Minimizing I/Os in Out-of-Core Task Tree Scheduling

Loris Marchal[1] Samuel McCauley[2] **Bertrand Simon**[1 3]
Frédéric Vivien[1]

1: CNRS, INRIA, ENS Lyon and Univ. Lyon, France.
2: IT University Copenhagen, Denmark.
3: University of Bremen.

Bremen – February 2019

Motivation

Scientific application workflows

- Described as DAGs: nodes = tasks, edges = dependencies
- Can be a tree (multifrontal sparse matrix factorization)

Focus on the memory needs

- Larger memory footprint: may not fit in main memory
- Resort to storing some files on disk: out-of-core execution
- Expensive disk access delays the execution
- Scheduling choices impact memory usage

Objective: Minimize I/Os while scheduling a tree-shaped workflow

Motivation

Scientific application workflows

- Described as DAGs: nodes = tasks, edges = dependencies
- Can be a tree (multifrontal sparse matrix factorization)

Focus on the memory needs

- Larger memory footprint: may not fit in main memory
- Resort to storing some files on disk: out-of-core execution
- Expensive disk access delays the execution
- Scheduling choices impact memory usage

Objective: Minimize I/Os while scheduling a tree-shaped workflow

Ultimate goal: parallel processing

- Problem: sequential case not well understood yet
- Our contribution: step towards its understanding

Outline

Problem modeling

Task tree

- In-tree $G = (V, E)$, each node has a single parent, $|V| = n$
- Output file of a node i: size w_i (integer number of slots)
- A node must be executed after all its children

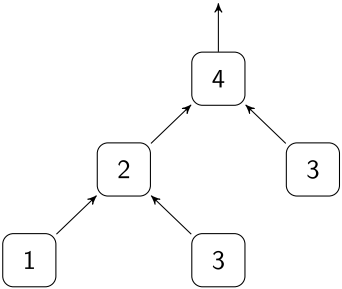
Memory model

- Main memory of size M, infinite disk
- Can move a slot to disk at unit cost: 1 I/O

Memory Management when a node is executed

- Children' output files stored in main memory
- Directly replaced by the node's output file (never coexist)

