African Journal of Mathematics and Computer Science Research Vol. 3(12), pp. 307-314, December 2010
Available online at http://www.academicjournals.org/AJMCSR
ISSN 2006-9731 ©2010 Academic Journals

Review

A metaheuristic and PTAS approach for NP-hard scheduling problem with controllable processing times

Hamed Bayat[1]*, Hassan Akbari[1] and Hamid Davoudpour[2]

[1]Islamic Azad University, Naragh Branch, Naragh, Iran
[2]Amirkabir University of Technology, Tehran, Iran.

Accepted 3 November, 2010

Most of the applied and interesting problems in industry and real world are difficult to solve. These problems are often NP-hard and the problem in this paper is strongly NP-hard. Approximation and metaheuristic algorithms are used to find a solution to these problems. In this paper we have used a polynomial time approximation algorithm, this algorithm is a suboptimal approach that provably works fast and that provably yields solutions of very high quality. In this paper, the problem of scheduling jobs on a single machine with controllable processing times is considered. The fact that the n jobs have controllable processing times means that it is possible to reduce the processing time of the jobs by paying a certain cost. In this paper, each job has a release date when it becomes available for processing, and, after completing its processing, requires an additional delivery time. Furthermore, preemption is allowed. The preemptive version allows an operation to be interrupted and continued at a later time. Feasible schedules are further restricted by job precedence constraints. The algorithm in this paper gives a substantial improvement for the special case without controllable processing times obtained by Hall et al. (1989) and the special case with controllable processing times by Mastrolilli (2009). In this paper we added controllable processing time instead fixed processing time and preemption to the problem. Moreover, we develop a polynomial time approximation scheme whose running time depends only linearly on the input size.This improves and generalize the previous ($3/2+\varepsilon$)-approximation algorithm by Mastrolilli (2009). At last it will be shown that the problem with its constraints has a polynomial time approximation scheme. It means that for any given ε, a polynomial algorithm exists for the problem. It will be shown by a numerical example finally.

Key words: Scheduling, controllable processing times, polynomial time approximation scheme (PTAS), metaheuristic methods, computational complexity, intractable problems.

INTRODUCTION

For single machine scheduling problems, it is generally assumed that release times and processing times of jobs are known and constant. However, in many cases, the release times and the processing times can be made earlier and shorter by using additional resources such as manpower, fuels, raw materials and so on. Many researchers have focused on problems with controllable processing times and problems with resource dependent release times, independently.

In this paper we consider the following single machine scheduling problem. A set, $J_1, J_2, ..., J_n$, of n jobs is to be processed without interruption on a single machine. For each job J_j there is an interval $[l_j, u_j], 0 \le l_j \le u_j$, specifying its possible processing times. The cost for processing job J_j in time l_j is $c_j^l \ge 0$, and for processing it in time u_j the cost is $c_j^u \ge 0$. For any value $\delta_j \in [0,1]$ the cost for processing job J_j in time $p_j(\delta_j) = \delta_j l_j + (1-\delta_j)u_j$ is $c_j(\delta_j) = \delta_j c_j^l + (1-\delta_j)c_j^u$, where δ_j is the compression parameter. Additionally,

*Corresponding author. E-mail: habayat@yahoo.com.

each job J_j has a release date $r_j \geq 0$, when it first becomes available for processing and, after completing its processing on the machine, requires an additional delivery time $q_j \geq 0$; if $s_j \left(\geq r_j \right)$ denotes the time J_j starts processing, then it has been delivered at time $s_j + p_j(\delta_j) + q_j$, for compression parameter δ_j. Delivery is a non-bottleneck activity, in that all jobs may be simultaneously delivered. Feasible schedules are further restricted by job precedence constraints given by the partial order $\prec$, where $J_j \prec J_k, (j,k \in \{1,2,...,n\})$ means that job J_k must be processed after job J_j. Let θ be a permutation of the set *J* that is consistent with the precedence constraints; θ denotes a processing order of jobs. Denote by $T(\delta,\theta)$ the (earliest) maximum delivery time of all the jobs for compression parameters $\delta = (\delta_1, \delta_2, ..., \delta_n)$ and processing order θ. The total cost of compression parameters δ is equal to $\sum_{j \in J} c_j(\delta_j)$, and the total scheduling cost for compression parameters δ and processing order θ is defined as

