

# A BRANCH-AND-BOUND ALGORITHM FOR SINGLE MACHINE TOTAL TARDINESS PROBLEM WITH RELEASE DATES

Ceyda Oğuz, Eda Yücel.

*Department of Industrial Engineering, Koç University, Rumeli Feneri Yolu, 34450 Sarıyer, İstanbul, Turkey*

This article presents the preliminary results on a new branch-and-bound algorithm proposed for the single machine total tardiness problem with arbitrary release dates. The algorithm relies on dominance properties from the literature as well as a new branching condition. A new lower bounding technique for the problem is also introduced. The performance of the algorithm is compared to the performance of one of the algorithms in the literature. These preliminary results show that the proposed approach is very promising as an efficient and effective exact algorithm for the problem.

Key words: Machine scheduling; Single machine; Total tardiness

## 1. PROBLEM DEFINITION AND LITERATURE REVIEW

In this paper, we consider the following scheduling problem: A set of  $n$  independent jobs has to be processed on a single machine. We assume that the single machine is available from time  $t=0$ , and it can process at most one job at a time. Each job  $j$  is characterized by its release date,  $r_j$ , its processing time,  $p_j$ , and its due date,  $d_j$ , ( $j = 1, \dots, n$ ). Hence, the processing of each job  $j$  can commence only after its release date,  $r_j$ , and it has to be completed at or before its due date,  $d_j$ , ( $j = 1, \dots, n$ ). Furthermore, preemptions are not allowed, that is, a job  $j$  has to be processed on the machine for an uninterrupted period of  $p_j$  units of time ( $j = 1, \dots, n$ ). Our objective is to find a schedule  $S$  which minimizes the total tardiness of all jobs,  $\sum T_j(S)$ , where  $T_j$  is the tardiness of job  $j$  and is defined as  $T_j = \max(C_j(S) - d_j, 0)$ , and  $C_j(S)$  is the completion time of job  $j$  under schedule  $S$  ( $j = 1, \dots, n$ ). Our problem can be denoted by  $1|r_j|\sum T_j$  by using the well-known three-field notation (see, for example, Błażewicz *et al.* (2001), Pinedo (2002)).

The  $1|r_j|\sum T_j$  problem has been shown to be NP-hard by Rinnooy Kan (1976). Erschler *et al.* (1983) have proved a dominance property for scheduling jobs on a single machine with unequal release dates. Later, Chu and Portmann (1989, 1991) have provided a dynamic priority rule for jobs and proposed heuristic solution methods.

Chu (1992) has proved additional dominance properties and proposed a lower bound for the problem. The branch-and-bound algorithm proposed by Chu solves hard problems up to 30 jobs and relatively easy problems (with equal release dates) up to 230 jobs.

Akturk (1995) has considered the same problem and proposed new dominance properties which are used during branching. He has also provided a tighter lower bound than that of Chu. However, the algorithm was not implemented so there are no comparisons of his algorithm with that of Chu (1992).

Most recently, Baptiste *et al.* (2004) have presented a new branch-and-bound procedure based on constraint propagation for the problem. They have introduced new lower bounds and generalized some well-known dominance properties.

In this paper, we propose a new branch-and-bound algorithm which is based on the branching and bounding techniques of Chu and Akturk. We describe this branch-and-bound algorithm in the next section. We also introduce a new branching condition for the algorithm in Section 2.1. We then present our preliminary results on the performance of the algorithm compared to the performance of the branch-and-bound algorithm of Chu in Section 3. We note that our computational experiments are continuing to compare the performance of our algorithm with that of Baptiste *et al.*'s algorithm.

## 2. THE BRANCH-AND-BOUND ALGORITHM

### 2.1 Branching

In this section, we present several branching strategies from the literature. The following theorem is due to Chu (1992) and was also used by Akturk (1995).

**Theorem 1(Chu (1992) - Akturk (1995)):** For any two jobs  $i$  and  $j$  with  $p_i \leq p_j$ , assuming both are ready at the current time  $t^c$ , the following rules minimize the total tardiness:

1. for  $p_i \neq p_j$ ,  
if  $d_i - p_j > t^c$ , use the Earliest Due Date (EDD) rule,  
otherwise, use the Shortest Processing Time (SPT) rule,
2. for  $p_i = p_j$ , use EDD.

