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 Mastroiilli (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 \((3/2 + \epsilon)\)-approximation algorithm by Mastroiilli (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 \(\epsilon\), 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, \(J_1, J_2, \ldots, J_n\), of \(n\) jobs is to be processed without interruption on a single machine. For each job \(J_j\) there is an interval \([l_j, u_j] \subseteq \mathbb{R}, 0 \leq l_j \leq u_j\), specifying its possible processing times. The cost for processing job \(J_j\) in time \(l_j\) is \(c_j l_j \geq 0\), and for processing it in time \(u_j\) the cost is \(c_j u_j \geq 0\). For any value \(\delta_j \in [0,1]\) the cost for processing job \(J_j\) in time \(\delta_j l_j + (1 - \delta_j) u_j\) is \(c_j \delta_j l_j + c_j (1 - \delta_j) u_j\), where \(\delta_j\) is the compression parameter. Additionally,
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 $s_j (\geq r_j)$ 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 $\delta_j$. 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 $J_j \prec J_k, (j, k \in \{1, 2, \ldots, n\})$ means that job $J_k$ must be processed after job $J_j$. Let $\theta$ be a permutation of the set $J$ that is consistent with the precedence constraints; $\theta$ denotes a processing order of jobs. Denote by $T(\delta, \theta)$ the (earliest) maximum delivery time of all the jobs for compression parameters $\delta = (\delta_1, \delta_2, \ldots, \delta_n)$ and processing order $\theta$. The total cost of compression parameters $\delta$ and processing order $\theta$ is defined as

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

The problem is to find $\theta$ and $\delta$ 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 $L_{\text{max}} \leq 0$ is NP-complete. Because of this equivalence, we shall denote the problem with fixed processing times as $|r_j| L_{\text{max}}$ (and $|r_j| L_{\text{max}}$ 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 $|r_j| L_{\text{max}}$, the running time of which are $O(n \log n + n(1/\varepsilon)^{O(1/\varepsilon^2)})$ and $O(n/\varepsilon)^{O(1/\varepsilon^2)}$. 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 ($|r_j|, prec |L_{\text{max}}$), Hall et al. (1990) give a PTAS. This consists of executing, for $\log_2 \Delta$ times, an extended version of their previous PTAS for $|r_j| prec |L_{\text{max}}$, where $\Delta$ 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 $|r_j| L_{\text{max}}$, 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 $|r_j|, prec |L_{\text{max}}$ 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
\begin{align*}
\max \quad P &= \sum_j p_j x_j \\
\text{s.t.} \quad A &= \sum a_j x_j \leq b, \quad x_j \in \{0,1\}.
\end{align*}

We may think of \( j \) as indexing items, with associated profits \( p_j \) and weights \( a_j \). The objective is to find the most profitable possible selection of items which 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 solution of the linear programming relaxation of the problem (Caprara 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 \( O(n^{1/\varepsilon-2} + n \log n) \) and \( O(n^{1/\varepsilon-1}) \) 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 \( O(n(\log 1/\varepsilon)^{O(1/\varepsilon)}) \) \( O(n + k(\log 1/\varepsilon)^{O(1/\varepsilon)}) \) 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 \( 0 \leq x_j \leq 1 \). If \( S_j, i = 1,2,\ldots,m, \) denotes the set of indices of items in the \( i \)th equivalence class, then we obtain a linear programming problem of the form

\begin{align*}
\max \quad & \sum_j p_j x_j \\
\text{s.t.} \quad & \sum_j a_j x_j \leq b, \\
& \sum_{j \in S_i} x_j \leq 1, \quad i = 1,2,\ldots,m, \\
& 0 \leq x_j \leq 1, \quad j = 1,2,\ldots,n.
\end{align*}

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 \( \{(a_j, p_j) : j \in S_i\} \cup \{(0,0)\} \).