$$L(\delta,\theta) = T(\delta,\theta) + \sum_{j \in J} c_j(\delta_j)$$

The problem is to find θ and δ that minimizes $L(\delta,\theta)$. Since the special case with fixed processing times and without precedence constraints is strongly NP-hard (Lenstra et al., 1977) the stated problem is also strongly NP-hard. When all processing times are fixed $(\forall j \in J)$; $u_j = l_j$, the problem as stated is equivalent to that with release dates and due dates, d_j, rather than delivery times, in which case the objective is to minimize the maximum lateness, $L_j = s_j + p_j - d_j$, of any job J_j. When considering the performance of approximation algorithms, the delivery time model, which assumes $d_j \leq 0$, is preferable (Hall and Shmoys, 1989; Hall and Shmoys, 1992). Without this restriction, results are likely to be elusive, since the problem of determining whether $L_{\max} \leq 0$ is NP-complete. Because of this equivalence, we shall denote the problem with fixed processing times as $1|r_j, prec|L_{\max}$ (and $1|r_j|L_{\max}$ when there are no precedence constraints), using the notation of Graham et al. (1979). As these scheduling problems are known to be hard to solve optimally, most research focuses on giving polynomial-time approximation algorithms that produce a solution close to the optimal one. Ideally, one hopes to obtain a family of polynomial algorithms such that for any given $\varepsilon > 0$ the corresponding algorithm guaranteed to produce a solution with a value within a factor of $(1+\varepsilon)$ of the optimum value; such a family is called a polynomial time approximation scheme (PTAS).

Hall et al. (1990) proposed two polynomial time approximation schemes for problem $1|r_j|L_{\max}$, the running time of which are $O\left(n \log n + n(1/\varepsilon)^{O(1/\varepsilon^2)}\right)$ and $O\left((n/\varepsilon)^{O(1/\varepsilon)}\right)$. For the corresponding problem with controllable processing times, Zdrzalka (1991) gives a polynomial time approximation algorithm with a worst-case ratio of $3/2 + \varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small. When the precedence constraints are imposed and the job processing times are fixed ($1|r_j, prec|L_{\max}$), Hall et al. (1990) give a PTAS. This consists of executing, for $\log_2 \Delta$ times, an extended version of their previous PTAS for $1|r_j, prec|L_{\max}$, where Δ denotes an upper bound on the optimal value of any given instance whose data are assumed to be integral. This polynomial running time should be contrasted with the time complexity of their result for problem $1|r_j|L_{\max}$, where they were able to achieve a considerably better time. To some extent, this is not surprising, since precedence constraints add a substantial degree of difficulty, and one important area of research in scheduling theory has been to study under what conditions a precedence-constrained problem is computationally harder than its counterpart with independent jobs.

In this paper we generalized the first known PTAS for problem $1|r_j, prec|L_{\max}$ with controllable processing times that runs in linear time (the hidden constant depends exponentially on $1/\varepsilon$) given by Mastrolilli (2006) by using a new approximation algorithm for knapsack problem. We use this to improve and generalize all the previous results (Shmoys, 1989; Hall and Shmoys, 1990; Hall and Shmoys, 1992; Zdrzalka, 1991; Mastrolilli, 2009). The linear complexity bound is a substantial improvement compared to the above mentioned result. Moreover, the existence of a PTAS whose running time is also polynomial in $1/\varepsilon$ for a strongly NP-hard problem would imply P=NP (Garey and Johnson, 1979).

