Tabu-search for Single Machine Scheduling With Controllable Processing Times

Zuren Feng
Xi'an Jiaotong University
28 Xianning Lane
Xi'an, China
fzr9910@mail.xjtu.edu.cn

Kailiang Xu
Xi'an Jiaotong University
28 Xianning Lane
Xi'an, China
xl.8046@stu.xjtu.edu.cn

ABSTRACT

In this paper, we consider scheduling n jobs with arbitrary release dates and due dates on a single machine, where jobs' processing times can be controlled by the allocation of a common resource, and the operation is modeled by a non-linear convex resource consumption function. The objective is to obtain an optimal processing sequence as well as optimal resource allocation, such that all the jobs could be finished no later than their due dates, and the resource consumption could be minimized. Since the problem is strongly $\mathcal{NP}$-hard, a two-layer-structured algorithm based on tabu-search is presented. The computational result compared with a branch-and-bound algorithm showed the algorithm is capable for producing optimal and near optimal solution for large sized problems in acceptable computational time.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem solving, Control Methods, and Search—*heuristic methods, scheduling*; G.1.6 [**Numerical Analysis**]: Optimization—*convex programming, tabu search*

General Terms

Algorithm

Keywords

Single machine scheduling,Tabu-search,Convex resource consumption function

1. INTRODUCTION

In this paper we consider following scheduling problem: Schedule a set of jobs $\mathscr{J} = \{1, 2, \ldots, n\}$ with arbitrary release dates r_j and due dates d_j on a single machine. Jobs' processing times can be controlled by allocating a common resource through a non-linear convex resource consumption function. The objective is to determine simultaneously the optimal processing sequence as well as resource allocation, such that all the jobs could be finished no later than their due dates, and the total resource consumption could be minimized.

A survey on US manufacturing practices pointed out, that meeting due dates is the single most important scheduling criterion[2]. In fact, for many manufacturers, delay of shipment can normally bring unacceptable huge penalty and loss of business credit. In many practical environments, jobs' processing times could be controlled by allocating a common resource to each job. Such resource normally include fuel, gas, electricity power, budget and human resource. An often asked question thus arise: How to determine jobs' processing sequence as well as processing times, such no delay of shipment would occur, and the total resource consumption is minimized?

For scheduling problems where job-processing times are controllable, a so called resource consumption function is employed as the connection between job-processing times and resource allocation. In our paper, we adopt non-linear convex function

$$p_j(u_j) = \left(\frac{\omega_j}{u_j}\right)^k, \quad a_j \le p_j \le b_j \tag{1}$$

as the resource consumption function, where ω_j is a positive parameter that represents the workload of job j, and k is a positive constant determined by the related environment. This function has the advantage that it reflects the *law of diminishing marginal returns*, which states that productivity increases at a decreasing rate with the amount of resource employed[8]. For this reason, it is widely used in continuous resource allocation theory. For job scheduling, Shabtay studied minimizing makespan[5], maximum lateness[6] and total weighted flowtime[7] on single machine. Reader may refer to survey[8] for more about scheduling problems with controllable processing times.

The computational complexity of our problem, where jobs have arbitrary release dates and due dates, is associated with the following decision problem: given an arbitrary resource allocation to jobs (that is, when job-processing times are fixed), does there exist a schedule in which no job is completed later than its due date? Since this problem is well known to be strongly $\mathcal{NP}$-hard, our problem is also of strong $\mathcal{NP}$-hardness. Janiak[6] has studied scheduling jobs with common due dates, where job processing times are controllable through a linear resource consumption function. Lee and Lei[3] has also studied the same problem with common release dates and non-identical due dates. In these early

GEC'09, June 12–14, 2009, Shanghai, China.

research works, either jobs' release dates or due dates are supposed to be identical, such that simple list rules such as EDD(earliest due date) can be applied to determine jobs' processing sequence in spite of jobs' actual processing times. In this way, the scheduling problem is simplified as a resource allocation problem, which can be solved in polynomial time. Obviously this is an important deviation from reality in most cases. For example, the existence of job release dates is consistent with many MRP-controlled environments. So in this paper, we develop a two-layer-structured algorithm. The upper layer uses a tabu-search approach to search for the optimal or near optimal job-processing sequence, while the lower layer calculates the optimal resource allocation of the processing sequence produced by the upper layer.

Our paper is divided into five sections. In Section 2, we solve the problem of optimally allocating resource to jobs. In Section 3, we develop a tabu-search algorithm to search for optimal or near optimal processing sequence. In Section 4, we present numerical experiment results. A summary is presented in Section 5.

2. OPTIMAL RESOURCE ALLOCATION

The problem of optimally allocating resource to jobs can be formulated as:

(P1)

$$\begin{aligned} \min \quad & U = \sum_{j=1}^{n} u_{\sigma(j)}(p_{\sigma(j)}) \\ \text{s.t.} \quad & \max_{1\le l\le j}\left\{ r_{\sigma(l)} + \sum_{i=l}^{j} p_{\sigma(j)} \right\} \le d_{\sigma(j)} \qquad j = 1,2,\ldots,n \\ & a_{\sigma(j)} \le p_{\sigma(j)} \le b_{\sigma(j)} \qquad j = 1,2,\ldots,n \end{aligned}$$

where $\sigma = \{\sigma(1), \sigma(2), \ldots, \sigma(n)\}$ is jobs' permutation, in which $\sigma(l)$ denotes the lth job to be processed, and $u_j(p_j)$ is jobs' resource consumption as the function of job-processing times p_j. It can be observed that for adjacent jobs $\sigma(j)$ and $\sigma(j+1)$, when $d_{\sigma(j)} \le r_{\sigma(j+1)}$, jobs $\{\sigma(1), \sigma(2), \ldots, \sigma(j)\}$ and jobs $\{\sigma(j+1), \sigma(j+2), \ldots, \sigma(n)\}$ can be divided into two independent subsets and solved separately. Thus in the follows, without loss of generality, we assume for $1 \le j \le n-1$, $d_{\sigma(j)} > r_{\sigma(j+1)}$.

