Investigation of some Stochastic Scheduling Problems

Master's Thesis in Computer Science

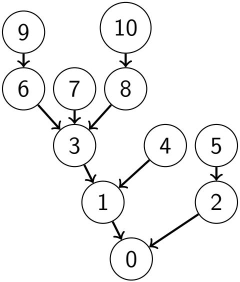
Philipp Müller

Technische Universität München

November 20, 2013

Problem statement

- Set of tasks
- Dependencies form intree structure
- Task times exponentially distributed with same parameter
- Nonpreemptive scheduling
- Goal: Minimize total expected make span



Schedules and snapshots

Schedules

- Describes the order of tasks
- Our scenario: non-deterministic (task times are random)
- Scheduling *strategy* influences expected make span

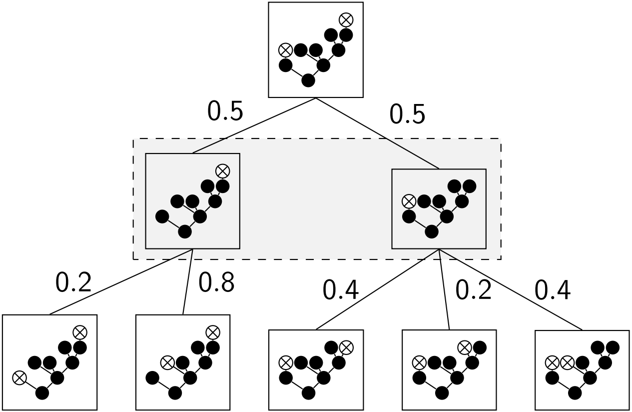
Snapshots

- States of a schedule
- Consist of current intree and the set of scheduled tasks

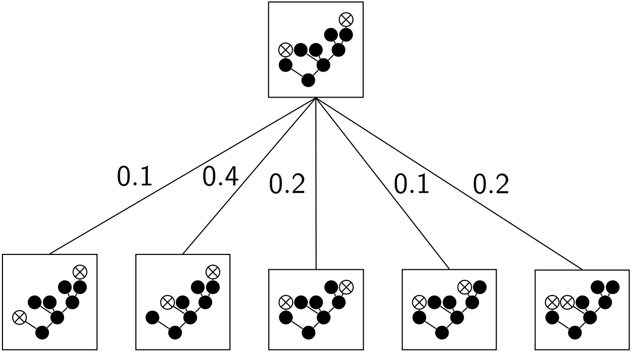
Epected run time

- r tasks currently scheduled
- Each of these tasks is the first to finish with probability $\frac{1}{r}$
- Compute run time recursively (task times memoryless)
    - Expected time for fastest task $\frac{1}{r}$
    - Weighted expected time for successive snapshots