The "0-1" knapsack problem is as follows:

Given n pairs of positive integers, (p_j, a_j) and a positive integer b, find $x_1, x_2, \ldots, x_n$ so as to

$$\max \quad P = \sum_j p_j x_j$$

$$s.t \quad A = \sum a_j x_j \le b, \qquad x_j \in \{0,1\}.$$

We may think of j as indexing items, with associated profits p_j and weights a_j. The objective is to find the most portifable possible selection of items wich can be made to fit into a knapsack with capacity *b*. One variation of the problem permits items to be chosen with repetition. That is, x_j is permitted to be any nonnegative integer. This is sometimes called the "unbounded" knapsack problem.

To obtain a new linear complexity bound we use the method for simplifying the input. Rounding the input is a widely used technique to obtain polynomial time approximation schemes. Arithmetic or geometric rounding is the most successfully and broadly used way of rounding to obtain a simpler instance that may be solved in polynomial time. It is well known that KP is NP-hard but pseudopolynomially solvable through dynamic programming, and the same properties hold for kKP. Basically, the developed approximation approaches for KP and kKP can be divided into two groups:

1) Approximation Algorithms*:* for KP the classical $\frac{1}{2}$ - approximation algorithm [] needs only $O(n)$ running time. An approximation ratio of $\frac{1}{2}$ can be obtained also for kKP by rounding the dolution of the linear programming relaxation of the problem (Capara et al., 2000) this algorithm can be implemented to run in linear time when the LP relaxation of kKP is solved by using the method by Megiddo et al. (2000).

2) Polynomial time approximation scheme (PTAS): PTAS reach any given approximation ratio and have a running time polynomial in the length of the encoded input. Caprara et al.(2000) gave an approximation ratio of $(1+\varepsilon)$ within $O\left(n^{[1/\varepsilon]-2} + n\log n\right)$ and $O\left(n^{[1/\varepsilon]-1}\right)$ running time, for KP and kKP, respectively. The best schemes currently known requiring linear space are given in Mastrolilli and Hutter (2006); they present a PTAS for KP and kKP requiring linear space a running time $O\left(n(\log 1/\varepsilon)^{O(1/\varepsilon)}\right)$ $O\left(n + k(\log 1/\varepsilon)^{O(1/\varepsilon)}\right)$ respectively.

In this paper we first obtain a multiple-choice knapsack problem for the best schemes currently known and at last we improve a linear complexity bound for the assumed scheduling problem.

MULTIPLE-CHOICE PROBLEMS

Suppose the n items are partitioned into m equivalence classes and it is stipulated that no more than one item (or multiples of one item) may be chosen from each equivalence class. Such a problem is sometimes called a multiple-choice knapsack problem.

In this paper we must relax the restrictions $x_j \in \{0,1\},\ j = 1,2,\ldots,n,$ to $0 \le x_j \le 1$. If $S_i, i = 1,2,\ldots,m,$ denotes the set of indices of items in the *i*th equivalence class, then we obtain a linear programming problem of the form

$$\max \quad \sum_j p_j x_j$$

$$s.t \quad \sum_j a_j x_j \le b,$$

$$\sum_{j \in S_i} x_j \le 1, \qquad i = 1,2,\ldots,m,$$

$$0 \le x_j \le 1, \qquad j = 1,2,\ldots,n.$$

For any feasible solution to this linear programming problem, the profit-weight contribution of the items in the *i*th equivalence class corresponds to the point in the convex hull of the set of points $\{(a_j, p_j) \mid j \in S_i\} \cup \{0,0\}$. Moreover, by dominance relaxations, this point can be assumed to lie on the “upper boundery" of the convex hull.

Let the n_i corner points of the upper boundary for equivalence class *i* be designated $(a_{j(1)}, p_{j(1)}), (a_{j(2)}, p_{j(2)}), \ldots, (a_{j(n_i)}, p_{j(n_i)})$. These points can be identified first by sorting the items into nondecreasing order of ratios p_j / a_j and then selecting out the desired items by observing that

$$\frac{p_j(k) - p_j(k-1)}{a_j(k) - a_j(k-1)} > \frac{p_j(k+1) - p_j(k)}{a_j(k+1) - a_j(k)} > 0, \qquad k = 1,2,\ldots,n_i - 1,$$

where $a_{j(0)} = p_{j(0)} = 0.$ All corner points for all equivalence classes can be identified in $O(n\log n)$ time.