The problem can be transformed into an acyclic directed graph as Fig.1 shows. The nodes of the graph represent jobs' release dates and due dates (ordered in ascending order). The arc that links two nodes represents a subset of jobs that start processing from the emanating node and complete at the sinking node. The length of the arc represents the resource consumption of jobs in the subset. In this way, a path from $r_{\sigma(1)}$ to $d_{\sigma(n)}$ makes up a feasible solution of the resource allocation problem, and the shortest path among them would be the optimal solution.

Consider an arc in the graph. Define $\sigma' \subset \sigma$ the permutation of jobs contained in the associated sub-problem of the arc. The emanating node of the arc may be a release date ($r_{\sigma(s)}$) or due date ($d_{\sigma(s)}$), and the sinking node may also be a release date ($r_{\sigma(t)}$) or due date ($d_{\sigma(t)}$), where $s \le t$. Define $S_{\sigma'}$ the emanating node and $C_{\sigma'}$ the sinking node of the arc. According to the type of the emanating node and sinking node, the associated sub-problem may contain jobs as follows:

1. If $S_{\sigma'} = r_{\sigma(s)}$ and $C_{\sigma'} = d_{\sigma(t)}$, $\sigma' = \{\sigma(s), \ldots, \sigma(t)\}$;
2. If $S_{\sigma'} = r_{\sigma(s)}$ and $C_{\sigma'} = r_{\sigma(t)}$, $\sigma' = \{\sigma(s), \ldots, \sigma(t-1)\}$;
3. If $S_{\sigma'} = d_{\sigma(s)}$ and $C_{\sigma'} = r_{\sigma(t)}$, $\sigma' = \{\sigma(s+1), \ldots, \sigma(t-1)\}$;
4. If $S_{\sigma'} = d_{\sigma(s)}$ and $C_{\sigma'} = d_{\sigma(t)}$, $\sigma' = \{\sigma(s+1), \ldots, \sigma(t)\}$;

The length of the arc is calculated in two stages. In the first stage, calculate the resource allocation of jobs in σ' as following problem:

(P1.1)

$$\begin{aligned} \min \quad & U_{s,t} = \sum_{\sigma(j)\in\sigma'} u_{\sigma(j)}\left(p_{\sigma(j)}\right) \\ \text{s.t.} \quad & \sum_{\sigma(j)\in\sigma'} p_{\sigma(j)} = C_{\sigma'} - S_{\sigma'} \\ & a_{\sigma(j)} \le p_{\sigma(j)} \le b_{\sigma(j)}, \qquad \forall \sigma(j) \in \sigma' \end{aligned}$$

The solution of P1.1 is presented in Appendix A. It can be seen the release date constraints and due date constraints in P1 are relaxed in P1.1, thus the feasibility of the solution should be checked. For every job $\sigma(j)$ in σ', following conditions shall be satisfied to make sure no job in the sub-problem will be completed later than its due date:

$$\begin{aligned} \max_{\sigma(i)\in\sigma', i\le j}\left\{ r_{\sigma(i)} + \sum_{\sigma(l)\in\sigma', i\le l\le j} p_{\sigma(l)} \right\} &\le d_{\sigma(j)} \\ S_{\sigma'} + \sum_{\sigma(l)\in\sigma', l\le j} p_{\sigma(l)} &\le d_{\sigma(j)} \end{aligned}$$

If the solution is feasible, the corresponding arc is denoted as feasible arc, and the length of the arc is the total resource consumption of the solution. If the solution is infeasible, the arc is removed from the graph. When all the feasible arcs are listed, the original problem is transformed into a directed graph.

For n jobs, there are at most $2n$ nodes in the graph. Some of the nodes can be removed from the graph without disturbing the solution, because the shortest path will certainly not pass them. Consider adjacent jobs $\sigma(i)$ and $\sigma(j)$, where $i < j$. Define $\underline{C}_{\sigma(i)}$ the earliest completion time of job $\sigma(i)$, calculated as

$$\underline{C}_{\sigma(i)} = \max_{1\le l\le i}\left\{ r_{\sigma(l)} + \sum_{k=l}^{i} a_{\sigma(k)} \right\}$$

It can be seen if $r_{\sigma(j)} < \underline{C}_{\sigma(i)}$, any arc that sinks to vertex $r_{\sigma(j)}$ would be infeasible, because there would be no enough processing time for jobs in the associated sub-problem. Since there is no arc that sinks to $r_{\sigma(j)}$, arc that emanates from $r_{\sigma(j)}$ will be of no meaning. Thus node $r_{\sigma(j)}$ can be removed from consideration.