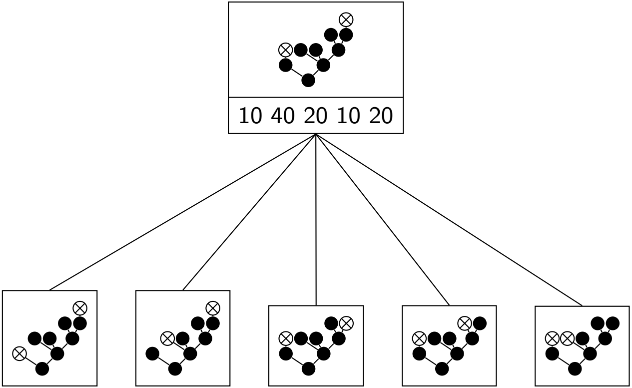
Schedule visualization



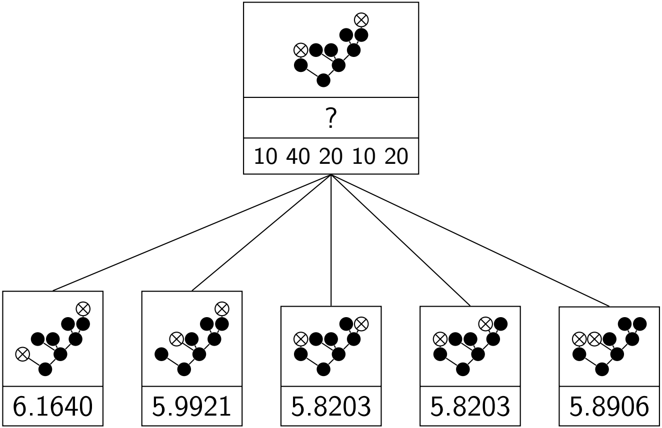
Schedule visualization



Schedule visualization

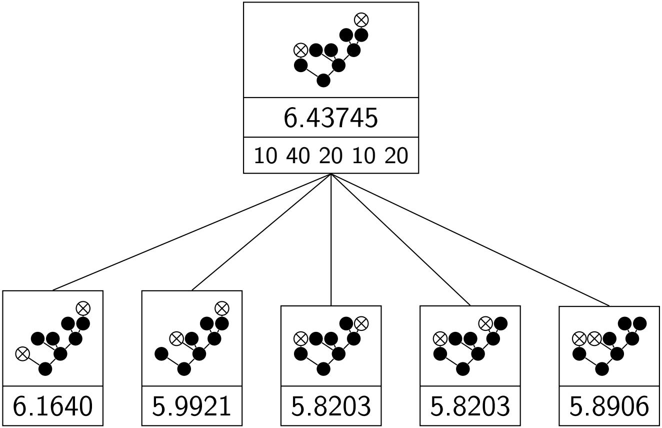


Schedule visualization



$$\frac{1}{2} + 0.1 \cdot 6.1640 + 0.4 \cdot 5.9921 + 0.2 \cdot 5.8203 + \cdots = 6.43745$$

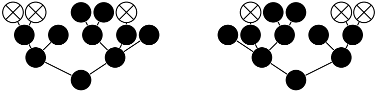
Schedule visualization



$$\frac{1}{2} + 0.1 \cdot 6.1640 + 0.4 \cdot 5.9921 + 0.2 \cdot 5.8203 + \cdots = 6.43745$$

Equivalent snapshots

These two snapshots describe the same:

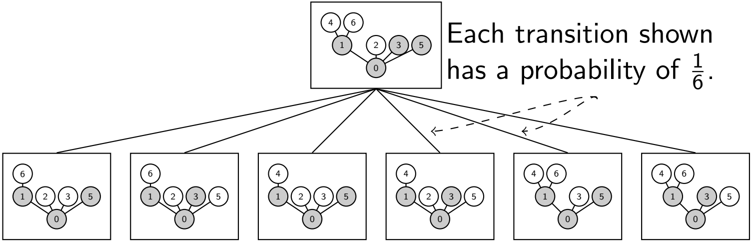


Equivalent snapshots

- Isomorphic intrees
- Scheduled tasks are mapped onto each other
- Equivalent snapshots excluded/re-used if they result from the same finishing task
- *Canonical* snapshots constructable in $O(n)$

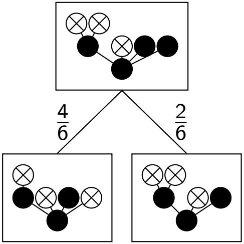
Equivalent snapshots — example

Each transition shown has a probability of $\frac{1}{6}$.



$\underbrace{\text{Task 4 finished first} \quad \text{Task 6 finished first}}_{\text{4 equivalent snapshots}} \quad \underbrace{\text{Task 2 finished first}}_{\text{2 equivalent snapshots}}$

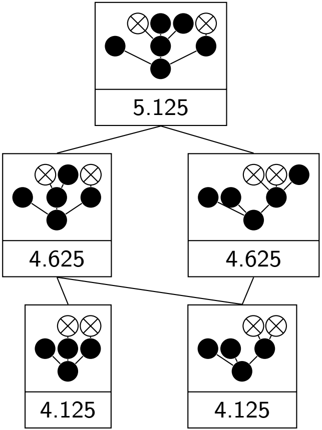
Equivalent snapshots — example



Two processors — optimal solution

Known results (Chandy, Reynolds 1975)

- Highest-level-first (HLF) optimal for two processors
- Profile (number of tasks per level) completely determines run time
- Reduce computation of expected run time to profiles