Now, for $k = 1,2,\ldots,n_i - 1,$ let

$$\overline{a}_{j(k)} = a_{j(k)} - a_{j(k-1)},$$

$$\overline{p}_{j(k)} = p_{j(k)} - p_{j(k-1)}.$$

With these new coefficients, we obtain a linear programming problem of the form

$$\max \quad \sum_j \bar{p}_j \bar{x}_j$$

$$s.t \quad \sum_j \bar{a}_j \bar{x}_j \le b,$$

$$0 \le \bar{x}_j \le 1.$$

We assert that the optimal value of the objective function for this new problem is equal to that of the previous one.

Now we solve this new linear programming problem, as follows. Firstly, sort the items in nonincreasing p_j / a_j ratio, so that, without loss of generality,

$$\frac{p_1}{a_1} \ge \frac{p_2}{a_2} \ge \ldots \ge \frac{p_n}{a_n}$$

Then place the items in the knapsack in order until either (a) the items are exhausted or (b) the capacity is exactly used up or (c) it is necessary to fractionalize one item to use up the capacity exactly. In case (a) and (b), an optimal solution is obtained, and it is unnecessary to proceed further.

So suppose case (c) occurs and

$$a_1 + a_2 + \ldots + a_j < b$$

But

$$a_1 + a_2 + \ldots + a_j + a_{j+1} > b$$

We assert that

$$P_0 \le P^* \le 2P_0,$$

Where

$$P_0 = \max\{p_1 + p_2 + \ldots + p_j, P_{\max}\},$$

and where

$$P_{\max} = \max_j \{p_j\},$$

as before. This is because

$$p_1 + p_2 + \ldots + p_j \le P^*, \qquad p_{j+1} \le p_{\max} \le P^*,$$

But

$$p_1 + p_2 + \ldots + p_{j+1} > P^*.$$

Replacing $p_{\max}$ by P_0 in (3.2), we obtain

$$K = \varepsilon P_0 / n,$$

and find that

$$\frac{P^*}{K} \le \frac{2n}{\varepsilon}.$$

The compution can now be carried out in $O\left(n^2 / \varepsilon\right)$ time and space, exclusive of the time required to sort the items in p_j / a_j order.

Now it is not hard to see that this optimal solution has the property that, for each S_i, $\bar{x}_{j(k)} = 1$ implies $\bar{x}_{j(k-1)} = 1 \ (k > 1)$, and $\bar{x}_{j(k)} = 0$ implies $\bar{x}_{j(k+)} = 0 \ (k < n_i)$. For each S_i let

$$p_i' = \begin{cases} p_{j(k)}, & if \ \bar{x}_{j(k)} = 1 \ and \ either \ k = n_i \ or \ \bar{x}_{j(k+1)} < 1, \\ 0, & if \ \bar{x}_{j(1)} < 1 \end{cases}$$

then, by reasoning similar to upon, the desired bound is

$$p_0 = \max\{p_1' + p_2' + \ldots + p_m', p_{\max}\}.$$

However, we can proceed as taking

$$K = \varepsilon P_0 / m,$$

so that

$$\frac{P^*}{K} \le \frac{2m}{\varepsilon}.$$

The list of pairs (Q, A) is processed with iteration over equivalence classes, rather than single items, making the computation rather similar to the large-item computation.

Initially the list contains only the pair $(0,0)$. At the end of iteration i, each pair (Q, A) is identified with a feasible solution containing items chosen from equivalence classes 1 through i. Suppose there are n_i items in equivalence class i, to perform iteration i, form n_i candidate items for each pair (Q, A) existing in the list at the end of iteration $i-1$. These candidate pairs are placed in n_i separate candidate lists. The $n_i + 1$ lists are then merged, eliminating dominated entries.