Similarly, define $\bar{S}_{\sigma(j)}$ the latest start time of job $\sigma(j)$. $\bar{S}_{\sigma(j)}$ equals

$$\bar{S}_{\sigma(j)} = \min_{j\le l\le n}\left\{ d_{\sigma(l)} - \sum_{k=j}^{l} a_{\sigma(k)} \right\}$$

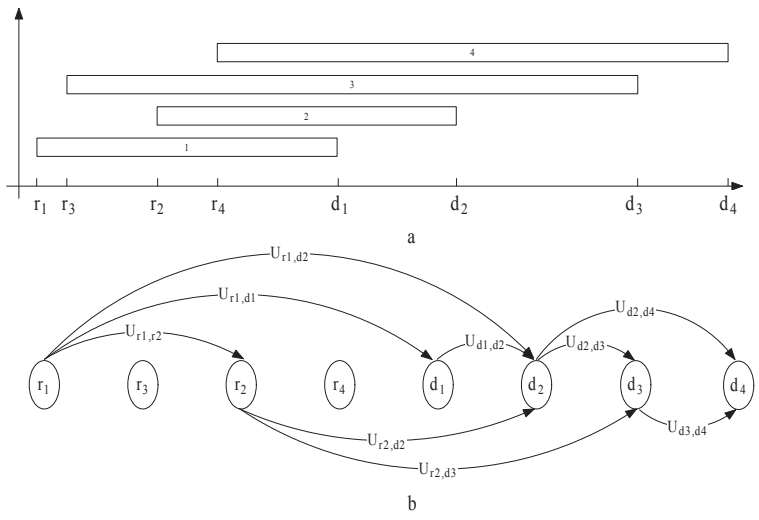

Figure 1: (a) The Gantt chart of jobs' available processing space and actural processing times(darkened). (b) The acyclic directed graph, where the shortest path is emphasized.

When $d_{\sigma(i)} > \bar{S}_{\sigma(j)}$, any arc that emanates from node $d_{\sigma(i)}$ would be infeasible. So in this case, node $d_{\sigma(i)}$ should also be removed form consideration.

When the directed graph is constructed, the shortest path from $r_{\sigma(1)}$ to $d_{\sigma(n)}$ can be found by a dynamic programming algorithm. As a whole, the algorithm can be stated formally as follows:

ALGORITHM 1. *1. List jobs' release dates and due dates in ascending order of their values as the nodes of the graph. Remove $r_{\sigma(j)}$ from consideration, if $r_{\sigma(j)} < \underline{C}_{\sigma(j-1)}$. Remove $d_{\sigma(j)}$ from consideration, if $d_{\sigma(j)} > \bar{S}_{\sigma(j+1)}$. Re-index other nodes as $1, 2, \ldots, m$ for convenience, where $m \leq 2n$;*

2. Draw an arc from node i to node j, where $1 \leq i < j \leq m$, and construct the associated sub-problem. Solve the sub-problem using Algorithm 1. If the solution of the sub-problem is feasible, calculate the total resource consumption as the length of the arc, denoted as $U_{i,j}$. Otherwise, discard the arc;

3. Search for the shortest path using following recursive relation:

(a) Initial condition:

$$U_1 = 0$$

(b) Recursive relation:

$$U_j = \min_{1 \leq i < j} \left\{ U_i + U_{i,j} \right\}, \qquad j = 2, 3, \ldots, m$$

(c) Optimal value:

$$U^* = U_m$$

4. Record solutions of sub-problems associated with arcs on the shortest path as the solution for resource allocation problem.

3. TABU-SEARCH FOR OPTIMAL PROCESSING SEQUENCE

3.1 Problem solvability and initial solution

For a reliable scheduling algorithm, it is important to provide at least one feasible solution (though may not be optimal) for the given problem. Besides, it is also necessary to find an initial solution for the tabu-search algorithm to start with. This requirement can be achieved by following strategy: Compress job-processing times to their lower bound a_j, and search for the first permutation that makes sure no job would be completed later than its due date. Obviously, this procedure is equivalent to the problem of $1|r_j|L_{\max}$, which is well know to be strongly $\mathcal{NP}$-hard. Since it is required to provide at least one feasible solution (unless the problem is unsolvable), we present a branch-and-bound method based on the algorithm developed by McMahon and Florian[4].

First, job-processing times are all compressed to their lower bound a_j. The algorithm starts from constructing heuristic solution by EDD rule. The resulting schedule usually contains a number of subsets of jobs, or *blocks*, in which jobs are processed continuously without gap among them. Define $\mathscr{P}(j)$ the subset of jobs that precede j in its block and j itself. Let C_j be the completion time of job j in the resulting schedule. The lower bound of the problem can be obtained from the schedule as

$$LB = \max_{j \in \mathscr{J}} \left\{ \max \left\{ C_j - \max_{i \in \mathscr{P}_j} d_i, r_j + a_j - d_j \right\} \right\}$$

Let the *critical job* be the job that has the maximum lateness in a given schedule, while the *generating set* of critical job j, $\mathscr{G}(j)$, be the set of jobs $i \in \mathscr{P}(j)$ such that their due dates are greater than that of j. It can be observed that for critical job j and job k, if $k \in \mathscr{P}(j)$ but $k \notin \mathscr{G}(j)$, then any solution generated by a relative permutation change of k and j can bring no improvement and may thus be ignored. However, if $k \in \mathscr{G}(j)$, then scheduling k later than j will generate an associated solution that might be an im-

provement(A convenient way to force job k scheduled later than j is to create a new problem where $r'_k = r_j$.). Thus, if the current schedule is not feasible ($L_{\max} > 0$), a set of associated solutions can be generated, and the procedure is repeated among them until one feasible solution is found, or the problem is determined to be unsolvable. The compact procedure is presented below:

ALGORITHM 2. *1. Construct the first schedule by EDD, and calculate the corresponding LB.*

(a) If $LB > 0$, stop. The problem is unsolvable.

