Numerical analysis of $1\|\sum T_j$ using exact algorithm and heuristics

Huiwen Ma
Haosong Wang
Wenzhu Yu
Introduction

• Problem: $1/\sum T_j$
• NP-hardness
• Paper review: algorithms solving this problem
  • Simple Heuristics:
    • SPT, EDD, and Modified Due Date (MDD)
  • Other Heuristics:
    • Wilkerson, L.J. and Irwin (W-I)
    • Fry et al., Adjacent Pairwise Interchange (API)
    • Holsenback, J.E. and Russell, Net Benefit of Relocation (NBR)
    • Panwalkar, Smith, and Koulamas, (P-S-K)
    • …
• Exact Algorithm:
  • The Pseudo-polynomial Time Algorithm
Pseudo-polynomial Time Algorithm

• I jobs, k = the job with the longest processing time.
• The subsets J(j, l, k): all jobs in the set {j, ..., l} with a processing time \( \leq p_k \).
• \( V(J(j, l, k), t) \): the total tardiness of \( J(j, l, k) \) in an optimal sequence that starts at time \( t \).

**Algorithm**:

• Initial conditions: \( V(\emptyset, t) = 0; \quad V\{j\}, t) = \max(0, t + p_j - d_j) \)
• Recursive relation:
\[
V(J(j, l, k), t) = \min_{\delta} (V(J(j, k' + \delta, k'), t) + \max(0, C_{k'}(\delta) - d_{k'}) + V(J(k' + \delta + 1, l, k'), C_{k'}(\delta)))
\]
• Where \( k' \) is such that, \( p_{k'} = \max(p_j | j' \in J(j, l, k)) \)
• Optimal value function is \( V\{1, ..., n\}, 0 \).

* cited from Professor Clifford Stein's Lecture Notes
MDD Rule

• Index: $l_i(t) = (t + p_i - d_i)^+ + d_i$
• Select the next job with the smallest index value for processing.
PSK Algorithm

- Ordered set $U(1,2,3,...,n)$ in the SPT order.
- $S$ = the set of scheduled jobs
- $p_j, d_j, c = \sum_{i \in S} p_i$.

**Step 1.** If $U$ contains only 1 job, schedule it in the last position in $S$ and go to Step 9. Otherwise label the first job of $U$ as the active job $i$.

**Step 2.** If $c+p_i \geq d_i$, go to Step 8.

**Step 3.** Select the next job of $U$ and label it as job $j$.

**Step 4.** If $d_i \leq c + p_j$, go to Step 8.

**Step 5.** If $d_i \leq d_j$, go to Step 7.

**Step 6.** Job $j$ becomes the active job $i$. If it is the last job in $U$, go to Step 8. Otherwise, go to Step 2.

**Step 7.** If $j$ is the last job in $U$, go to Step 8. Otherwise, go to Step 3.

**Step 8.** Remove job $i$ from $U$ and put it in the last position in $S$. Set $c = c + p_i$ and go to Step 1.

**Step 9.** Calculate total tardiness for the sequence and stop the algorithm.
P-S-K heuristic

• Other Heuristics
  • Wilkerson-Irvin works better when EDD is optimal
  • API uses 9 switching strategies and generates 9 sequences
  • Holsenback-Russell starts with MDD, and uses net benefit of relocation

• Panwalkar et al. concluded that the P-S-K algorithm performs better than the W-I, H-R, and API heuristics for a wide range of problems, especially when due dates become tight.
Algorithm Properties

- **When works the best**
  - **MDD**
    - At most 1 job has positive tardiness or all processing times are equal
    - MDD reduces to the EDD rule when processing times are equal, and to the SPT rule when due dates are equal.
  - **PSK**
    - All jobs have positive tardiness or all due dates are equal, similar to SPT
  - *Pseudo-polynomial time algorithm*
    - Optimal

- **Cost-worst case scenario**
  - **MDD**
    - $O(n \log n)$
  - **PSK**
    - Performs better than the pseudo-polynomial, but worse than MDD
  - *Pseudo-polynomial time algorithm*
    - $O[n^4 \times \Sigma p_j]$
Computation Setup

- Implemented in Java
- Algorithms
  - MDD
  - PSK
  - Pseudo-polynomial time algorithm (exact algorithm)
    - Using a recursive function to implement the dynamic programming routine.
- Sorting is implemented by a standard java.collections.sort method (essentially a mergesort with complexity fewer than $O(n\log n)$)
Computation Setup