Iteration i requires $O(n_i m / \varepsilon)$ time, $O(m / \varepsilon)$ space,

and at most $O(m/\varepsilon)$ nodes are added to the tree used for backtracing. In this paper we obtain our resultes by using these methods.

SIMPLIFYING THE INPUT

We start by transforming any given instance into a standard form. Let,

$$r_{\max} = \max\{r_j; j \in J\}, \quad q_{\max} = \max\{q_j; j \in J\}$$
$$d_j = \min\{l_j + c_j^l, u_j + c_j^u\}, \quad D = \sum_{j \in J} d_j,$$

Moreover, let OPT denote the optimal solution value of the given instance.

Lemma 1

Without loss of generality, we can assume that the following holds:

$$1 \le OPT \le 3$$
$$\max\{D, r_{\max}, q_{\max}\} \le 1$$
$$0 \le l_j \le u_j \le 3 \;\&\; 0 \le c_j^u \le c_j^l \le 3$$

Proof (Capara et al., 2000)

Following Lageweg et al. (1977), if $J_j \prec J_k$ and $r_j > r_k$, then, we can reset $r_k := r_j$ and each feasible schedule will remain feasible. Similarly, if $q_j < q_k$ then we can reset $q_j := q_k$ without changing the objective function value of any feasible schedule. Thus, by repeatedly applying these updates we can always obtain an equivalent instance that satisfies,

$$J_j \prec J_k \Rightarrow (r_j \le r_k, q_j \ge q_k) \qquad (1)$$

Such a resetting requires $O(l)$ time, where l denotes the number of precedence constraints. Thus in the following we assume that (1) holds. A technique used by Hall et al. (1989) allows us to deal with only a constant number of release dates and delivery times. The idea is to round each release and delivery time down to the nearest multiple of $i\varepsilon$, for $i \in \mathrm{N}$. Since $r_{\max} \le 1$, the number of different release dates and delivery times is now bounded by $1/\varepsilon + 1$. Clearly, the optimal value of this transformed instance cannot be greater than OPT. Every feasible solution for the modified instance can be transformed into a feasible solution for the original instance just by adding ε to each job's starting time, and reintroducing the original delivery times. It is easy to see that the solution value may increase by at most 2ε. Therefore, we will assume henceforth that the input instance has a constant number of release dates and delivery times, and that condition (1) holds. We shall refer to this instance as I. By the previous arguments, OPT ¸ OPT (I), where OPT (I) denotes the optimal value for instance I.

Partitioning the set of Jobs

Partition the set of jobs in two subsets:

$$L = \{J_j : d_j > \varepsilon^2\}, \quad S = \{J_j : d_j \le \varepsilon^2\}$$

Let us say that L is the set of large jobs, while S the set of small jobs. Observe that the number of large jobs is bounded by $1/\varepsilon^2$ by Lemma 1. We further partition the set S of small jobs as follows. For each small job $J_j \in S$ consider the following three subsets of L:

$$\mathrm{Pre}(j) = \{J_j \in L : J_i \prec J_j\},$$

$$Suc(j) = \{J_j \in L : J_j \prec J_i\},$$

$$Free(j) = L - (\mathrm{Pre}(j) \cup Suc(j))$$

Let us say that $T(j) = \{\mathrm{Pre}(j), Suc(j), Free(j)\},$ represents a 3-partition of set L with respect to job J_j. We form the set $T = \cup_j T(j)$ which is the set of all distinct 3-partitions. We will index the items in set T by $T_1, ..., T_\tau$, where $\tau = |T|$. The number τ of distinct 3-partitions is clearly bounded by the number of small jobs and by $3^{|L|} \le 3^{1/\varepsilon^2}$, therefore $\tau \le \min\{n, 3^{1/\varepsilon}\}$. Now, we define the execution profile of a small job J_j to be a 3-tuple $\langle i_1, i_2, i_3 \rangle$ such that $r_j = \varepsilon . i_1$, $q_j = \varepsilon . i_2$ and $T(j) = T_{i_3}$ where $i_1, i_2 = 1, 2, \ldots, 1/\varepsilon$ and $i_3 = 1, \ldots, \tau$. For any given instance, the number of distinct execution profiles is clearly bounded by the number of jobs and, by the previous arguments, cannot be larger than $(1 + 1/\varepsilon)^2 \tau$.