(b) Find the critical job j and its lateness, $L_{\max}$. If $L_{\max} \leq 0$, select the solution as initial schedule and exit.

2. Find the generating set $\mathscr{G}_j$ and create $|\mathscr{G}_j|$ new nodes. For each of these nodes, let $r'_k = r_j$ for $k \in \mathscr{G}_j$, generate new schedule by EDD rule, calculate lower bounds LB^{loc}, critical jobs and its lateness, $L^{loc}_{\max}$.

3. For each new node, if $L^{loc}_{\max} \leq 0$, select the schedule as the initial solution and exit. If $LB^{loc} > 0$, close the node.

4. If all the nodes are closed, stop, the problem is unsolvable.

5. Select among open nodes, the node with the least lower bound. Call this node the parent node, and go to Step 2.

3.2 Schedule representation, neighborhood generation and candidate list strategy

As job-preemption is not allowed for our problem, any schedule can be specified by a simple permutation of n jobs. For permutation problems, swap moves and insert moves are two frequently used move types. For a schedule with n jobs, there are $n-1$ schedules in the neighborhood of the original schedule generated by swap moves, and at most $n(n-1)$ schedules by insert moves. Evaluation against all the neighbors in each iteration would be time consuming and not necessary. In the follows we present candidate list strategy to restrict the number of schedules in the neighborhood to be examined in each iteration.

For each schedule produced in iteration, there are a number of critical jobs (the definition of critical job is here different from that in the first part of this section). A *critical job* is a job such that either its start time equals its release date, or its completion time equals its due date. We define other jobs as *relaxed jobs*, that is, their start times are strictly later than their release dates, and their completion times are strictly earlier than their due dates. For relaxed job j, define $\mathscr{R}(j)$ *relaxed job set* that contains relaxed jobs (including job j) between neighboring critical jobs. For a critical job j, define $\mathscr{P}(j)$ its *preceding relaxed job set* that contains relaxed jobs that precede job j consecutively without interruption by critical job, define $\mathscr{S}(j)$ its *succeeding relaxed job set* that contains relaxed jobs that succeed job j consecutively without interruption by critical job. We discuss in the follows under what situation a move may lead to a new schedule with smaller total resource consumption.

THEOREM 1. *An insert move between relaxed jobs in the same relaxed job set leads to no reduction of total resource consumption.*

PROOF. (By contradiction) Suppose in schedule σ_1, there are jobs i preceding j in the same relaxed job set, and job k succeeds job j adjacently (Fig.2a). Assume by inserting job j to the front of i, the total resource consumption of the new schedule σ_2 is smaller than that of schedule σ_1, that is, $U_{\sigma_2} < U_{\sigma_1}$ (Fig.2b).

Construct a new schedule σ_3 from schedule σ_2 as follows: Set the due date of job j to be the start time of job k, that is, $d'_j = S_k$. For jobs l between job i and job k (including job i), set their release dates $r'_l = \min\{r_l, r_j\}$. Insert job j back to the front of job k (Fig.2c). It can be seen that the feasibility of job-processing times of schedule σ_2 is not violated in schedule σ_3, thus the resource consumption of schedule σ_3 is smaller than or equal to that of σ_2, that is, $U_{\sigma_3} \leq U_{\sigma_2}$.

Construct a new schedule σ_4 from schedule σ_3 by setting the due date of job j to be d_j (Fig.2d). Since $S_k < d_j$, this is a relaxation of the due date constraint of job j. Thus the total resource consumption of schedule σ_4 is smaller than or equal to that of σ_3, that is, $U_{\sigma_4} \leq U_{\sigma_3} \leq U_{\sigma_2} < U_{\sigma_1}$.

Construct schedule σ_5 from σ_1 as follows: For jobs l from job i to job j, set their release dates $r'_l = \min\{r_l, r_j\}$ (Fig.2d). Since $r_l < S_l$ in σ_1, the relaxation of their release date constraints has no influence to the resource allocation. So we have $U_{\sigma_5} = U_{\sigma_1}$.

It can be observed that the jobs' permutation, release date constraints and due date constraints of schedule σ_4 and σ_5 are exactly the same. Thus they have the same resource allocation and consumption, that is, $U_{\sigma_4} = U_{\sigma_5} = U_{\sigma_1}$. Obviously, this causes a contradiction. For other insert moves between job i and job j, contradiction can also be deduced in the same way. Thus the theorem is proved. □

Figure 2: An illustration to the proof of Theorem 1. (a) Schedule σ_1. (b) Schedule σ_2. (c) Schedule σ_3. (d) Schedule σ_4 and σ_5.