Moreover, by dominance relaxations, this point can be assumed to lie on the "upper boundary" 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

\[ p_j(k) - p_j(k-1) \geq \frac{p_j(k+1) - p_j(k)}{a_j(k+1) - a_j(k)} > 0, \quad k = 1,2,\ldots,n_i - 1, \]

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

\[ a_{j(k)} = a_{j(k-1)} - a_{j(k-1)}, \quad p_{j(k)} = p_{j(k-1)} - p_{j(k-1)}. \]

With these new coefficients, we obtain a linear programming problem of the form
\[ \max \sum_j \bar{p}_j x_j \]
\[ \text{s.t.} \sum_j \bar{a}_j x_j \leq b, \]
\[ 0 \leq x_j \leq 1. \]

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 ratio, so that, without loss of generality,

\[ \frac{p_1}{a_1} \geq \frac{p_2}{a_2} \geq \ldots \geq \frac{p_n}{a_n}. \]

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

\[ P_0 \leq P^* \leq 2P_0, \]

Where

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

and where

\[ P_{\max} = \max \{p_j\}. \]

as before. This is because

\[ p_1 + p_2 + \ldots + p_j \leq P^*, \quad p_{j+1} \leq P_{\max} \leq P^*. \]

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

\[ \frac{P^*}{K} \leq \frac{2n}{\varepsilon}. \]

The computation can now be carried out in \( O(n^2/\varepsilon) \) time and space, exclusive of the time required to sort the items in \( \frac{p_j}{a_j} \) order.

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

\[ p'_i = \begin{cases} p_j(k), & \text{if } x_j(k) = 1 \text{ and either } k = n_i \text{ or } x_j(k+i) < 1, \\ 0, & \text{if } x_j(k) < 1 \end{cases} \]

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

\[ \frac{P^*}{K} \leq \frac{2m}{\varepsilon}. \]

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 results by using these methods.

SIMPLIFYING THE INPUT

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

\[
\begin{align*}
    r^\max &= \max\{r_j; j \in J\}; & q^\max &= \max\{q_j; j \in J\}; \\
    d_j &= \min\{c_j', u_j + c''_j\}, & D &= \sum_{j \in J} d_j.
\end{align*}
\]

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:

\[
\begin{align*}
    1 &\leq OPT \leq 3 \\
    \max\{D, r^\max, q^\max\} &\leq 1 \\
    0 &\leq l_{j} \leq u_{j} \leq 3 &\text{\&} & 0 \leq c''_{j} \leq c'_{j} \leq 3
\end{align*}
\]

Proof (Capara et al., 2000)

Following Lageweg et al. (1977), if $J_{j} < J_{k}$ 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,

\[
J_{j} < J_{k} \Rightarrow \left( r_{j} \leq r_{k}, q_{j} \geq q_{k} \right)
\]  

(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 $i \in \mathbb{N}$. Since $r^\max \leq 1$, 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 $\varepsilon$ 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 $2\varepsilon$. 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, $OPT \geq OPT \left( I \right)$, where $OPT \left( I \right)$ denotes the optimal value for instance $I$.

Partitioning the set of Jobs

Partition the set of jobs into two subsets:

\[
L = \left\{ J_{j} : d_{j} > \varepsilon^{2} \right\}, \quad S = \left\{ J_{j} : d_{j} \leq \varepsilon^{2} \right\}
\]

Let us say that $L$ is the set of large jobs, while $S$ is 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 $L$:

\[
\begin{align*}
    Pr_{e}(j) &= \left\{ J_{j} \in L : J_{j} < J_{j} \right\}, \\
    Su_{c}(j) &= \left\{ J_{j} \in L : J_{j} < J_{j} \right\}, \\
    Fr_{e}(j) &= L - (Pr_{e}(j) \cup Su_{c}(j))
\end{align*}
\]

Let us say that $T_{j} = \left\{ Pr_{e}(j), Su_{c}(j), Fr_{e}(j) \right\}$ 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 $T_{1}, \ldots, T_{\tau}$, where $\tau = \left| T \right|$. The number $\tau$ of distinct 3-partitions is clearly bounded by the number of small jobs and by $3^{\left| L \right|} \leq 3^{\varepsilon \varepsilon^{2}}$, therefore $\tau \leq \min\left\{ n, 3^{\varepsilon \varepsilon^{2}} \right\}$. Now, we define the execution profile of a small job $J_{j}$ to be a 3-tuple $\left< i_{1}, i_{2}, i_{3} \right>$ 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 $\pi$ of distinct execution profiles is bounded by