Expected run time — two-leaves intrees

Theorem (Maaß 2001)

Let $l, k \in \mathbb{N}$, $a \in \mathbb{N}_0$. Intrees with profile $\left[\!\left[(1)^{l-k}, (2)^k, (1)^{a+1}\right]\!\right]$ have expected run time

$$
\begin{aligned}
&\sum_{i=1}^{k} \left(\frac{1}{2}\right)^{l+i-1} \cdot \binom{l+i-2}{i-1} \cdot (k-i+2) \\
&+ \sum_{j=1}^{l} \left(\frac{1}{2}\right)^{k+j-1} \cdot \binom{k+j-2}{j-1} \cdot (l-j+2) \\
&+ \sum_{i=1}^{k} \sum_{j=1}^{l} \left(\frac{1}{2}^{k-i+l-j+1} \cdot \binom{ki+l-j}{l-j}\right) \\
&+ a.
\end{aligned}
$$

Expected run time — intrees with many 1-levels

Theorem

Intrees with profile $\left[\!\left[n_1,(1)^{j-2},n_j,(1)^{r-j}\right]\!\right]$ *(for* $j \geq 2$*) has expected run time*

$$\mathbb{E}\left[\left[\!\left[n_1,(1)^{j-2},n_j,(1)^{r-j}\right]\!\right]\right] = r + \frac{A_0(n_1-2)}{2^{n_1-1}} + \frac{A_{j-1}(n_j-2)}{2^{n_j+j-2}},$$

where A_i *is inductively defined as follows:*

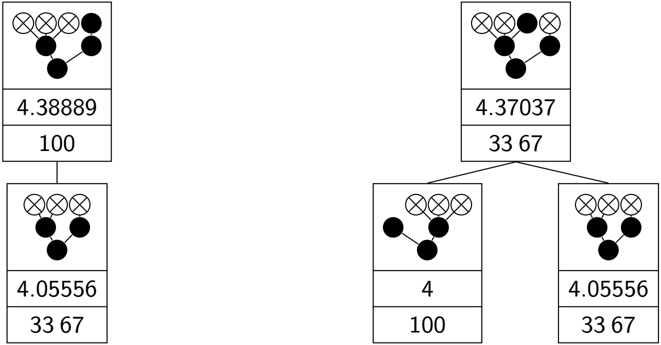
$$A_0(n) = (n+1)\cdot 2^n \quad A_{i+1}(n) = \sum_{k=0}^{n} A_i(k)$$

Profile DAG

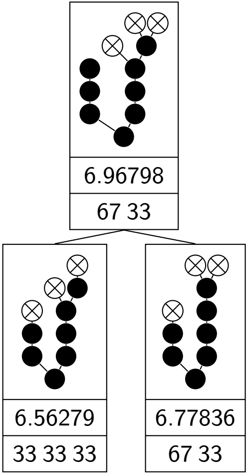
- Use profiles instead of whole snapshots
- Scheduled tasks given implicitly (HLF)
- Worst case profile $\left[\!\left[(1)^{\lfloor \frac{n}{2} \rfloor - 1}, \lceil \frac{n}{2} \rceil, 1\right]\!\right]$
- Worst case profile DAG size has $\lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil + 1$ "profile snapshots"
- Each profile snapshot accounts for less than n^2 "original snapshots"
  $\Rightarrow$ Simple proof: At most $O(n^4)$ original snapshots.

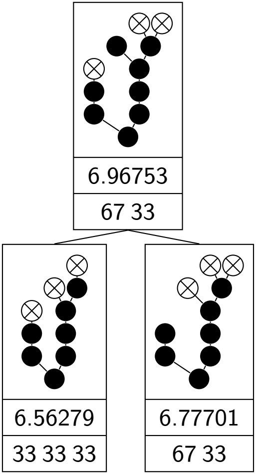
HLF may be ambiguous . . .