Lemma 2

The number π of distinct execution profiles is bounded by

$$\pi \le \min\left\{n, 3^{1/\varepsilon^2}\left(1+1/\varepsilon\right)^2\right\}.$$

Partition the set S of small jobs into π subsets, $S_1, S_2, ..., S_\pi$ such that jobs belonging to the same subset have the same execution profile. Clearly,

$$\forall i \neq j \in \{1,2,..,\pi\} \qquad S = S_1 \cup S_2 \cup ... \cup S_\pi, \qquad S_i \cap S_j = \{\}.$$

Adding new precedences

Let us say that job J_h is a neighbor of set

$\{ S_i \ ; (i = 1,\ldots,\pi) \}$ if:

J_h Is a small job;

$J_h \notin S_i$;

there exists a precedence relation between job J_h and some job in S_i.

Moreover, we say that J_h is a front-neighbor (back-neighbor) of S_i if J_h is a neighbor of S_i and there is a job $J_j \in S_i$ such that $J_j \prec J_h$ $(J_h \prec J_j)$.

Let $(i = 1,\ldots,\pi)\ n_i = |S_i|$, and let $(J_{1.i},\ldots,J_{n_i,i})$ denote any fixed and complete ordering of the jobs from S_i that is consistent with the precedence relation. In the rest of this section we restrict the problem such that the jobs from S_i are processed according to this fixed ordering. Furthermore, every back-neighbor (front-neighbor) J_h of $(i = 1,\ldots,\pi)$ S_i must be processed before (after) every job from S_i. This can be accomplished by adding a directed arc from $J_{j,i}$ to $J_{j+1,i}$, for $j = 1,\ldots,n_i - 1$, and by adding a directed arc from J_h to $J_{1,i}$, if J_h is a back-neighbor of S_i, or an arc from $J_{n_i,i}$ to J_h, if J_h is a front-neighbor.

We will see later that this transformation does not considerably affect the optimal solution value. The idea is that the jobs from S_i are "similar" and small. So fixing an arbitrary order among them and updating precedence constraints appropriately does not deteriorate too much the solution quality.

Finally, note that the resulting precedence graph is without cycles. Indeed, it can be easily checked that a neighbor J_h of S_i for $(i = 1,\ldots,\pi)$ cannot simultaneously be a front neighbor and a back-neighbor of S_i. The number of added arcs can be bounded by $n + l$ (recall that l denotes the number of precedence constraints of the input instance).

We observe that condition (1) is valid also after these changes. Indeed, if J_h is a back-neighbor of S_i then there is a job $J_j \in S_i$ such that $J_h \prec J_j$, and therefore by condition (1) we have $r_h \le r_j$ and $q_h \ge q_i$. But, the jobs from S_i have the same release dates and delivery times, therefore $r_h \le r_j$ and $q_h \ge q_i$ for each $J_j \in S_i$. It follows that if we restrict J_h to be processed before the jobs from S_i, condition (1) is still valid. Similar arguments hold if J_h is a front-neighbor. Moreover, all the jobs from S_i have the same release dates and delivery times, therefore condition (1) is still satisfied, if we restrict these jobs to be processed in any fixed ordering that is consistent with the precedence relation.