Example, $M = 5$

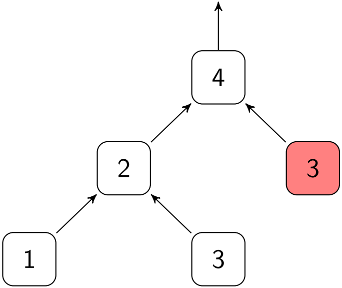


Memory: 0 / 5

Disk: 0

I/Os: 0

Example, $M = 5$

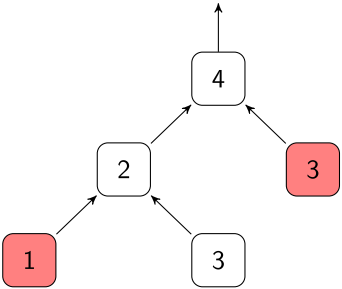


Memory: 3 / 5

Disk: 0

I/Os: 0

Example, $M = 5$

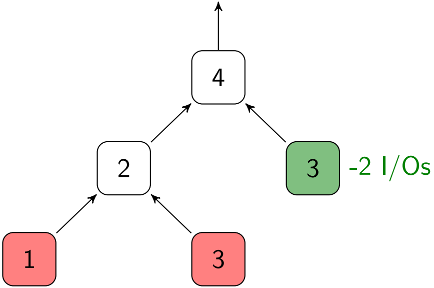


Memory: 4 / 5

Disk: 0

I/Os: 0

Example, $M = 5$

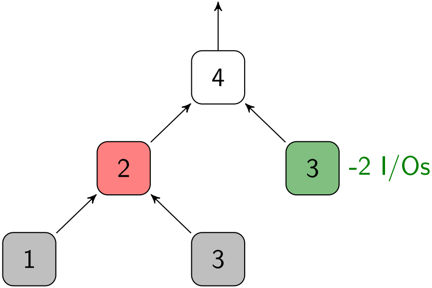


Memory: 5 / 5

Disk: 2

I/Os: 2

Example, $M = 5$

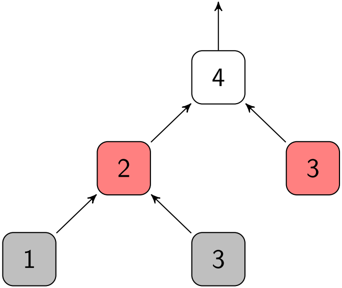


Memory: 3 / 5

Disk: 2

I/Os: 2

Example, $M = 5$

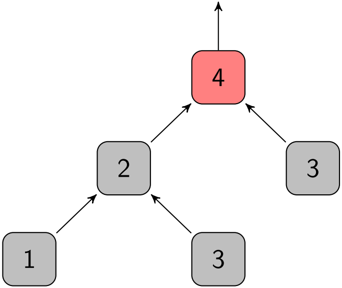


Memory: 5 / 5

Disk: 0

I/Os: 2

Example, $M = 5$



Memory: 4 / 5

Disk: 0

I/Os: 2

Model motivation

Other models used in the literature

- Input and output files coexist
- Additional memory used during execution
- Describe more accurately the reality

Advantages of this model **[Liu 1986, 1987]**

- Simpler theoretic study
- Previous models can be simulated by this one

Description of a solution

Traversal

- Schedule σ: $\sigma(i) = t$ if task i is the t- th executed
- I/O function τ: output file of task i has $\tau(i)$ slots written to disk
- Assume wlog that the data is written to disk ASAP and read ALAP

Validity of a traversal

- Schedule respects precedences
- I/Os consistent: $\tau(i) \leq w_i$
- The main memory (size M) is never exceeded, $\forall i \in V$:

$$\left(\sum_{\substack{(k,p)\in E \\ \sigma(k)<\sigma(i)<\sigma(p)}} (w_k - \tau(k)) \right) + \max\left(w_i, \sum_{(j,i)\in E} w_j \right) \leq M$$

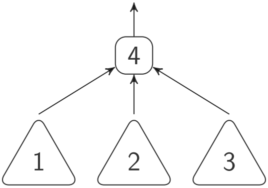
Objective

The MinIO problem

Given a tree G and a memory limit M, find a valid traversal that minimizes the total amount of I/Os ($= \sum \tau(i)$).

An interesting subclass: postorder traversals

- Fully process a subtree before starting a new one
- Completely characterized by the execution order of subtrees
- Widely used in sparse matrix softwares (e.g., MUMPS, QR-MUMPS)



Related work

Peak memory minimization **[Liu 1986, 1987]**

- Optimal Postorder algorithm: PostOrderMinMem in $\mathcal{O}(n \log n)$
- Optimal algorithm MinMemAlgo in $\mathcal{O}(n^2)$

Minimizing I/Os without splitting files **[Jacquelin et al. 2011]**

- Implies combinatorial choices: NP-complete
- NP-complete even restricted to postorders, or with σ known

Model similar to ours **[Agullo et al. 2010]**

- Optimal Postorder algorithm PostOrderMinIO in $\mathcal{O}(n \log n)$
- Did not consider the general problem

Complexity summary

	Trees	PostOrder	Unit DAGs (Pebble games)
Memory minimization	n^2	$n \log n$	NP-hard
I/O minimization, no splits	NP-hard	NP-hard	N/A
I/O minimization, splits	?	$n \log n$	N/A
I/O minimization, unit files	$n \log n$	$n \log n$	NP-hard

Preliminary results

Let (σ, τ) be an optimal traversal for MinIO of a given instance

Lemma (Schedule is enough)

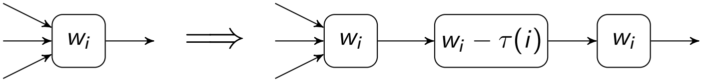
Given σ: the Furthest In the Future I/O policy minimizes I/Os.

Lemma (I/O function is enough)

Given τ: a valid traversal (σ', τ) can be computed in polynomial time.