HLF may result in different run times:



...or even strictly suboptimal





HLF — summary

- HLF is suboptimal ...
- ... but asymptotically good (Papadimitriou, Tsitsiklis 1987): There is a function $\beta : \mathbb{N} \mapsto \mathbb{R}_0^+$ with $\lim_{n\to\infty} \beta(n) = 0$ such that for each intree I and an arbitrary HLF strategy HLF we have
$$T_{HLF}(I) \leq T_{\pi^*}(I) \cdot (1 + \beta(N)),$$
where π^* is the optimal strategy.
- Optimal schedule is often one particular run of HLF Wie oft – kurze Tabelle in Anhang! $\Rightarrow$ HLF is “can-optimal” for these intrees

(Dynamic) list scheduling

- Can not be optimal for our problem
- Optimal schedule has to consider previous choices
- HLF is particular instance of dynamic list scheduling

“2-HLF plus 1”

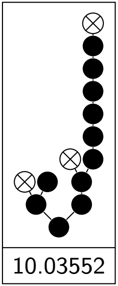
Motivation

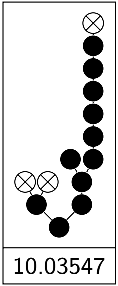
Non-HLF intrees up to 13 tasks have at most one non-HLF task scheduled.

Strategy

Discard snapshots with more than one non-HLF task scheduled.

Counterexample





Only highest and lowest leaves

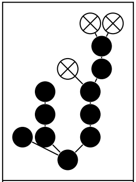
Motivation

Optimal schedules seen so far scheduled only topmost and lowest leaves.

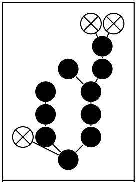
Strategy

Restrict to snapshots where only topmost and lowest leaves are scheduled.

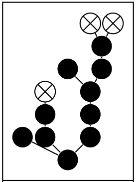
Counterexample



7.79154



7.8629



7.78793

Subtree with fewer topmost tasks

Motivation

In many cases, a topmost task being the single requirement for its direct successor has to be scheduled

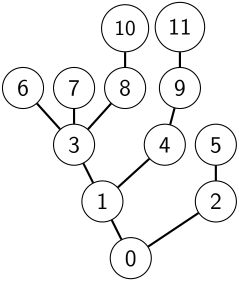
Definition (Topmost-maximal subtree for a leaf)

Let t be a leaf and let $p = (t, t_1, t_2, t_3, \ldots, r)$ be the path from t to the root r.
The *topmost-maximal subtree* for t is the subtree rooted at the *lowest* task t^* within p different from t that does *not* contain more topmost tasks than the subtree rooted at the direct successor of t.

Subtree with fewer topmost tasks

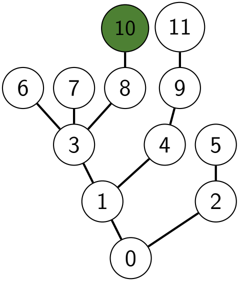
Topmost-maximal subtree for task 10:

Example intree



Subtree with fewer topmost tasks

Example intree

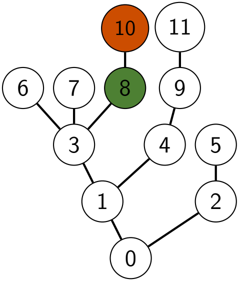


Topmost-maximal subtree for task 10:

- The subtree rooted at 10 (called I_{10}) contains only the topmost task 10 (omitted).

Subtree with fewer topmost tasks

Example intree

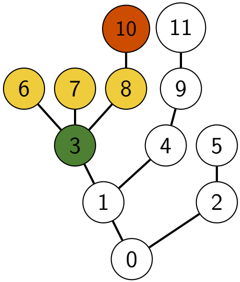


Topmost-maximal subtree for task 10:

- The subtree rooted at 10 (called I_{10}) contains only the topmost task 10 (omitted).
- Subtree I_8 contains only topmost task 10 (reference).

Subtree with fewer topmost tasks

Example intree

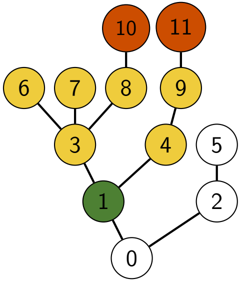


Topmost-maximal subtree for task 10:

- The subtree rooted at 10 (called I_{10}) contains only the topmost task 10 (omitted).
- Subtree I_8 contains only topmost task 10 (reference).
- Subtree I_3 still contains only 10 as the topmost task (it introduces only new *leaves*, namely 6 and 7).