COROLLARY 1. *Any swap move between jobs in the same relaxed job set leads to no reduction of total resource consumption.*

THEOREM 2. *For job $i \in PR(j)$, if $S_j = r_j$ and $C_j < d_j$, then inserting i to the back of j leads to no reduction of the total resource consumption.*

PROOF. (By contradiction) Suppose in schedule σ_1, job $i \in PR(j)$, and job k precedes job i adjacently (Fig.3a). Assume by inserting job i to the back of job j, total resource consumption of schedule σ_2 is smaller than that of schedule σ_1, that is, $U_{\sigma_2} < U_{\sigma_1}$ (Fig.3b).

Construct a new schedule σ_3 from schedule σ_2 as follows: Set the release date of job i to be the completion time of job k, that is, $r'_i = C_k$. For jobs l between job k and job j (including job j), set their due dates $d'_l = \max\{d_l, d_i\}$. Insert job i back to the front of job k (Fig.3c). It can be seen that the feasibility of job-processing times of schedule σ_2 is not violated in schedule σ_3, thus the resource consumption of schedule σ_3 is smaller than or equal to that of σ_2, that is, $U_{\sigma_3} \leq U_{\sigma_2}$.

Construct a new schedule σ_4 from schedule σ_3 by setting the release date of job i to be d_i (Fig.3d). Since $r'_i = C_k > r_i$, this is a relaxation of the release date constraint of job i. Thus the total resource consumption of schedule σ_4 is smaller than or equal to that of σ_3, that is, $U_{\sigma_4} \leq U_{\sigma_3} \leq U_{\sigma_2} < U_{\sigma_1}$.

Construct schedule σ_5 from σ_1 as follows: For jobs l from job i to job j, set their due dates $d'_l = \max\{d_l, d_i\}$ (Fig.2d). Since $d_l > C_l$ in σ_1, the relaxation of their release date constraints has no influence to the resource allocation. So we have $U_{\sigma_5} = U_{\sigma_1}$.

It can be observed that jobs' permutation, release date constraints and due date constraints of schedule σ_4 and σ_5 are exactly the same. Thus they have the same resource allocation and consumption, that is, $U_{\sigma_4} = U_{\sigma_5} = U_{\sigma_1}$. Obviously, this causes a contradiction. The theorem is proved. □

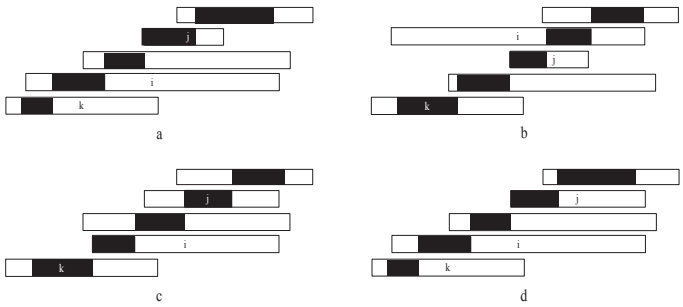

Figure 3: An illustration to the proof of Theorem 2. (a) Schedule σ_1. (b) Schedule σ_2. (c) Schedule σ_3. (d) Schedule σ_4 and σ_5.

THEOREM 3. *For job $j \in \mathscr{S}(i)$, if $S_i = r_i$ and $r_i \leq r_j$, then inserting j to the front of j leads to no reduction of the total resource consumption.*

PROOF. The theorem can be proved in the same way as Theorem 2. □

THEOREM 4. *For job $j \in \mathscr{S}(i)$, if $C_i = d_i$, $S_i > r_i$, then inserting j to the front of i leads to no reduction of the total resource consumption.*

PROOF. The theorem can be proved in the same as Theorem 2. □

THEOREM 5. *For job $i \in \mathscr{P}(j)$, if $C_j = d_j$ and $d_i \leq d_j$, then inserting i to the back of j leads to no reduction of the total resource consumption.*

PROOF. The theorem can be proved in the same way as Theorem 2. □

It can be seen from above discussion, that when swap move or insert move is applied between certain jobs, the total resource consumption of the resulting neighboring schedule can certainly not be reduced. Thus we only need to try other jobs that may bring improvement. Define $\mathscr{G}(j)$ the *generating set* of critical job j. If $S_j = r_j$, $\mathscr{G}_j$ contains jobs $i \in \mathscr{S}(j)$ with $r_i < r_j$. If $C_j = d_j$, $\mathscr{G}(j)$ contains jobs $i \in \mathscr{P}(j)$ with $d_i > d_j$. The candidate list strategy can be described as follows:

1. If $S_j = r_j$, generate candidate neighboring schedule by inserting job $i \in \mathscr{G}(j)$ to the front of j;
2. If $C_j = d_j$, generate candidate neighboring schedule by inserting job $i \in \mathscr{G}(j)$ to the back of j.

3.3 Tabu-list management

The tabu-list is managed according to the principles presented by Glover[1]. The maximum number of entries in the tabu-list is fixed to be 10. Experiment shows the algorithm is not sensitive against this value. An aspiration criterion is used, which allows a tabu move to be performed if it leads to a solution better than the best solution obtained so far.

3.4 Optimality criterion and termination condition

Local search algorithms, such as simulated annealing and tabu-search, start out with a complete schedule, which may be selected arbitrarily, and then try to obtain a better schedule by manipulating the current schedule. This behavior makes it difficult to evaluate the solution. Thus a local search algorithm does not guarantee an optimal solution. However, for our problem, it is possible to recognize the optimality of some of the solutions that have special structure. It can be observed that the tabu-search would stop when all the generating sets of a schedule are empty. For such a schedule, no improvement can be obtained, and it is natural to be considered as an optimal schedule. In the follows we prove this observation.

