On the N-Job, One-Machine, Sequence-Independent Scheduling Problem with Tardiness Penalties: A Branch-Bound Solution

Joel Shwimer

Management Science (1972), Vol. 18, No. 6, p142-159

2016년도 여름방학 Lab Seminar

2016. 07. 26

Jeong Hoon Shin
## Contents

- **The Problem**
- **Other Research**
- **Branch-Bound Algorithm**
- **Discussion of the Algorithm**
- **Sample Problem**
- **Computational Results for Final Algorithm**
- **Extensions and Future Work**
The Problem
The problem of this paper

- The problem to be discussed in this paper is the scheduling of \( n \) jobs on a single machine where the goal is to minimize the total penalty costs resulting from the tardiness of the jobs.
  - \( i \) : index of job
  - \( t_i \) : processing time
  - \( d_i \) : due date
  - \( P_i \) : penalty function
  - \( C_i \) : completion time

\[
P_i(d_i, C_i) = p_i \text{max}(0, C_i - d_i)
\]

- Zero ready time, processing times of all jobs are independent
Other Research
Researches (1)

- McNaughton [11]
  - proved: an optimal solution exists in which no job is split [11, p. 4]. → Permutation
  - Developed methods without full enumeration (McNaughton [11])

- Schild and Fredman [13].
  - generalize some of McNaughton’s results (Not optimal solution).
  - also develop a DP model (nonlinear penalty function case)

  - computationally infeasible (20 jobs) → 15 job problem was solved on an IBM 7090

- Lawler [9] presents a LP formulation requiring $n + 2T$ constraints
  (T: the total required processing time for all $n$ jobs).
  - A modified simplex algorithm required.
  - formulate a much smaller MILP formulation
    ($n^2$ constraints, $3n$ continuous variables, and $n(n - 1)/2$ zero-one variables)
Researches [2]

  - Objective: shortest route (cost-minimization of DP)
  - $S_k$: a subset of $k$ jobs not yet scheduled
  - $S_{k-1}^j$: subset of jobs still unscheduled after job $j$ is scheduled last among the jobs in $S_k$.
  - Backward scheduling
    - representing the set $S_n$ by a node
    - then joining it by $n$ arcs to $n$ nodes representing the sets $S_{n-1}^j$.
    - process continues until the nodes with $S_0$ (the empty set) are reached.
      - the node with $S_k$ is joined by $k$ arcs to $k$ nodes (possible sub sets $S_{k-1}^j$)
  - Length ($= p_j \max(0, T_k - d_j); T_k = \sum_{i \in S_k} t_i$)
    : the penalty cost incurred by scheduling job $j$ last among the jobs in $S_k$.
    = the length of the arc from the node $S_k$ to the node $S_{k-1}^j$.
  - shows how branch-bound methods substantially reduce computation time (vs DP)
  - outlines a six-step branch-bound algorithm (not coded)

- Emmons [6]
  - proves several theorems.
  - then devises a branch-bound algorithm (not coded)
Branch-Bound Algorithm
B&B algorithm(1)

- Shortest route formulation of Elmaghraby [3] is the underlying structure for this algorithm.

Step 0. initialize

- 0.a. Order the jobs by increasing $d_i$, (breaking ties : larger $p_i$, than smaller $t_i$)
- 0.b. Test each pair of jobs $i$, $j$ ($i < j$) by:
  - THEOREM A. If $p_i \geq p_j$, $d_i \leq d_j$, and $t_i \leq t_j$, then save the information that we will only consider schedules with job $i$ preceding job $j$.
- 0.c. Create root node, and put it on node list. Perform the bookkeeping initialization.
- 0.d. Search for a good initial UB by considering the n schedules formed by Step 0.a. except for the last scheduled job $i(= 1, \cdots, n)$. Save the best of these n schedules and its total penalty cost as the current "best".

Step 1. Choose a Node for Expansion

- 1.a. Consider the last node placed on the node list. If it has been already expanded, or if its value is not less than the current "best", drop it from the node list, and restart Step 1.
- 1.b. When the node list is empty, stop.(∴ the current best schedule proved optimal)
- 1.c. Otherwise expand the last node on the node list.
B&B algorithm(2)

Step 2. Expand Chosen Node (node k)

2.a. If $S_k$ consists of only one job, $m$, find the total penalty cost of the schedule formed by placing job $m$ first, by adding $p_m \max (0, t_m - d_m)$ to the incurred cost at node $k$. If this new cost is $<$ the current "best", save this schedule and its total penalty cost as the new "best". Otherwise the old "best" remains. In either case, drop node $k$ from the node list and go to Step 1.

2.b. If $S_k$ consists of more than one job, test to see if the total processing time of all jobs in $S_k$ is $\leq$ the due date of the job, $j$, with the latest due date among jobs in $S_k$. If this condition holds, only consider creating one successor node to node $k$ by scheduling job $j$ last among the jobs in $S_k$. Then go to 2.d.