Subtree with fewer topmost tasks

Example intree

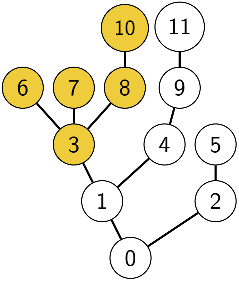


Topmost-maximal subtree for task 10:

- The subtree rooted at 10 (called I_{10}) contains only the topmost task 10 (omitted).
- Subtree I_8 contains only topmost task 10 (reference).
- Subtree I_3 still contains only 10 as the topmost task (it introduces only new *leaves*, namely 6 and 7).
- Subtree I_1 contains 10 *and* 11 as topmost tasks.

Subtree with fewer topmost tasks

Example intree



Topmost-maximal subtree for task 10:

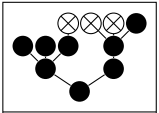
- The subtree rooted at 10 (called I_{10}) contains only the topmost task 10 (omitted).
- Subtree I_8 contains only topmost task 10 (reference).
- Subtree I_3 still contains only 10 as the topmost task (it introduces only new *leaves*, namely 6 and 7).
- Subtree I_1 contains 10 *and* 11 as topmost tasks.
- Subtree I_3 is the topmost maximal subtree for task 10.

Subtree with fewer topmost tasks

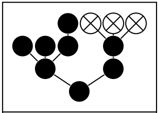
Strategy

Prefer tasks whose topmost-maximal subtree has fewer topmost tasks.

Counterexample



5.93467



5.93416

Subtree with fewer leaves

Motivation

Maybe we were wrong and should have focussed on *ready*, and not on *topmost* tasks.

Definition (Leaf-maximal subtree for a leaf)

Let t be a leaf and let $p = (t, t_1, t_2, t_3, \ldots, r)$ be the path from t to the root r.

The *leaf-maximal subtree* for t is the subtree rooted at the *lowest* task t^* within p different from t that does *not* contain more leaves than the subtree rooted at the direct successor of t.

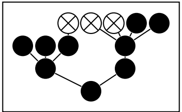
Remark: Preferring subtrees with more does not work.

Subtree with fewer leaves

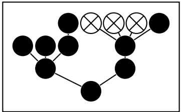
Strategy

Prefer tasks whose leaf-maximal subtree has fewer topmost tasks.

Counterexample



6.26835



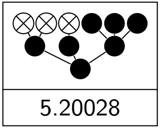
6.26749

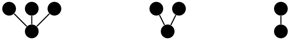
Recursive approach

Strategy

Prefer root's predecessors with maximal processing time.

Counterexample



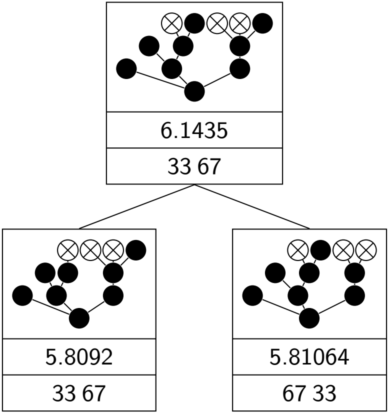


Filling up subtrees

Motivation

Subtrees seem to be filled up one after another.

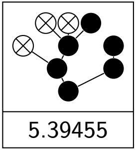
Counterexample



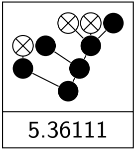
Maximizing 3-processor time, minimizing 1-processor time

Optimal schedules do *not* necessarily …

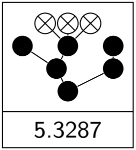
- …maximize the expected time span T_3 where 3 processors are busy
- …minimize the expected time span T_1 where only 1 processor can work