The job that can be sequenced at the current time  $t^c$  according to Theorem 1 is called desirable job at time  $t^c$ . Theorem 1 states the sufficient condition for job  $i$  to precede job  $j$  at current time  $t^c$ , but it is not necessary condition for job  $i$  to precede job  $j$  in an optimal sequence. However, the following definition, which is called as the global dominance rule, states the necessary and the sufficient condition for job  $i$  to precede job  $j$  in an optimal sequence (Chu 1992).

**Definition 1:** If job  $i$  dominates job  $j$  for every  $t \geq t^c$ , then this unconditional ordering is called a global dominance for the  $1|r_j|\sum T_j$  problem and denoted by  $i \rightarrow j$ .

For our algorithm and the strategies presented below, the following sets are defined:

- $S(t^c)$  is the set of scheduled jobs until time  $t^c$ .
- $A(t^c)$  is the set of available and unscheduled jobs at time  $t^c$ , i.e.  $A(t^c) = \{i | r_i \leq t^c\} - S(t^c)$ .
- $B(t^c)$  is the set of unavailable and unscheduled jobs at time  $t^c$ , i.e.  $B(t^c) = \{k | r_k > t^c\}$ .
- Job  $j$  is the desirable job at time  $t^c$  due to Theorem 1.
- $D(t^c)$  is the set of unavailable and unscheduled jobs at time  $t^c$  that are more desirable than job  $j$  at their respective release dates,  $r_k$ .

Each node in the branch-and-bound tree represents a job and the path from the root node to a child node defines a partial schedule starting with the job represented by the root node and ending with the job represented by the last node in the path. In addition to the desirable job, which is defined by Theorem 1 above and is a candidate to be the next job in the sequence, there might be other unscheduled jobs to be sequenced as the next job in order to reach an improved sequence. The following corollaries define the conditions for scheduling such a job at the current time. Therefore, these corollaries constitute the

branching conditions for the nodes in the branch-and-bound tree. We note that while Corollaries 1, 2, and 4 are due to Akturk (1995), Corollary 3 describes a new branching strategy that we introduce.

**Corollary 1 (Akturk 1995) (Branching Condition 1):** For the desirable job  $j$  at time  $t^c$ , and for any job  $i \in A(t^c)$ , if  $p_i < p_j$  and  $t^c + p_i > d_i - p_j$ , then the current schedule may be improved by scheduling job  $i$  first.

**Corollary 2 (Akturk 1995) (Branching Condition 2):** For a desirable job  $j$  scheduled at time  $t^c$ , and for a job  $k \in B(t^c)$  that becomes available during the processing of job  $j$ , i.e.  $t^c + p_j > r_k$ , scheduling job  $k$  first may improve the schedule, if  $t^c + \min_{i \in A(t^c)}(p_i) > r_k$ , and either

1.  $p_k \geq p_j$ ,  $d_j - p_k \geq r_k$  and  $d_k \leq d_j$  (due to EDD rule), or
2.  $p_k < p_j$ , and  
either  $d_k - p_j \geq r_k$  and  $d_k \leq d_j$  (due to EDD rule),  
or  $d_k - p_j < r_k$  (due to SPT rule).

**Corollary 3 (Branching Condition 3):** For the desirable job  $j$  at time  $t^c$ , and for any job  $i \in A(t^c)$  that fails the Branching Condition 1, the current schedule may still be improved by scheduling job  $i$  first, if

1.  $p_i \neq p_j$ , and
2.  $t^c + \sum_{i \in A(t^c)} p_i > \min_{k \in B(t^c)}(r_k)$ , and
3. Either  $p_j > \min_{k \in B(t^c)}(p_k)$ ,  
or  $d_j > \min_{k \in B(t^c)}(d_k)$ , and
4. Either  $t^c + p_j < \max_{m \in D(t^c)}(r_m)$ ,  
or  $t^c + p_j \geq \max_{m \in D(t^c)}(r_m)$ , and  $p_i < p_j$ .