- Implemented in *Java*
- Algorithms
  - MDD
  - PSK
  - Pseudo-polynomial time algorithm (exact algorithm)
    - Using a *recursive function* to implement the dynamic programming routine.
- Sorting is implemented by a standard *java.collections.sort* method (essentially a *mergesort* with complexity fewer than $O(n\log n)$)

(randomly generated instances)

1. MDD
2. PSK
3. Pseudo-polynomial

- Schedule decision
- Total Tardiness
- CPU Time (milliseconds)
Random Instances Generation

- Generated by the method suggested by *Potts and Van Wassenhove*.
- Two instance characteristics control factors
  - RDD (Range of Due Date) \{0.2, 0.4, 0.6, 0.8, 1.0\}
    - Controls the variance of different due date
  - TF (Tardiness Factor) \{0.2, 0.4, 0.6, 0.8, 1.0\}
    - Controls the tightness of schedules

Let \( p_{\text{max}} \) be the maximum number of jobs. The following steps are used to generate each job's due date:

1. Compute the total processing time \( P = \sum_{i=1}^{n} p_i \).
2. Select values of RDD and TF from the set \{0.2, 0.4, 0.6, 0.8, 1.0\}.
3. Select an integer due date \( d_i \) from the uniform distribution \([P(1-\text{TF}-1/2\times\text{RDD}), P(1-\text{TF}+1/2\times\text{RDD})]\).

Due date uniformly distributed
Results

• Performance: total tardiness and computational time
• Total tardiness of pseudo-polynomial algorithm is the benchmark of accuracy.
• Tardiness error = \( \frac{\text{tardiness} - \text{optimal tardiness}}{\text{optimal tardiness}} \)
Results

• Performance: total tardiness and computational time
• Total tardiness of pseudo-polynomial algorithm is the benchmark of accuracy.
• Tardiness error = \( \frac{\text{tardiness} - \text{optimal tardiness}}{\text{optimal tardiness}} \)

<table>
<thead>
<tr>
<th></th>
<th>MDD</th>
<th>PSK</th>
<th>Pseudo-polynomial Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tardiness error</td>
<td>time</td>
<td>tardiness error</td>
</tr>
<tr>
<td>10 jobs</td>
<td>0.3783</td>
<td>0.008</td>
<td>0.2728</td>
</tr>
<tr>
<td>15 jobs</td>
<td>0.4628</td>
<td>0.008</td>
<td>0.3330</td>
</tr>
<tr>
<td>20 jobs</td>
<td>0.4837</td>
<td>0.024</td>
<td>0.3362</td>
</tr>
<tr>
<td>25 jobs</td>
<td>0.5024</td>
<td>0.052</td>
<td>0.3505</td>
</tr>
<tr>
<td>Mean</td>
<td>0.45687</td>
<td>0.023</td>
<td>0.32317</td>
</tr>
</tbody>
</table>

• Tardiness MDD>PSK>Pseudo-polynomial time Algorithm
• Time MDD<PSK<Pseudo-polynomial time Algorithm
Results (cont’d)

- **Tardiness error** with respect to **number of Jobs**
  - Insight: As # jobs increases, error increases, difference between two algorithms also increases
  - Reason: Problem becomes combinatorial complex when # of job increases. MDD starts to lose effectiveness.

![Bar chart showing MDD and PSK tardiness error for 10, 15, 20, and 25 jobs.](chart.png)
Results (cont’d)

• **Computational time** with respect to **number of jobs**

  • Insight: CPU time of Pseudo-polynomial time algorithm is highly sensitive to number of jobs, which demonstrates the NP-hardness of the problem to solve to optimality.
Results (cont’d)

- **Tardiness error** with respect to **Tightness of instances (TF)**
  - **10 jobs case**
  - **Insight:** As TF increases, error increases, difference between two algorithms also increases. PSK starts to become much more effective compared to MDD as instances become tighter in due date.
Conclusions

• Generally, we get
  - Tardiness  MDD > PSK > Pseudopolynomial Algorithm
  - Time  MDD < PSK < Pseudopolynomial Algorithm
• The difference in accuracy between MDD and PSK will become larger as the number of jobs increase.
• Both the two heuristics become less accurate when the number of jobs is large.
• PSK will perform much better than MDD in accuracy especially when the tardiness factor is large.
• Overall, PSK has an average 32.3% error rate; MDD has an average 45.6% error rate. Neither does these two heuristics have very good performance. While considering their efficiencies in computational efforts, PSK would still be applicable in real applications.
Thank you