2.c. Otherwise, we must consider creating one successor node to node $k$ for each job in $S_k$ that can be scheduled last among the jobs in $S_k$. If some two jobs in $S_k$ are satisfied Theorem A, we need not consider scheduling the precede job of the two jobs last among the jobs in $S_k$. 
B&B algorithm(3)

- Step 2. Expand Chosen Node(node k)
  - 2.d. Evaluate each potential successor node to node k (i.e. the scheduling of job last among the jobs in $S_k$) in the following fashion:

  \[ \text{VALUE} = \text{INCURRED COST AT NODE } k + p_j \max(0, T_k - d_j) \]
  \[ + \min_{i \in S_k; i \neq j} \{ p_i \max(0, T_k - t_j - d_i) \} \]
  \[ + \max_{h \in S_k; h \neq j; h \neq i} (\text{tardiness of job } h \text{ in a schedule formed by ordering the jobs } h \in S_k; h \neq j; h \neq i \text{ as in Step 0.a.)} \cdot \min_{g \in S_k; g \neq j; g \neq i} (p_g) \] 

  - $j$: index of the job being scheduled to create this node as a successor to node k
  - $S_k$: the set of unscheduled jobs at node k
  - $T_k(=\Sigma_{i \in S_k} t_i)$: the processing time of all jobs unscheduled at node k

  - If the value of the node created by scheduling job j is $\geq$ the current "best", this node can be discarded without ever being added to the node list.
  - Otherwise, the incurred cost for this new node should be calculated as follows:

  \[ \text{INCURRED COST} = \text{INCURRED COST AT NODE } k + p_j \max(0, T_k - d_j) \]

  - 2.e. Now order all of the nodes created in Step 2.b or 2.c but not discarded in Step 2.d in order of decreasing VALUE and place them on the node list as nodes $k+1, k+2 \ldots$.
  - 2.f. Mark node k as expanded, and go to Step 1.
Discussion of the Algorithm
References & extensions.

- The algorithm outlined above.
  - creates the same basic tree as in Elmaghraby [3]
  - the node to be expanded is always on the lowest level on the tree

- The partial schedule.
  - having the minimum lower bound on the total penalty cost (VALUE)
  - Is chosen to be the one next evaluated, from among those on the tree having the most jobs already scheduled.

- Step 2.b.
  - is the test indicated in a lemma proven by Elmaghraby [3, p. 105].

- In step 2.c.
  - it is not always necessary to consider all still unscheduled jobs ← Theorem A (Step 0.b)
  - Theorem A is an extension of Emmon's theorem [6, p. 703].
    - the case where the penalty function for each job can be different.
Lower Bound

- the lower bound used in Step 2.d.

To find this LB

\[
VALUE = \text{INCURRED COST AT NODE } k + p_j \max (0, T_k - d_j) \\
+ \min_{i \in S_k; i \neq j} \{ p_i \max (0, T_k - t_j - d_i) \} 
\]

Elmaghraby uses:

This LB(↑ VALUE) includes the smallest possible penalty cost incurred by scheduling some job, i, last among the remaining unscheduled jobs after job j is scheduled last among the jobs in \( S_k \). ← a one-step, look-ahead procedure

Step 2.d. goes further: additional consideration of other jobs in \( S_k \) may still remain unscheduled after schedule some job, i, last.

\[
VALUE = \text{INCURRED COST AT NODE } k + p_j \max (0, T_k - d_j) \\
+ \min_{i \in S_k; i \neq j} \{ p_i \max (0, T_k - t_j - d_i) \} \\
+ \max_{h \in S_k; h \neq j; h \neq i} (\text{tardiness of job } h \text{ in a schedule formed by ordering the jobs } h \in \ S_k; h \neq j; h \neq i \text{ as in Step 0.a.)}) \cdot \left[ \min_{g \in S_k; g \neq j; g \neq i} (p_g) \right] 
\]

Taking the minimum of the sum of these two additional penalty costs (over all jobs i in \( S_k \) excluding job j) gives us a tighter bound on penalty cost than was obtained in the one-step, look-ahead method.
Storage requirements

- Storage requirements → Low
  - Max \((n^2 + n)/2\) nodes will maintained at any given time\((n : \# \text{ of jobs})\)
    → exceedingly unlikely case

- To fully specify a node, at most the following seven items are needed:
  1. a node identification \# (unique for each node that is currently being maintained);
  2. the identification \# of its parent node;
  3. the job scheduled to create this node from its parent node;
  4. the total processing time required for all unscheduled jobs;
  5. the cost incurred by those jobs already scheduled upon reaching this node;
  6. a lower bound (VALUE) on the cost of all solutions passing through this node;
  7. a marker to indicate the status of this node (i.e. expanded or not).