THEOREM 6. *A schedule is optimal, if all its generating sets are empty.*

PROOF. Suppose in schedule σ, all the generating sets are empty. Denote $\mathcal{C}$ the set that contains all the critical jobs $\sigma(i)$ with $s_{\sigma(i)} = r_{\sigma(i)}$. Relax part of the release date constraints as follows:

$$r'_{\sigma(j)} = \max_{\sigma(i)\in\mathcal{C}, i<j} \{r_{\sigma(i)}\}$$

Order jobs according to EDD rule, and denote the new schedule σ'. Since jobs in σ' have compatible release dates and due dates, and are ordered by EDD rule, σ' is optimal for the relaxed problem. Obviously the optimal solution of the relaxed problem is no larger than that of the original problem, thus we have

$$U_\sigma \geq U^* \geq U_{\sigma'}$$

It can be verified easily that the resource allocation and job-processing times in σ' is the same as those in σ, thus we have $U_\sigma = U^*$, that is, σ is the optimal solution. □

Normally a tabu-search algorithm terminates after certain steps of iteration. For our algorithm we terminate the iteration after 50 processing sequences are already generated.

Besides, the optimal criterion introduces one more termination condition. Experiments showed this condition reduces computational time effectively.

4. COMPUTATIONAL EXPERIMENT

To evaluate the performance of the presented tabu-search algorithm, we conduct a set of computational experiments in this section. The algorithm was implemented in C++ and ran on a personal computer with 2GHz Pentium processor and 512M RAM.

The experiment was carried out upon groups of problems. Each group contains 50 problem instances. All the problem instances in the experiment were generated at random. The discrete uniform distribution of jobs' workload ω_j ranges in $[10, 40]$. The width of jobs' feasible processing space, $\delta_j = d_j - r_j$, is an uniform distribution that ranges in $[10, 50]$. The lower bound of jobs' processing times is generated by $a_j = \alpha_j * \delta_j$, and the upper bound is generated by $b_j = \beta_j * \delta_j$, where α_j is a uniform distribution between $[5\%, 30\%]$, β_j is a uniform distribution between $[50\%, 80\%]$. Jobs' release dates r_j are also generated randomly with equal probability, and the range of the distribution for each group is different. The constant value $k = 1$.

First we check the effectiveness of the tabu-search algorithm. In order to check the optimality of the solution, we developed a branch-and-bound algorithm, which is capable for producing optimal solution for small and media sized problems. Thus the experiment was carried out over problem groups with problem size $n = 20$ and $n = 30$. The experimental result is shown in Table 1, which contains following contents:

- $U(r_j)$: The range of the uniform distribution of jobs' release dates;
- #: The number of instances for which the specified algorithm found an optimal solution;
- AvgRD(%): The average relative deviation in percent from optimum, where the relative deviation RD for a given instance is calculated according to the formula:

$$\mathrm{RD} = \frac{U - U^*}{U^*} 100$$

  in which U is the total resource consumption of the schedule produced by the tabu-search algorithm, and U^* is the optimal resource consumption produced by the branch-and-bound algorithm;
- MaxRD(%): The maximum relative deviation over all the instances tested;
- Time(s): The average computational time in millisecond for the specified algorithm.

Then we check the performance of the tabu-search algorithm over media sized problems, that is, groups sizing $n = 60$ and $n = 70$. In this part it is hard for branch-and-bound algorithm to produce optimal solution in acceptable time, thus the algorithm terminates when it finds an optimal solution, or at most 90 minutes have passed. Table 2 shows the experimental results, which contains following contents:

- $U(r_j)$: The range of the uniform distribution of jobs' release dates;
- #: The number of instances for which the algorithm found a solution equal to the best solution known (i.e. best solution found by any of the algorithms);
- $\leq 1\%$ ($\leq 2\%$, $\leq 5\%$, $\leq 10\%$): The number of instances for which the algorithm found a solution whose deviation against the best solution is equal to or smaller than 1% ($\leq 2\%$, $\leq 5\%$, $\leq 10\%$);
- Time(s): The average computational time in second for the specified algorithm(tabu-search or branch-and-bound).

Table 1: Experimental results for small sized problems

n	$U(r_j)$	TS				BB
		#	AvgRD (%)	MaxRD (%)	Time (s)	Time (s)
20	[0, 50]	50	0	0	0.55	1.02
	[0, 100]	50	0	0	0.76	4.20
	[0, 150]	49	0	0.05	1.13	5.74
30	[0, 50]	50	0	0	1.08	5.68
	[0, 100]	50	0	0	1.16	21.98
	[0, 150]	50	0	0	7.83	182.35
	[0, 200]	45	0	0.05	5.39	197.40

For large sized problems($n = 90$ and $n = 100$), however, it is difficult for branch-and-bound algorithm to produce sensible solution in acceptable computational time. Thus in this part we check the computational time of the tabu-search algorithm. Table 3 contains following contents:

- $U(r)$: The range of the uniform distribution of jobs' release dates;
- ≤ 50(s): The number of instances for which the computational time does not exceed 50 seconds;
- AvgTime(s): The average computational time in second for problem instances in the group;
- MaxTime(s): The maximum computational time in second for problem instances in the group;
- MinTime(s): The minimum computational time in second for problem instances in the group.