Compact representation of job subsets

Consider set S_i (for $i = 1,\ldots,\pi$). Note that the number of jobs in S_i may be *O*(n). We replace the jobs from S_i with one compact job $J_i^{\#}$. Job $J_i^{\#}$ has the same release $(r_i^{\#})$ and delivery time $(q_i^{\#})$ as the jobs from S_i. Furthermore, if $J_j \prec J_k$ $(J_k \prec J_j)$, $J_i \in S_i, J_k \notin S_i$, then in the new modified instance we have $J_i^{\#} \prec J_k$ $(J_k \prec J_i^{\#})$, and the resulting precedence relation on $J_i^{\#}$ is "well-defined", since all jobs from S_i are predecessors (successors) of J_k.

Finally, the processing requirement of $J_i^{\#}$ is s pecified by a finite set of alternative pairs of processing times and costs determined as follows. Consider the following set $V_s = \{\varepsilon/\pi, \varepsilon/\pi(1+\varepsilon), \ldots, 3\}$ by simple algebra, we have $|V_s| = O(1/\varepsilon^2)$. Recall that π is the number of distinct execution profiles. Let $(B_i)\ A_i$ be the

value obtained by rounding $\left(\sum_{j\in S_i} u_j\right)\sum_{j\in S_i} l_j$ up to the nearest value from set V_s. The possible processing times for $J_i^{\#}$ are specified by set *Pi* of values from *VS* that fall in interval $[A_i, B_i]$, that is, $P_i := V_s \cap [A_i, B_i]$. For each value $p \in P_i$, we compute the corresponding cost value $C_i(p)$ as follows. Consider the problems (S_i, p) of computing the minimum sum of costs for jobs belonging to S_i, when the total sum of processing times is at most p, for every p 2 Pi. We can formulate problem (S_i, p) by using the following linear program LP(Si; p):

$$\min \sum_{j\in S_i}\left(\delta_j c_j^l + (1-\delta_j)c_j^u\right)$$

$$s.t. \sum_{j\in S_i}\left(\delta_j l_j + (1-\delta_j)u_j\right) \le p$$

$$, J_j \in S_i \; 0 \le \delta_j \le 1$$

By setting $\delta_j = 1 - x_j$, it is easy to see that an optimal solution for LP(Si; p) can be obtained by solving the following linear program:

$$\max \sum_{j\in S_i}\left(c_j^l - c_j^u\right)x_j$$

$$s.t. \sum_{j\in S_i}\left(u_j - l_j\right)x_j \le p - \sum_{j\in S_i} l_j$$

$$0 \le x_j \le 1 \quad , \quad J_j \in S_i$$

Note that $p - \sum l_j$ is non-negative, since $p \in P_i$ and the smallest value of P_i cannot be smaller than $\sum l_j$. The previous linear program corresponds to the classical knapsack problem with relaxed integrality constraints. By section 1 and by partially sorting jobs in nonincreasing $\frac{c_j^l - c_j^u}{u_j - l_j}$ ratio order, the set $\{LP(S_i, p);\ p \in P_i\}$ of $O(1/\varepsilon^3)$ many problems can be solved in $O\left(|S_i|(\log 1/\varepsilon)^{O(1/\varepsilon)}\right)$ time by using the method of section 3. For each value $p \in P_i$, the corresponding cost value $C_i(p)$ is equal to the optimal solution value of $\{LP(S_i, p);\ p \in P_i\}$ rounded up to the nearest value in set V_s. It follows that the number of alternative pairs of processing times and costs for each compact job $J_i^{\#}$ is bounded by the cardinality of set P_i. Furthermore, since $\sum_{\pi} |S_i| \le n$, it is easy to check that the amortized total time to compute the processing requirements of all compact jobs is $O\left(n(\log 1/\varepsilon)^{O(1/\varepsilon)}\right)$. Therefore, each set S_i is transformed into one compact job $J_i^{\#}$ with $O(1/\varepsilon^3)$ alternative pairs of costs and processing times. We use $S^{\#}$ to denote the set of compact jobs.

