Merging and Memorization in search trees: on the exact solution of scheduling problems

Lei Shang (PhD Student)
Pr. Vincent T’Kindt (Director)

Université François Rabelais Tours
Laboratoire d’Informatique (EA 6300)
Equipe ROOT (CNRS ERL 6305)
Outline

1. Problem: $1\|\Sigma T_i$
2. Branch & Merge (theoretical guarantee)
3. Memorization (practical efficiency)
4. Extension: Branch & Memorize framework on sequencing problems
Problem: $1||\sum T_i$

- Jobset $S$, single machine, $p_j =$ processing time, $d_j =$ due date
- Objective: minimize the total tardiness $\sum_j \max(0, C_j - d_j)$
- NP-hard (ordinary sense)
- In theory (complexity):
  - Brute force $O(n!)$
  - Dynamic programming: $O^*(2^n)$ in time and space
  - Divide & Conquer: $O^*(4^n)$, polynomial space (Gurevich et al., 1987)
  - Branch & Reduce: $O^*(3^n)$, polynomial space (F. D. Croce et al, 2015)
  - Branch & Merge => $O^*((2 + \epsilon)^n)$ in time and polynomial space

- In practice:
  - The B&B of Szwarc et al. => 500 jobs in 2001. (900 jobs today!)
  - Memorization => 1200 jobs.
In Theory

Objective

- **Exact** algorithms with **worst-case** running time/space guarantee ($O^*(c^n)$, with $c$ a constant as small as possible)

Notation

- LPT (Longest Processing Time first) job sequence: $(1, 2, \ldots, n)$
- EDD (Earliest Due Date first) job sequence: $(e_1, e_2, \ldots, e_n)$
In Theory

Lawler’s Property (1977)

- Let job $1 = e_h$, then job 1 can only be set in position $s \geq h$
- Jobs preceding 1 are: $B_1 = \{e_1, e_2, \ldots, e_{h-1}, e_{h+1}, \ldots, e_s\}$
- Jobs following 1 are: $A_1 = \{e_{s+1}, e_{s+2}, \ldots, e_n\}$

$\Rightarrow$ Worst case: LPT=EDD

- => Worst case: LPT=EDD
Branch & Reduce

- LPT=EDD
- Depth-First

\[ T(n) \leq 2T(n-1) + 2T(n-2) + \cdots + 2T(1) \Rightarrow T(n) = O(3^n) \]
Branch & Reduce

- LPT=EDD
- Notation
Branch & Reduce: observations

- LPT=EDD
- Depth-First

Some sub-problems are solved repeatedly!
Branch & Reduce: observations

- LPT=EDD
- Depth-First

Some sub-problems are solved repeatedly!
Branch & Merge (left)

**Idea:** merge nodes based on the fixed part

- Solve sub-problem \(\{2,3\}\) (e.g. 32)
- Compare 321 and 132
- Cannot apply on all pairs
Branch & Merge (left)

- Idea: merge identical nodes based on the fixed part
  - On first k nodes, k is a constant
  - Why not just cut?

\[
P : \{1, ..., n\}
\]

\[
P_1 \quad P_2 \quad \ldots \quad P_k \quad \ldots \quad P_n
\]

\[
P_{\sigma_{1,2}} \quad \ldots \quad P_{\sigma_{1,k}} \quad \ldots \quad P_{1,n}
\]

\[
P_1 : 1\{2, ..., n\}
P_2 : 21\{3, ..., n\}
P_k : 2..k\{1\{k + 1..n\}\}
P_{\sigma_{1,2}} : \text{BEST}(12, 21)\{3..n\}
P_{\sigma_{1,k}} : \text{BEST}(2..k\{1, 1\{3..k\}\}2\{k + 1..n\})
\]
Branch & Merge (left)

Idea: merge identical nodes based on the fixed part
- On first k nodes, k is a constant
- Transitivity
- General case
Merging and Memorization

Branch & Merge (right)

More complex...