$T_1 = 2.98$



$T_3 = 2.36$



$T_3 = 2.33$, $T_1 = 2.99$

Optimal strategies — facts

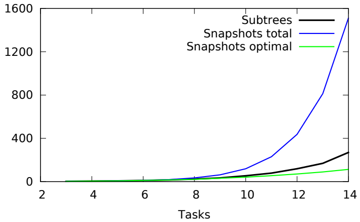
- No deliberate idleness (Chandy, Reynolds 1979)
- More processors do not worsen run time (Maaß 2001)
- Preemptive scheduling is strictly better than non-preemptive scheduling for 3 or more processors (equal for 2 processors)
- "Optimal schedules for subtrees do not help"

Computing optimal schedules

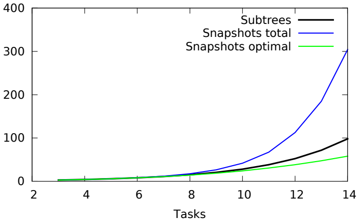
- Exhaustive search (*LEAF* scheduler)
- "Optimize" result from exhaustive search by recursively discarding bad choices
- Exponential run time

Size of the snapshot DAG

- Number of subtrees grows exponentially
  $\Rightarrow$ Number of overall snapshots as well
- Number of snapshots in optimal DAG remarkably smaller



Maximum



Average

Compressing the snapshot DAG

- Optimal snapshot DAG allows (recursive) merging of some snapshots
- LEAF snapshot DAG prevents us from merging because the run times are not the same for some snapshots Verbessern!

Optimal strategies — conjectures

- *In the beginning*, as many topmost task as possible shall be scheduled
- If the scheduler has a choice, it shall pick a topmost task, if available
- If only non-top tasks are scheduled, we can exchange any one with a topmost task and can obtain a better run time

Degenerate intrees

- On each level, at most one task has predecessors
- Uniquely determined by their profile

Theorem

Degenerate intrees are optimally scheduled by HLF.

Proof.

- Assert that for degenerate intree I, two tasks z_1, z_2 with $\mathit{level}(z_1) \geq \mathit{level}(z_2)$: $T^*_{x,y,z_1}(I \cup \{z_1\}) \geq T^*_{x,y,z_2}(I \cup \{z_2\})$
- Induction over the number of tasks, comparing $T_{HLF}(I)$ to $T_{x',y',z'}(I)$

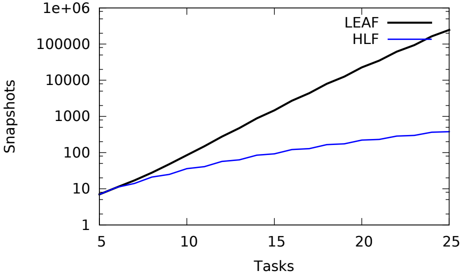
$\square$

Parallel chains

- Each task, except the root, has at most one predecessor
- Parallel chains with up to 27 tasks are optimally scheduled by HLF

Degenerate intrees and parallel chains — outcome

- HLF is deterministic for these classes ...
- ...and needs remarkably less snapshots than LEAF
- Example: Number of snapshots for degenerate binary trees

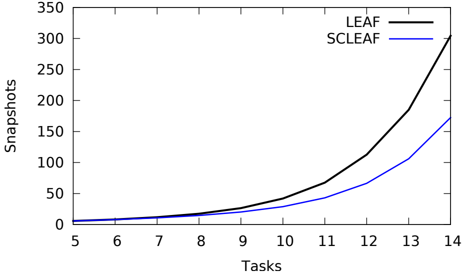


- Analogous results for parallel chains

Improving LEAF with conjectures

SCLEAF

- Simple LEAF scheduler, but ...
- ... using HLF when encountering degenerate intrees or parallel chains ...
- ... restricts to snapshots where as many topmost task as possible are scheduled



Practical results

- Excluding equivalent snapshots speeds up the program by at least factor 3 (for intrees with 11 or more tasks)
- Computing optimal schedules for all intrees with up to 15 tasks in 11 minutes and less than 2Gb of main memory
- More tasks require too much memory
- Computing optimal schedules for non-trivial intrees with 19 tasks takes one day
- Using Boost Rational to represent expectancies as fractions requires slightly more time and memory
- Using GMP to represent expectancies as fractions requires roughly doubles time and increases memory consumption by roughly 40%