Proof.

Expand each node following:



Then minimize the memory peak. □

Outline

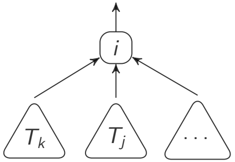
Postorder algorithms [Liu 1986, Agullo et al. 2010]

T_i : subtree rooted at node i

Characterization of a postorder schedule σ

- When executing T_i : order of execution of children of i
- S_i^σ, memory requirement of T_i :

$$S_i^\sigma = \max\left(w_i\ ,\ \max_{j \in Chil(i)}\left(S_j^\sigma + \sum_{\substack{k \in Chil(i) \\ \sigma(k) < \sigma(j)}} w_k\right)\right)$$



Solutions

- PostOrderMinMem: sort by decreasing values of $(S_j - w_j)$
- PostOrderMinIO: sort by decreasing values of $(\min\{S_j, M\} - w_j)$
- *Recall that the corresponding I/O function τ can be deduced*

Postorder traversals are optimal on homogeneous trees

Theorem

Both PostOrderMinMem *and* PostOrderMinIO *minimize I/Os on homogeneous trees (unit file sizes).*

Proof sketch (Generalization of [Sethi & Ullman 1970]).

- Define labels on nodes reflecting I/Os of PostOrderMinMem
- Prove the result by induction on $|V|$ of G
  - Take a schedule that needs I/Os
  - $G' \leftarrow$ subgraph of G remaining to schedule after the first I/O
  - Extensive case study: labels show that PostOrderMinMem does at most 1 I/O less on G' than on G □

Note: PostOrderMinMem does not rely on M so is optimal for any memory size and several memory layers (cache-oblivious)

Postorder traversals are optimal on homogeneous trees

Theorem

Both PostOrderMinMem *and* PostOrderMinIO *minimize I/Os on homogeneous trees (unit file sizes).*

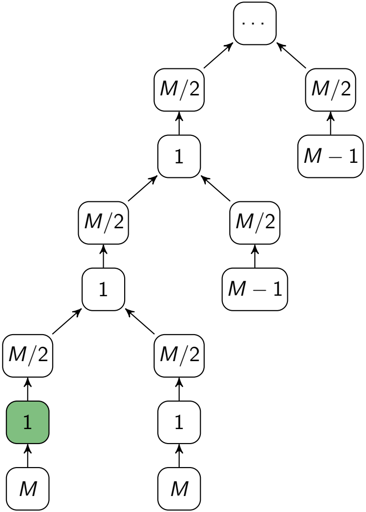
Proof sketch (Generalization of [Sethi & Ullman 1970]).

- Define labels on nodes reflecting I/Os of PostOrderMinMem
- Prove the result by induction on $|V|$ of G
  - Take a schedule that needs I/Os
  - $G' \leftarrow$ subgraph of G remaining to schedule after the first I/O
  - Extensive case study: labels show that PostOrderMinMem does at most 1 I/O less on G' than on G □

Note: PostOrderMinMem does not rely on M so is optimal for any memory size and several memory layers (cache-oblivious)

But PostOrderMinIO is not competitive on heterogeneous trees...