$k = 3$

\[
\begin{aligned}
&\{3, \ldots, n - 2\} 2\{n - 1, n\} \\
&\{3, \ldots, n - 2\} 2\{n - 1\} 1\{n\} \\
&\{3, \ldots, n - 2\} 2\{n - 1, n\} 1
\end{aligned}
\]
Branch & Merge

Recurrence: $T(n) \leq 2T(n - 1) + (5k - 1)T(n - k - 1) + O(p(n))$
Branch & Merge

- $T(n)$ converges to $O^*(2^n)$. $T(n) = O^*(2.0367^n)$ when $k = 10$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$T(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$\mathcal{O}^*(2.3065^n)$</td>
</tr>
<tr>
<td>10</td>
<td>$\mathcal{O}^*(2.0367^n)$</td>
</tr>
<tr>
<td>15</td>
<td>$\mathcal{O}^*(2.0022^n)$</td>
</tr>
<tr>
<td>20</td>
<td>$\mathcal{O}^*(2.0001^n)$</td>
</tr>
</tbody>
</table>
Summary 1

- Branch & Merge in $\sim O^*(2^n)$ time and polynomial space
- Can be generalized to other problems: branch smartly
- Work done together with:
  - Federico Della Croce
  - Vincent T’Kindt
  - Michele Garraffa
In Practice

- **BB2001:** Szwarc et al. 2001
  - Solved 500 jobs in 2001 (900 jobs today!)
  - **Split:** decompose by precedence relations
  - **PosElim:** eliminates bad branching positions
  - **Memorization:** avoids solving a problem twice by storing its solution (basically merging without moving nodes)

- **Without Split, PosElim**
  - Branch & Merge is clearly more efficient than Branch & Reduce

- **With Split, PosElim**
  - Split & PosElim: break the structure of merging
  - **Memorization** is more practical, even though theoretically exponential space.
In Practice: Memorization

« Never solve a problem twice »
In Practice: Memorization

« Never solve a problem twice »
In Practice: Memorization

« Never solve a problem twice »
The power of Memorization

- BB2001 of Szwarc et al. has no LB procedure:
  - Paradox (Szwarc et al. 2001): removal of LB evaluation drastically accelerate the solution.
  - \( \Rightarrow \) cut a sub-problem many times by computing LB is slower than solving it once and memorize the solution

- Can be further boosted!
Enhanced Paradox

- Enhanced Paradox (our work)
  - Removing **Split** from BB2001 drastically accelerate the solution
  - **Split**: decompose the problem by precedence relations

<table>
<thead>
<tr>
<th>TMin (s)</th>
<th>TAvg (s)</th>
<th>Tmax (s)</th>
<th>#Nodes</th>
<th>#Hit</th>
<th>#SolMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>192.81</td>
<td>2963.0</td>
<td>880268</td>
<td>227203</td>
<td>111175</td>
</tr>
<tr>
<td>0.0</td>
<td>8.0</td>
<td>114.0</td>
<td>3053648</td>
<td>899031</td>
<td>1262895</td>
</tr>
</tbody>
</table>

**Table**: Results for instances of size 700

- But...the memory is filled quickly (solve up to 700 jobs)
Memory Analysis

Are all memorized solutions useful?
Memory Analysis

- Are all memorized solutions useful?
Memory Cleaning Strategies

- LUFO (Least Used First Out)
  - Attach a counter \( \text{nbUsed} \) to each solution
  - When a solution is used: \( \text{nbUsed} = \text{nbUsed} + 1 \)
  - Memory full: \( \text{nbUsed} = \text{nbUsed} - 1 \) for all solution, remove a solution if its \( \text{nbUsed} < 0 \)

- Also tested:
  - FIFO (First In First Out)
  - BEFO (Biggest Entry First Out)