- Storage requirement could be reduced by 50 %
  - (4) & (5) : could be calculated by tracing up the tree
  - (7) : can be stored simply negating (2) or (3) to indicate an expanded node.
Sample Problem
Sample Problem

- 10 jobs sample problem (10! Permutations)
  - Problem data (Min total penalty cost of 27)

<table>
<thead>
<tr>
<th>Job</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$d_j$</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>11</td>
<td>15</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>$p_j$</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

- By Theorem A.
  - Job 1 precedes job 4; job 2 precedes job 7; job 3 precedes jobs 4, 7, 9;
    job 5 precedes jobs 7, 9; job 8 precedes job 9.
  - reduces feasible schedule to 127,140 (4% of 10! = 3,628,800).

- Initial UB(step 0.d.)
  - From best of ten schedules → total penalty cost of 29.
  - no nodes need be created which have a VALUE ≥ 29
A total of 32 nodes have been created so far. (\(=16+1 + 14+1\))

schedule, 
\((1, 2, 3, 5, 4, 6, 7, 8, 9, 10)\)
16 nodes +1(final)

schedule, 
\((1, 2, 3, 5, 4, 6, 8, 9, 7, 10)\)
14 nodes (4 additional nodes : 6, 7, 9, 10) 
+1(final)
Entire tree of 36 nodes (final: X)

- The entire tree was never held in the computer at one time.
  - The generation and discarding procedure required a maximum of 16 nodes to be stored in the computer at any one time.
- Max 55 (= (10^2 + 10)/2) nodes
  16/55 = 29.09%
Computational Results for Final Algorithm
Data generation

- Random test instance generator.
  - Processing time $t_j \sim U(a, b)$ [uniform distribution] or $Tr(a, b)$ [Triangular distribution]
  - Due date $d_j \sim U(t_j, 5.5n)$; (n : # of jobs; 5.5 : expected mean of $t_j$)
  - penalty function coefficients $p_j \sim U(1, 5)$
  - All of the generated data were kept as integers.
  - Create 60 test problems in each of the following four categories (240 problems)
    - 10 job; $t_j \sim U(a, b)$
    - 10 job; $t_j \sim Tr(a, b)$
    - 20 job; $t_j \sim U(a, b)$
    - 20 job; $t_j \sim Tr(a, b)$

- Computation environments
  - IBM 360/65 computer.
  - Fortran IV, H-level (optimization level 2)
Computational Results

Problems having processing times drawn from a uniform distribution are somewhat easier to solve.

- Median solution time for all
  - 10 job problems: 0.07 sec
  - 20 job problems: 0.29 sec
  - 30 job problems: 4.13 sec

- 45% of 10 job problems and
  - 15% of 20 job problems: optimal schedule was found in Step 0.d. (search for good initial UB)

- 80% of 10 job problems and
  - 45% of 20 job problems: opt sol found in step 0.d. or first complete schedule reached in tree.

- 20 job problem, less than 60% (on the average) of the solution time was spent in finding the optimum
  → the problems for which it might be necessary to stop the algorithm short of completion, and, these are also the problems in which the optimal solutions are found sooner.
Extensions and Future Work
Extensions

- The inclusion of pairwise precedence constraints.
  - to insure that Theorem A does not "create" any pseudo-precedence constraints which are inconsistent with those specified in the problem.

- allow for nonlinear penalty functions

  THEOREM A (GENERALIZED). If \( t_i \leq t_j, d_i \leq d_j, \) and \( P_i(x) \geq P_j(x) \) for all \( x, 0 \leq x \leq T - d_i, \) then there exists an optimal solution in which job \( i \) precedes job \( j \) (where \( P_i(x) \) is the penalty function of tardiness for job \( i \), and \( T \) is the total processing time required by all \( n \) jobs).

  INCURRED COST = INCURRED COST AT NODE \( k \) + \( P_j(\max(0, T_k - d_j)) \).

  VALUE = INCURRED COST AT NODE \( k \) + \( P_j(\max(0, T_k - d_j)) \)
  + \( \min_{i \in S_k; i \neq j} [P_j(\max(0, T_k - t_j - d_i))] \)
  + \( \min_{g \in S_k; g \neq j; g \neq i} P_g(\max \{ \text{tardiness of job } h \text{ in a schedule formed by ordering the jobs in } S_k, h \neq j; h \neq i \text{ as in Step 0.a.} \}) \)
Future works

- improve, the branch-bound algorithm
  - obtaining a better initial problem upper bound
- Develop a heuristic to generate complete solutions at nodes other than the root node.
  - such a technique would be to increase computation time.
- The use of dominance relations between nodes, as suggested by Elmaghraby [3].
  - reduce the necessary amount of searching in the lower levels of the tree.
  - As problems become larger, the potential gains become greater, but so do the potential storage and computation time requirements.
감사합니다