PostOrderMinIO is not competitive



I/O optimal

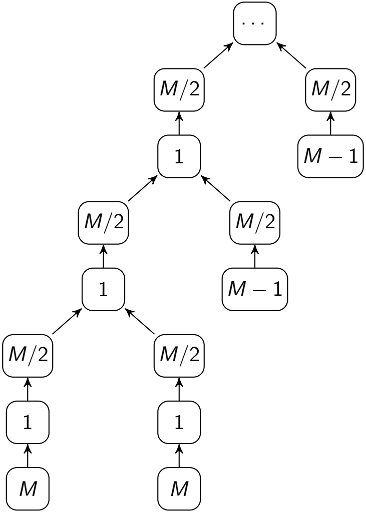
- Peak memory: $M + 1$
- I/Os: 1

PostOrderMinIO

- Peak memory: $\frac{3}{2}M$
- I/Os: $\Theta(|V|M)$

Competitive ratio: $\Omega(|V|M)$

PostOrderMinIO is not competitive



I/O optimal

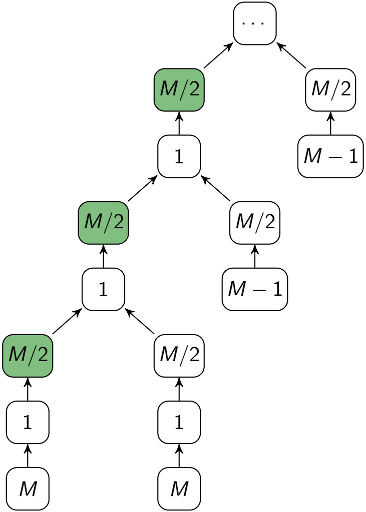
- Peak memory: $M+1$
- I/Os: 1

PostOrderMinIO

- Peak memory: $\frac{3}{2}M$
- I/Os: $\Theta(|V|M)$

Competitive ratio: $\Omega(|V|M)$

PostOrderMinIO is not competitive



I/O optimal

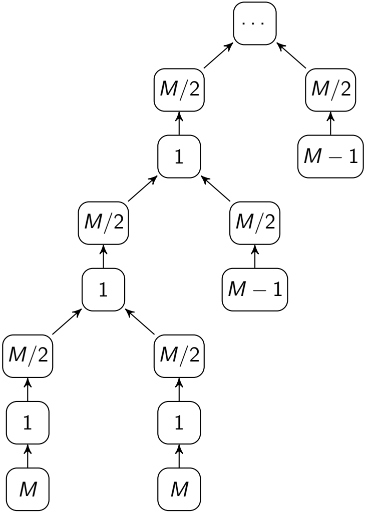
- Peak memory: $M + 1$
- I/Os: 1

PostOrderMinIO

- Peak memory: $\frac{3}{2}M$
- I/Os: $\Theta(|V|M)$

Competitive ratio: $\Omega(|V|M)$

PostOrderMinIO is not competitive



I/O optimal

- Peak memory: $M + 1$
- I/Os: 1

PostOrderMinIO

- Peak memory: $\frac{3}{2}M$
- I/Os: $\Theta(|V|M)$

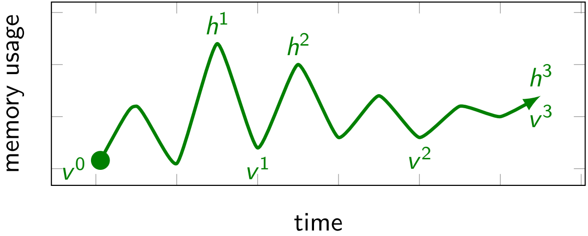
Competitive ratio: $\Omega(|V|M)$

Can we rely on MinMemAlgo?

MinMemAlgo [Liu 1986]

Hills and valleys of a traversal

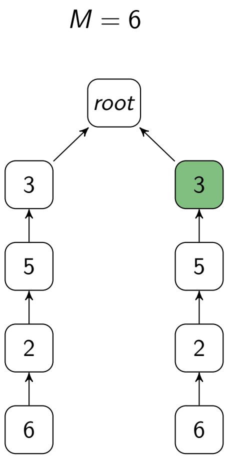
- First valley: v^0 = initial state
- Hill h^k: last state of highest memory consumption after valley v^{k-1}
- Valley v^k: last state of lowest memory consumption after hill h^k



Recursive algorithm

- Compute solution on each child subtree T_i, hills h_i^k and valleys v_i^k
- Sort hills and valleys by decreasing $Mem(h_i^k) - Mem(v_i^k)$
- For each couple (h_i^k, v_i^k) in this order:
  Schedule T_i following the recursive solution until valley v_i^k