**Corollary 4 (Akturk 1995) (Branching Condition 4):** For the desirable job  $j$  at time  $t^c$ , which is also the desirable job compared to job  $k \in B(t^c)$  at time  $r_k$ , the current schedule may still be improved by scheduling job  $k$  first, if

1.  $t^c + \min_{i \in A(t^c)}(p_i) > r_k$ , and
2.  $r_k < \max_{m \in D(t^c)}(r_m)$ , and
3. Either  $p_j > p_k$  and  $t^c + p_j > r_k + p_k$ ,  
or  $p_j < p_k$  and  $t^c + p_j < \max_{m \in D(t^c)}(r_m)$ .

Therefore, at current time in order for a job to be scheduled next, it should either be a desirable job or a job proposed by one of the given branching conditions. In addition to this, if a job is supposed to be scheduled according to Branching Condition 1, 3 or 4 at current time, then it means that the desirable job of the current time cannot be scheduled just after that job. This is stated in the following lemma.

**Lemma 1 (Akturk 1995):** For the desirable job  $j$  at time  $t^c$ , any job  $i \in A(t^c)$  or job  $k \in B(t^c)$  that is scheduled due to either Branching Condition 1 or Branching Condition 3 or Branching Condition 4 cannot immediately precede job  $j$  in an optimal schedule.

## 2.2 Bounding

A lower bound for  $1|r_j|\sum T_j$  can be found by relaxing the problem to its preemptive version. However,  $1|r_j, \text{prmp}|\sum T_j$  is also NP-hard (Chu (1992)). Chu (1992) proposes to use SRPT (Shortest Remaining Processing Time First) rule with due dates ordered in non-decreasing order to generate a lower bound for the total tardiness problem. When this technique is combined with the following theorem, we get a tighter bound.

**Theorem 2 (Akturk (1995)):** For  $S'$ , which is the set of unscheduled jobs at time  $t^c$ , and for the desirable job  $j$  at time  $t^c$ , if either  $j \rightarrow i$  or  $t^c + p_i \leq d_i - p_j$  and  $d_j \leq d_i$  or  $r_i \geq t^c + p_j$  for every job  $i \in S'$ , then job  $j$  will be the first job in the remaining sequence when the preemption is allowed.

The upper bound in our algorithm is calculated by using the heuristic algorithm NDPRTT (Nondelay Schedule and Theorem 1) (Chu and Portmann (1991)). The algorithm runs as follows. At each iteration, the job which can be processed the earliest with the smallest PRTT value (if any tie occurs, the one with the smallest processing time is selected) is scheduled. PRTT value for a job  $i$  at time  $t^c$  is calculated as

$$\text{PRTT}(i, t^c) = b_i(t^c) + \max(b_i(t^c) + p_i, d_i),$$

where  $b_i(t^c)$  is the earliest starting time of job  $i$  at time  $t^c$  ( $b_i(t^c) = \max(t^c, r_i)$ ).

## 2.3 The Branch-and-Bound Algorithm

Our proposed branch-and-bound algorithm can be summarized as follows: Each node is defined by a partial sequence  $P$  of jobs scheduled from the beginning of the schedule. Initially,  $P$  is empty. Each descendant node is obtained by adding, behind the partial sequence  $P$ , a new job  $i$  chosen among the unscheduled jobs. The choice of job  $i$  is performed such that job  $i$  does not violate Lemma 1 and is either a desirable job at the completion time of the partial sequence  $P$ , or satisfies one of the Branching Conditions 1 to 4. The algorithm uses the breadth-first search and is based on only forward sequencing. Although Chu's algorithm based on both forward and backward sequencing, by using the new dominance properties, our algorithm eliminates more nodes in the branch-and-bound tree than that of Chu's.