From above experiments it can be seen that our tabu-search algorithm is a very promising algorithm. It is capable for producing optimal or near optimal solution for problems with different size, distribution and parameter, while the computational time consumption keeps acceptable even for large problem.

5. CONCLUSION

In this paper we studied the problem of scheduling n jobs on a single machine. Jobs' release dates and due dates are arbitrary, and their processing times can be controlled by allocating a common resource to each job through a nonlinear convex resource consumption function. The objective

Table 2: Experimental results for media sized problems

n	$U(r_j)$	TS						BB					
		#	≤ 1 (%)	≤ 2 (%)	≤ 5 (%)	≤ 10 (%)	Time (s)	#	≤ 1 (%)	≤ 2 (%)	≤ 5 (%)	≤ 10 (%)	Time (s)
60	[0, 200]	50	50	50	50	50	16.631	41	48	49	50	50	1721.997
	[0, 250]	42	50	50	50	50	22.994	34	44	47	50	50	3400.879
	[0, 300]	47	50	50	50	50	39.084	25	37	44	50	50	3724.691
	[0, 350]	47	50	50	50	50	63.313	20	34	44	47	50	4551.092
70	[0, 250]	50	50	50	50	50	30.781	30	43	47	50	50	3506.741
	[0, 300]	47	50	50	50	50	77.534	20	34	42	45	50	4915.112
	[0, 350]	50	50	50	50	50	105.591	12	32	38	46	50	4955.417
	[0, 400]	49	50	50	50	50	139.052	8	29	39	46	50	5296.638

Table 3: Experimental results for large sized problems

n	$U(r)$	≤ 1 (min)	≤ 5 (min)	≤ 10 (min)	≤ 30 (min)	≤ 60 (min)	≤ 90 (min)	AvgTime (s)	MinTime (s)	MaxTime
90	[0, 350]	19	42	46	50	50	50	197.752	5.165	830.094
	[0, 400]	10	38	46	50	50	50	246.189	5.235	1058.484
	[0, 450]	9	27	39	50	50	50	320.187	25.687	1297.516
	[0, 500]	7	27	35	50	50	50	412.607	14.204	1390.515
100	[0, 400]	13	34	45	50	50	50	404.273	5.187	1344.969
	[0, 450]	6	18	37	50	50	50	465.197	5.688	1777.360
	[0, 500]	4	24	31	49	50	50	542.885	10.250	2179.515
	[0, 550]	5	17	30	47	50	50	626.389	14.297	2070.078

is to obtain simultaneously the optimal processing sequence and resource allocation, such that all the jobs could be finished no later than their due dates, and the total resource consumption could be minimized. The problem is strongly $\mathcal{NP}$-hard, and a two-layer-structured algorithm based on tabu-search is presented. Experimental results showed for small and media problem instances the algorithm is capable for producing optimal solution, while for large problem instances the algorithm can produce satisfactory solutions in acceptable time.

APPENDIX

A. SOLUTION OF P1.1

Without loss of generality, and for convenience, we assume $\sigma' = \{1, 2, \ldots, n\}$, and jobs have a common release date r and a common due date d. Rewrite P1.1 as follows:

(P2)

$$\begin{aligned} \min \quad & U = \sum_{j=1}^{n} u_j(p_j) \\ \text{s.t.} \quad & \sum_{j=1}^{n} p_j \leq d - r \\ & a_j \leq p_j \leq b_j, \qquad j = 1, 2, \ldots, n \end{aligned}$$

By introducing Lagrangian multiplier $\lambda \geq 0$, P1 can be written as following Lagrangian function:

$$L(p_1, p_2, \ldots, p_n, \lambda) = -\lambda T + \sum_{j=1}^{n} \Big(u_j(p_j) + \lambda p_j \Big), \quad a_j \leq p_j \leq b_j,$$

where $T = d - r$. For fixed $\lambda \geq 0$, the minimization of $L(p_1, p_2, \ldots, p_n, \lambda)$ over p_j yields following dual function:

$$D(\lambda) = -\lambda T + \sum_{j=1}^{n} \min_{a_j \leq p_j \leq b_j} \Big\{ u_j(p_j) + \lambda p_j \Big\}$$

which can be decomposed into a set of independent subproblems:

(P2.1)

$$\min \quad u_j(p_j) + \lambda p_j, \quad a_j \leq p_j \leq b_j, \quad j = 1, 2, \ldots, n$$

Since $u_j(p_j)$ is a convex decreasing function, the solution of P1.1 is

$$\begin{aligned} p_j^* &= a_j, && \text{if } u_j'(a_j) \leq -\lambda && \text{(2a)} \\ p_j^* &= b_j, && \text{if } u_j'(b_j) \geq -\lambda && \text{(2b)} \\ a_j < p_j^* &< b_j, && \text{if } u_j'(p_j^*) = -\lambda && \text{(2c)} \end{aligned}$$

Because $u_j(p_j)$ are all convex functions, $D(\lambda)$ is also a convex function of λ. Thus the optimal solution of P1 equals the maximization of dual function $D(\lambda)$ over $\lambda \geq 0$. State this problem formally as follows:

(P2.2)

$$\max \quad D(\lambda) = -\lambda T + \sum_{j=1}^{n} \Big(u_j(p_j^*) + \lambda p_j^* \Big), \quad \lambda \geq 0$$