MinMemAlgo is not competitive



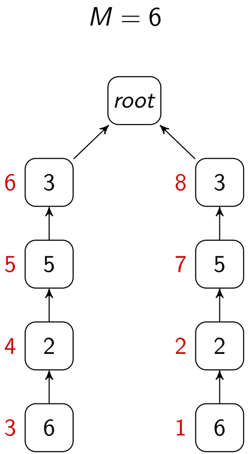
I/O Optimal

- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive



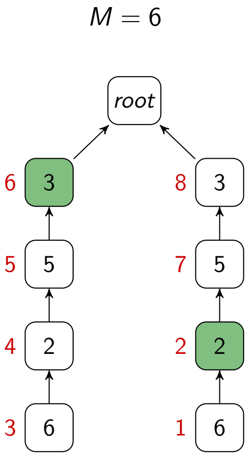
I/O Optimal

- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive



I/O Optimal

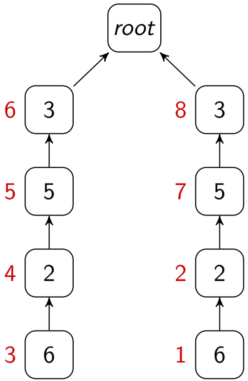
- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive

$M = 6$



I/O Optimal

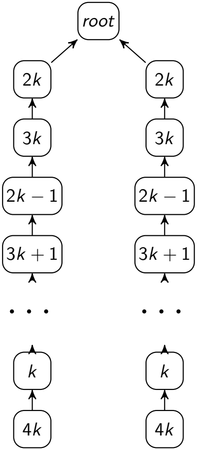
- Peak memory: 9
- I/Os: 3

MinMemAlgo (red labels)

- Peak memory: 8
- I/Os: 4

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

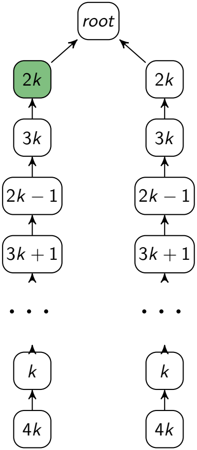
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

Competitive ratio: $\Omega(|V| + M)$

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

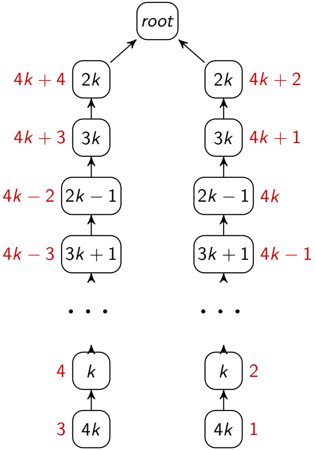
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

Competitive ratio: $\Omega(|V| + M)$

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

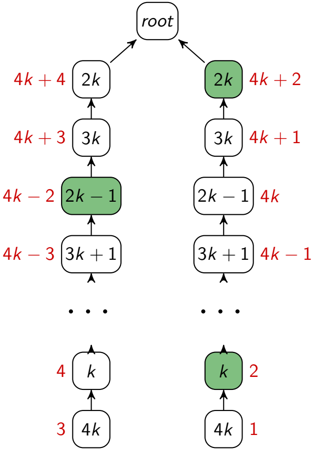
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

Competitive ratio: $\Omega(|V| + M)$

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

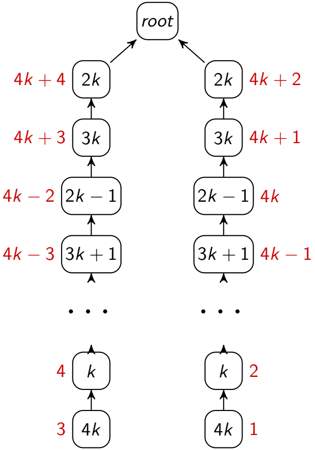
MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

Competitive ratio: $\Omega(|V| + M)$

MinMemAlgo is not competitive

$M = 4k$



I/O Optimal

- Peak memory: $6k$
- I/Os: $2k$

MinMemAlgo (red labels)

- Peak memory: $5k$
- I/Os: $> k^2$

Competitive ratio: $\Omega(|V| + M)$

Is a similar algorithm using M efficient?