## 3. COMPUTATIONAL EXPERIMENTS AND RESULTS

We implemented our branch-and-bound algorithm in C++. The computational experiments were performed on Pentium M 1.50 GHz processor with 248 MB of RAM. We generated the data according to the scheme given by Chu (1992). The experiment consists of 120 problems with 20 jobs and 120 problems with 30 jobs. Each problem is generated at random from three uniform distributions of  $r_i$ ,  $p_i$ , and  $d_i - (r_i + p_i)$ . The distribution of  $p_i$  is always between 1 and 10. The distribution of  $r_i$  is between 0 and  $\alpha \times \sum p_i$ . The distribution of  $d_i - (r_i + p_i)$  is between 0 and  $\beta \times \sum p_i$ . Four values for  $\alpha$ , and three values for  $\beta$  were combined to produce 12 problem sets, each containing 10 problems with 20 jobs and 10 problems with 30 jobs.

Tables 1-3 show the results of the problems with 20 jobs, respectively, for the number of nodes considered, the computation time (in seconds), and the mean deviation of lower bounds of nodes considered from the optimum ( $\sum_{i=1}^m (\text{OPT} - \text{LB}_i) / \text{OPT} / m = (\text{OPT} - \sum_{i=1}^m \text{LB}_i / m) / \text{OPT}$ , where  $\text{OPT}$  and  $\text{LB}_i$  are, respectively, the optimum value of the total tardiness for the problem and the lower bound of node  $i$ , and  $m$  is the number of nodes considered) for each problem set. In each table, the minimum, maximum and average values of these measures in the first, the second and the third columns for three different values of  $\beta$  are provided, respectively. In Table 3, we also report in parenthesis the minimum, the maximum and the average tardiness values for the set of 10 problems generated when we obtain the optimum solution. Tables 4-6 are analogous to Tables 1-3 for problems with 30 jobs.

When we analyze the results we observe that the number of nodes considered and consecutively the computation time for the algorithm increase for the difficult cases of the problem ( $\alpha=0.5$  and  $\beta=0.5$ ). This is in line with the observations made by Chu (1992). We note that this difficulty is increased with the increase in the number of jobs. From Tables 3 and 6, we can also notice that the lower bounds are quite successful for these difficult problems as the gap is not much larger compared to the other cases. These

preliminary results show that our algorithm is an alternative to Chu's algorithm (1992) and it efficiently solves problems involving 30 jobs. We remark from the results of Tables 1 and 4 that our algorithm has the potential to be more efficient than that of Chu's since we branch much less in the algorithm. This will be verified in our ongoing computational experiments for larger number of jobs.

## 4. CONCLUSIONS

In this paper we presented a branch-and-bound algorithm for  $1|r_j|\sum T_j$  problem. The algorithm is a combination of ideas from the literature, namely the branching conditions of Akturk (1995) and the bounding idea of Chu (1992), as well as a new branching condition we proposed which improves the performance of the algorithm. Our results indicate that the algorithm can be more efficient than that of Chu's but this should be verified with extended computational experiments. We will also consider the ideas presented in Baptiste *et al.* (2004) to improve our algorithm. We note that the branch-and-bound algorithm developed by Baptiste *et al.* is different from ours in its basic content. Nevertheless, their lower bounding algorithm may be integrated into ours in order to obtain better lower bounds, which might however increase the time complexity.

## REFERENCES

- M.S. Akturk (1995). An exact approach to minimize total tardiness on a single machine subject to release dates. Presented at Combinatorial Optimization Workshop.
- Ph. Baptiste, J. Carlier, and A. Jouglet (2004). A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 158:595-608.
- J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Weglarz (2001). *Scheduling Computer and Manufacturing Processes*, 2<sup>nd</sup> edition, Springer-Verlag, Berlin.
- C. Chu (1992). A branch-and-bound algorithm to minimize total tardiness with unequal release dates. *Naval Research Logistics*, 39:265-283.
- C. Chu and M. Portmann (1989). Minimisation de la somme des retards pour les probl'emes d'ordonnancement `a une machine. *Rapport de Recherche*.
- C. Chu and M. Portmann (1991). Some new efficient methods to solve  $n|1|r_i|\sum t_i$  scheduling problem. *European Journal of Operational Research*, 58:404-413.
- C. M. J. Erschler, G. Fontan and F. Roubellat (1983). A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Operations Research*, 31:114-127.
- M. Pinedo (2002). *Scheduling Theory, Algorithms and Systems*, 2<sup>nd</sup> edition, Prentice Hall, New Jersey.
- A.H.G. Rinnooy Kan (1976). *Machine Sequencing Problem: Classification, Complexity and Computation*, Nijhoff, The Hague.