<table>
<thead>
<tr>
<th></th>
<th>Tmin</th>
<th>TAvg</th>
<th>Tmax</th>
<th>#Nodes</th>
<th>SizeMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO-800</td>
<td>0.0</td>
<td>60.0</td>
<td>3144.0</td>
<td>16161758</td>
<td>1727397</td>
</tr>
<tr>
<td>BEFO-800</td>
<td>0.0</td>
<td>59.0</td>
<td>4828.0</td>
<td>6356245</td>
<td>2006948</td>
</tr>
<tr>
<td>LUFO-800</td>
<td>0.0</td>
<td>19.0</td>
<td>275.0</td>
<td>5408511</td>
<td>1354477</td>
</tr>
<tr>
<td>LUFO-1200</td>
<td>0.0</td>
<td>192.0</td>
<td>3763.0</td>
<td>28223765</td>
<td>1424612</td>
</tr>
</tbody>
</table>
Summary 2

- An enhanced paradox for $1||\sum T_i$
- An efficient memory cleaning strategy: LUFO
- Solve instances with up to 1200 jobs (from 900)

Work done together with:
- Federico Della Croce
- Vincent T’Kindt
Further: a Branch & Memorize framework

- We have witnessed the power of Memorization
- Can be applied on other problems?
- Three problems are considered: $1|r_i|\sum C_i$, $1|\tilde{d}|\sum w_i C_i$, $F2||\sum C_i$
- Treated in T’Kindt et al. (2004).
  - Different search strategies are revisited
  - The so-called DP property is implemented
    - Consider two nodes: $123\{4,\ldots,n\}$ vs $132\{4,\ldots,n\}$
    - If 123 dominates 132, then the second node should be cut
    - Use memory to store the prefixed part
Further: a Branch & Memorize framework

A framework: different ways of doing Memorization:

- Solution Memorization \((1||\sum T_i)\)
  - Depth-first

![Diagram of tree structure](image)
A framework: different ways of doing Memorization:

- **Passive Node Memorization**
  - Memorize the current best solution for the fixed part given by branching
  - Used for cutting
  - Consider $\sigma'$ dominates $\sigma$ and $\sigma''$ (breadth-first)
Further: a Branch & Memorize framework

Different ways of doing Memorization:

- **Predictive Node Memorization**
  - Memorize the current best solution for the fixed part given by active search
  - Passive Node Memo + Local search
    - Dominance Rules Relying on Scheduled Jobs (Jouglet et al. 2004)
  - Used for cutting
  - Consider $\pi$ dominates $\sigma$ and $\sigma''$

Figure 3: Predictive node memorization
Choose the right Memo scheme

Given a branching algorithm, choose a Memorization scheme

- Branching scheme
- Search strategy
- Other properties: whether « Decomposable »...
Choose the right Memo scheme

Figure 4: Decision tree for choosing the memorization scheme

A: the problem is decomposable
B: no context dependent dominance conditions
C: concordance property verified
D: solution memorization
E: passive/predictive node memorization
F: passive node memorization
G: check() applied on active nodes only
H: check() applied on explored nodes only
I: suggested scheme when none is dominant
Further: a Branch & Memorize framework

* The evidence of the power of memorization

<table>
<thead>
<tr>
<th>Problem</th>
<th>Largest instances solved</th>
<th>Features of the best algorithm with memorization</th>
<th>Best in literature?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1</td>
<td>r_i</td>
<td>\sum C_i$</td>
<td>80 jobs</td>
</tr>
<tr>
<td>$1</td>
<td>\vec{d}_i</td>
<td>\sum w_iC_i$</td>
<td>40 jobs</td>
</tr>
<tr>
<td>$F2</td>
<td></td>
<td>\sum C_i$</td>
<td>30 jobs</td>
</tr>
<tr>
<td>$1</td>
<td></td>
<td>\sum T_i$</td>
<td>300 jobs</td>
</tr>
<tr>
<td></td>
<td>130 jobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>130 jobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40 jobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40 jobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30 jobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30 jobs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Part 3: work done together with:

- Federico Della Croce
- Vincent T’Kindt

For theoretical guarantee: branch smartly and Merge !

For practical efficiency: Branch & Memorize

- Memorization is a powerful technique for scheduling problems
- Should be considered as an essential building block of branching algorithms
- The choice of branching scheme and search strategy are important
GRACIAS
ARIGATO
SHUKURIA
THANK YOU
BOLZIN
MERCI

Merging and Memorization

Conclusion