Define $\mathscr{J}_1$ the subset that contains jobs satisfying condition(2a), $\mathscr{J}_2$ the subset that contains jobs satisfying condition(2b), and $\mathscr{J}_3$ the subset that contains jobs satisfying condition (2c). Let $D'(\lambda^*) = 0$, we have

$$\sum_{j \in \mathscr{J}_3} p_j^* = T - \sum_{j \in \mathscr{J}_1} a_j - \sum_{j \in \mathscr{J}_2} b_j \tag{3}$$

Eq.3 is the sufficient and necessary condition for λ^*. Calculate $p_j'(a_j)$ and $p_j'(b_j)$ for all $j \in J$. Order them in ascending order and denote them as $DP_1 \leq DP_2 \leq \cdots \leq DP_{2n}$. Under the assumption that $\sum_{j \in \mathscr{J}} a_j \leq d - r \leq \sum_{j \in \mathscr{J}} b_j$, we have $DP_1 \leq -\lambda^* \leq DP_{2n}$. Now assume there exists an $1 \leq l \leq 2n-1$, such that $DP_l \leq \lambda^* \leq DP_{l+1}$. From condition 2a, 2b and 2c it can be seen jobs with $p_j'(a_j) \leq DP_l$ belong to $\mathscr{J}_1$, jobs with $p_j'(b_j) \geq DP_{l+1}$ belong to $\mathscr{J}_2$, while others belong to $\mathscr{J}_3$. Write p_j as the function of λ, that is,

$$p_j^* \quad = \quad \Big(\frac{\omega_j}{k\lambda^*} \Big)^{\frac{k}{k+1}} \qquad j \in \mathscr{J}_3$$

From Eq.3 we have

$$\sum_{j \in \mathscr{J}_3} \Big(\frac{\omega_j}{k\lambda^*} \Big)^{\frac{k}{k+1}} \quad = \quad T - \sum_{j \in \mathscr{J}_1} a_j - \sum_{j \in \mathscr{J}_2} b_j$$

which yields

$$\lambda^* = \frac{1}{k} \left(\frac{\sum_{j \in \mathscr{J}_3} \omega_j^{\frac{k}{k+1}}}{T - \sum_{j \in \mathscr{J}_1} a_j - \sum_{j \in \mathscr{J}_2} b_j} \right)^{\frac{k+1}{k}} \tag{4}$$

Check λ^*. If $DP_l \leq -\lambda^* \leq DP_{l+1}$, we find the optimal value for P1.2. In this case, calculate the optimal processing times as follows:

$$p_j^* \quad = \quad \begin{cases} a_j & j \in \mathscr{J}_1 \\ b_j & j \in \mathscr{J}_2 \\ \dfrac{\omega_j^{\frac{k}{k+1}}}{\sum_{i \in \mathscr{J}_3} \omega_i^{\frac{k}{k+1}}} \Big(d - r - \sum_{i \in \mathscr{J}_1} a_j - \sum_{i \in \mathscr{J}_2} b_j \Big) & j \in \mathscr{J}_3 \end{cases} \tag{5}$$

Otherwise try another l. Repeat this procedure for $l = 1, 2, \ldots, 2n-1$, the optimal solution could be obtained. The detailed steps are given in the follows:

ALGORITHM 3. *1. Calculate $u_1'(a_1), u_1'(b_1), ..., u_n'(b_n)$ and order them in ascending order. For convenience, re-index them as $DU_1, DU_2, \ldots, DU_{2n}$, such that $DU_1 \leq DU_2 \leq \cdots \leq DU_{2n}$. Let $l = 0$.*

2. Let $l = l + 1$, $\mathscr{J}_1 = \mathscr{J}_2 = \emptyset$.

3. For $i = 1, 2, \ldots, l$, if $DU_i = u_j'(b_j)$, insert job j into $\mathscr{J}_2$.

4. For $i = l+1, \ldots, 2n$, if $DU_i = u_j'(a_j)$, insert job j into $\mathscr{J}_1$.

5. Let $\mathscr{J}_3 = \mathscr{J} - \mathscr{J}_1 - \mathscr{J}_2$. Calculate λ^* according to Eq.4. If $DU_l \leq -\lambda^* \leq DU_{l+1}$, calculate p_j^* according to Eq.5 and exit; otherwise go to step 2.

B. REFERENCES

[1] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, March 1997.

[2] Wisner J.D and Siferd S.P. A survey of us manufacturing practices in make-to-order machine shops. *Production and Inventory Management Journal*, (1), 1995.

[3] Chung-Yee Lee and Lei Lei. Multiple-project scheduling with controllable project duration and hard resource constraint: Some solvable cases. *Annals of Operations Research*, 102(1-4):287–307, February 2001.

[4] Graham McMahon and Michael Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Operations Research*, 23(3):475–482, May 1975.

[5] Dvir Shabtay. Convex resource allocation for minimizing the makespan in a single machine with job release dates. *Computers and Operations Research*, 31(9):1481–1489, August 2004.

[6] Dvir Shabtay. Single and two-resource allocation algorithms for minimizing the maximal lateness in a single machine-scheduling problem. *Computers and Operations Research*, 31(8):1303–1315, July 2004.

[7] Dvir Shabtay and Kaspi Moshe. Minimizing the total weighted flow time in a single machine with controllable processing times. *Computers and Operations Research*, 31(13):2279–2289, August 2004.

[8] Dvir Shabtay and George Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, August 2007.