Table 1. The number of nodes considered (n=20)

$\beta$									
	0.05			0.25			0.50		
$\alpha$	min.	max	avg.	min.	max	avg.	min.	max	avg.
0.0	21	35	25.2	21	143	65.9	21	177	78.5
0.5	21	147	65.0	21	258	157.8	37	689	398.5
1.0	22	125	47.6	21	138	69.4	21	45	30.2
1.5	21	30	25.3	21	60	30.3	21	40	26.2

Table 2. The computation time in seconds (n=20)

$\beta$									
	0.05			0.25			0.50		
$\alpha$	min.	max	avg.	min.	max	avg.	min.	max	avg.
0.0	0	1	0.5	0	5	1.1	0	2	0.9
0.5	0	3	1.0	0	27	5.7	0	165	35.5
1.0	0	25	4.9	1	6	1.9	0	33	15.1
1.5	0	2	0.7	0	2	0.3	0	1	0.3

Table 3. Mean deviation of lower bounds from optimum (n=20)

$\beta$									
	0.05			0.25			0.50		
$\alpha$	min.	max	avg.	min.	max	avg.	min.	max	avg.
0.0	0.062 (422)	0.070 (626)	0.065 (503.2)	0.024 (168)	0.091 (647)	0.042 (353.4)	0.080 (118)	0.120 (373)	0.92 (259.3)
0.5	0.010 (264)	0.060 (483)	0.021 (302.7)	0.000 (162)	0.137 (331)	0.070 (248.2)	0.000 (11)	0.175 (176)	0.068 (74.0)
1.0	0.000 (0)	0.100 (238)	0.070 (92.4)	0.000 (5)	0.137 (387)	0.070 (208.1)	0.000 (32)	0.175 (266)	0.164 (128.7)
1.5	0.060 (257)	0.096 (1040)	0.066 (470.5)	0.058 (221)	0.127 1050	0.090 (489.0)	0.076 (20)	0.158 (477)	0.099 (264.5)

Table 4. The number of nodes considered (n=30)

$\beta$									
	0.05			0.25			0.50		
$\alpha$	min.	max	avg.	min.	max	avg.	min.	max	avg.
0.0	31	164	54.7	32	348	143.7	61	581	207.9
0.5	33	271	149.0	76	13668	3587.8	32	57321	25123.2
1.0	94	2482	1182.6	197	1101	490.8	95	4511	1329.8
1.5	84	1839	717.4	31	1199	359.4	31	1028	390.0

Table 5. The computation time in seconds (n=30)

$\beta$									
$\alpha$	0.05			0.25			0.50		
	min.	max	avg.	min.	max	avg.	min.	max	avg.
0.0	0	2	0.9	0	9	1.8	0	5	2.1
0.5	0	3	1.4	1	99	48.4	1	834	251.0
1.0	2	23	10.8	1	8	3.6	1	29	9.2
1.5	1	15	6.2	0	12	3.6	0	8	3.2

Table 6. Mean deviation of lower bounds from optimum (n=30)

$\beta$									
$\alpha$	0.05			0.25			0.50		
0.0	0.045 (797)	0.064 (1484)	0.058 (1193.5)	0.044 (669)	0.072 (1248)	0.052 (914.9)	0.051 (379)	0.088 (735)	0.068 (581.3)
0.5	0.017 (533)	0.052 (958)	0.034 (837.0)	0.000 (265)	0.153 (512)	0.097 (358.6)	0.000 (12)	0.080 (257)	0.033 (127.3)
1.0	0.010 (108)	0.070 (879)	0.051 (524.2)	0.000 (28)	0.110 (426)	0.089 (184.4)	0.000 (11)	0.182 (340)	0.171 (135.2)
1.5	0.052 (524)	0.112 (2030)	0.07 (1199.2)	0.049 (48)	0.167 (619)	0.113 (332.8)	0.065 (335)	0.098 (1624)	0.076 (670.0)