorithm be found for the MIN-JSD problem which is of the same complexity as the algorithms for the maximization form of the JSD problem? Or, more interesting, is it possible to establish that the MIN-JSD problem is inherently more complex then MAX-JSD?

Acknowledgment

We would like to thank the referees for their detailed comments which improved significantly the presentation of this paper.

References

[1] L.G. Babat, Linear functions on the N-dimensional unit cube, Soviet Math. Dokl. 222 (1975) 761–762.

[2] G.V. Gens and E.V. Levner, Approximation algorithms for some scheduling problems, Engeneering Cybernetics 6 (1978) 38–46.

[3] G.V. Gens and E.V. Levner, Computational complexity of approximation algorithms for combinatorial problems, Mathematical Foundations of Computer Science, Proc. 8th Symp., Olomouc/Czech. 1979, Lect. Notes Comput. Sci., Vol. 74 (Springer-Verlag, Berlin, 1979) 292–300.

[4] G.V. Gens and E.V. Levner, Fast approximation algorithms for knapsack type problem, Proc. IX IFIP Confer. on Optimization Techniques, Warsaw, 1979, Lect. Notes Control and Information Sciences, Vol. 23 (Springer-Verlag, Berlin, 1980).

[5] O.H. Ibarra and C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems. J. Assoc. Comput. Mach. 22 (1975) 463–468.

[6] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and I.W. Thatcher, eds., Complexity of Computer Computations (Plenum Press, New York, 1972) 85–104.

[7] E.L. Lawler and J.M. Moore, A functional equation and its application to resource alocation and sequencing problems, Management Sci. 16 (1) (1969) 77–84.

[8] S. Sahni, Algorithms for scheduling independent tasks, J. Assoc. Comput. Mach. 23 (1) (1976) 116–127.