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

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

$$\forall i \neq j \in \{1, 2, \ldots, \pi\} \quad S_i = S_{i1} \cup S_{i2} \cup \cdots \cup S_{i\pi}, \quad S_i \cap S_j \neq \emptyset.$$

Adding new precedences

Let us say that job $J_h$ is a neighbor of set

$$\{ S_i \mid (i = 1, \ldots, \pi) \}$$

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 $(J_{i1}, \ldots, J_{in_i})$ 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 $J_{ij}$ to $J_{ij+1}$, for $j = 1, \ldots, n_i - 1$, and by adding a directed arc from $J_{in_i}$ to $J_{h}$, if $J_h$ is a back-neighbor of $S_i$, or an arc from $J_{in_i}$ 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 $n + \pi$ (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 $r_h \leq r_j$ and $q_h \geq q_j$. But, the jobs from $S_i$ have the same release dates and delivery times, therefore $r_h \leq r_j$ and $q_h \geq q_j$ 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 specified by a finite set of alternative pairs of processing times and costs determined as follows. Consider the following set $V_i = \{ e / \pi, e / \pi (1 + e), \ldots, 3 \}$ by simple algebra, we have $|V_i| = O(e^{-2})$. Recall that $\pi$ is the number of distinct execution profiles. Let $(B_e)A_i$ be the
value obtained by rounding \( \left( \sum_{j \in S_i} u_j \right) \sum l_j \) up to the nearest value from set \( V_j \). The possible processing times for \( J_i \) are specified by set \( P_i \) 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 \( p \in P_i \). We can formulate problem \((S_i, p)\) by using the following linear program \( LP(S_i; p) \):

\[
\min \sum_{j \in S_i} \left( \delta j^c_{j} + (1- \delta j^c_{j})c_j^u \right) \\
\text{s.t.} \sum_{j \in S_i} \left( \delta j_{l} + (1- \delta j_{l})u_j \right) \leq p \\
\quad J_j \in S_i, 0 \leq \delta j \leq 1
\]

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

\[
\max \sum_{j \in S_i} \left( c_j^c - c_j^u \right)x_j \\
\text{s.t.} \sum_{j \in S_i} \left( u_j - l_j \right)x_j \leq p - \sum_{j \in S_i} l_j \\
\quad 0 \leq x_j \leq 1, J_j \in S_i
\]

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 \( \frac{c_j^c - c_j^u}{u_j - l_j} \) ratio order, the set \( \{LP(S_i, p); p \in P_i\} \) of \( O(1/e^3) \) many problems can be solved in \( O(\|S_i\| (\log 1/e)^{O(1/e)}) \) 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 \( V_j \). 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 \( \sum_{i} |S_i| \leq n \), it is easy to check that the amortized total time to compute the processing requirements of all compact jobs is \( O(n(\log 1/e)^{O(1/e)}) \). Therefore, each set \( S_i \) is transformed into one compact job \( J^*_i \) with \( O(1/e^3) \) 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 \( A_j \) (and \( B_j \)) be the value obtained by rounding \( \left( u_j \right) \) up to the nearest value from set \( V_L = \{e^3, e^3(1 + e), \ldots, 3\} \). The possible processing times for \( J^*_i \) are specified by set \( P_j := V_L \cap [l_j, u_j] \). 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 \( O(n(\log 1/e)^{O(1/e)} + 2^{O(1/e^3)}) \) time: the time required to partition the set of jobs into \( \pi \) subsets can be bounded by \( O(n \log 1/e + 2^{O(1/e^3)}) \) is the time to add new precedences; \( O(n(1/e^3) \log 1/e) \) is the time to compute the alternative pairs of costs and processing times. Moreover, this new instance has at most \( V = 3^{1/e^3} \cdot (1 + 1/e)^2 + 1/e^2 \) 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