Now, let us consider the modified instance as described so far and turn our attention to the set L of large jobs. We map each large job $J_j \in L$ to a new job $J_i^{\#}$ which has the same release date, delivery time and set of predecessors and successors as job J_j, but a more restricted set of possible processing times and costs. This restricted set is chosen such that we can still obtain a near-optimal solution, so the restriction does not significantly deteriorate the objective function value. More precisely, let *Aj* (and *Bj*) be the value obtained by rounding $(u_j,) l_j$ up to the nearest value from set $V_L = \{\varepsilon^3, \varepsilon^3(1+\varepsilon), \ldots, 3\}$. The possible processing times for $J_i^{\#}$ are specified by set $P_j := V_L \cap \lfloor l_j, u_j \rfloor$. For each value $p \in P_j$, the corresponding cost value $C_j(p)$ is obtained by rounding up to the nearest value of set V_L the cost of job J_j when its processing time is p. We use $L^{\#}$ to denote the set of jobs obtained by transforming jobs from *L* as described so far.

Let $I^{\#}$ denote this modified instance. We observe that $I^{\#}$ can be computed in $O\left(n(\log 1/\varepsilon)^{O(1/\varepsilon)} + 2^{O(1/\varepsilon^2)}\right)$ time: the time required to partition the set of jobs into π subsets can be bounded by $O\left(n + l + 2^{O(1/\varepsilon^2)}\right)$ is the time to add new precedences; $O\left(n(1/\varepsilon^2)\log 1/\varepsilon\right)$ is the time to compute the alternative pairs of costs and processing times. Moreover, this new instance has at most $V = 3^{1/\varepsilon^2} \cdot (1 + 1/\varepsilon)^2 + 1/\varepsilon^2$ jobs; each job has a constant number of alternative pairs of costs and processing times. Now let us focus on $I^{\#}$ and consider the problem of finding the schedule for $I^{\#}$ with the minimum scheduling cost such that compact jobs can be preempted, while interruption is not allowed for jobs from $L^{\#}$.

RESULT

We have obtained a new linear complexity bound for the specific scheduling problem with controllable processing time by using a new complexity bound for the kKP.

ACKNOWLEDGEMENTS

The authors are grateful to Islamic Azad University Naragh Branch for the assistance rendered to our project. This thesis is extracted from the project “Using metaheuristic methods and polynomial time approximation algorithm for solving a strongly NP-hard scheduling problem”.

REFERENCES

Capara A, Kellerer H, Pfershy U, Pisinger D (2000). Approximation algorithms for knapsack problems with cardinality constraints, Eur. J. Oper. Res., 123: 333-345.

Garey MR, Johnson DS (1979). Computers and intractability; A guide to the theory of NP-completeness. W.H. Freeman.

Graham R, Lawler E, Lenstra J, Kan AR (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In Ann. Discrete Math., North–Holland, 5: 287–326.

Hall L, Shmoys D (1989). Approximation algorithms for constrained scheduling problems. In Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, pp. 134–139.

Hall L, Shmoys D (1990). Near-optimal sequencing with precedence constraints. In Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference, University of Waterloo Press, pp. 249–260.

Hall L, Shmoys D (1992). Jackson’s rule for single-machine scheduling: Making a good heuristic better. MOR: Math. Oper. Res., 17: 22–35.

Lageweg B, Lenstra J, Kan AR (1977). Minimizing maximum lateness on one machine: Computational experience and some applications. Statist. Neerlandica, 30: 25–41.

Lenstra JA, Kan R, Brucker P (1977). Complexity of machine scheduling problems. Ann. Oper. Res., 1: 343–362.

Zdrzalka S (1991). Scheduling jobs on a single machine with release dates, delivery times, and controllable processing times: Worst-case analysis. Oper. Res. Lett., 10: 519–532.

Mastrolilli M (2009). A linear time approximation scheme for single machine scheduling problem with controllable processing times. J. Algorithms, Article in press.

Megiddo M, Tamir A (2000). Linear time algorithms for some separable quadratic programming problems, Oper. Res. Lett., 13: 203-211.

Mastrolilli M, Hutter M (2006). Hybrid rounding techniques for knapsack problems. Discrete Appl. Math., 154: 640-649.