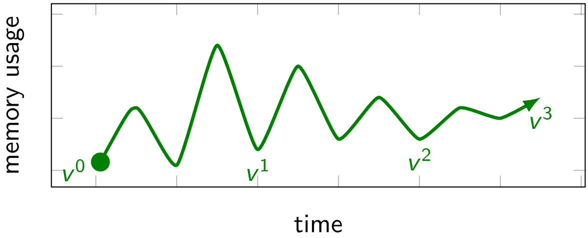
Lowcut schedules are not competitive

Low cut: cut of $T(i)$ of weight at most w_i

Lowcut schedule: only switch between two subtrees at low cuts

Intuition: generalizes MinMemAlgo, only switches subtree after

- making progress & reducing memory consumption



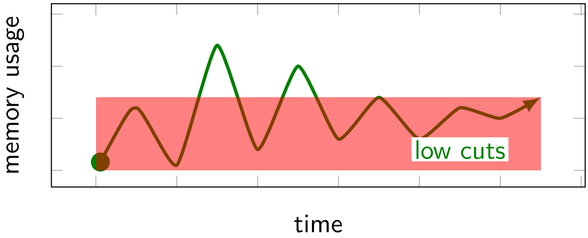
Lowcut schedules are not competitive

Low cut: cut of $T(i)$ of weight at most w_i

Lowcut schedule: only switch between two subtrees at low cuts

Intuition: generalizes MinMemAlgo, only switches subtree after

- making progress & reducing memory consumption



Lowcut schedules are not competitive

Low cut: cut of $T(i)$ of weight at most w_i

Lowcut schedule: only switch between two subtrees at low cuts

Intuition: generalizes MinMemAlgo, only switches subtree after

- making progress & reducing memory consumption

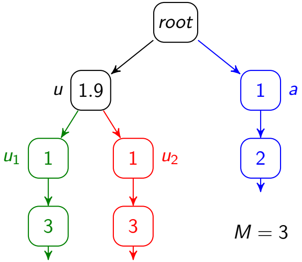
Theorem

No lowcut schedule is $O(|V|)$-competitive.

Example which can be expanded.

- MinIO: 1 I/O on u_1
- Lowcut schedules:
  - $+$ 1 I/O on a
  - or $+0.9$ I/O on u

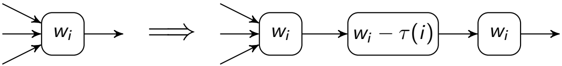
□



New heuristic: FULLRECEXPAND

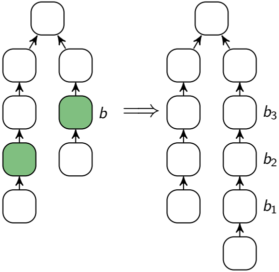
General description

- Underlying concept: run MINMEMALGO several times
- Each run: identify an I/O, then enforce it in the graph



FULLRECEXPAND

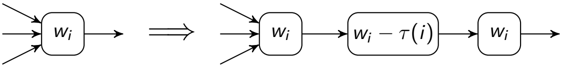
- Recursive calls on the root's children
- While MINMEMALGO needs I/Os:
  - Enforce the I/O that is the latest to be read from disk



New heuristic: FullRecExpand

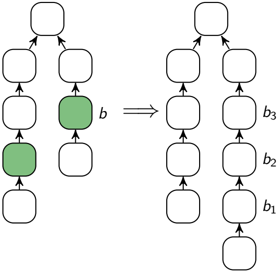
General description

- Underlying concept: run MinMemAlgo several times
- Each run: identify an I/O, then enforce it in the graph



FullRecExpand

- Recursive calls on the root's children
- While MinMemAlgo needs I/Os:
  - Enforce the I/O that is the latest to be read from disk



RecExpand ($\mathcal{O}(n^3)$): ≤ 2 iterations

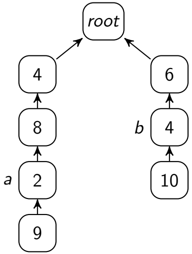
Example of RECEXPAND, M=10

MINMEMALGO:

peak = 12

I/Os = 4

I/Os enforced: 0

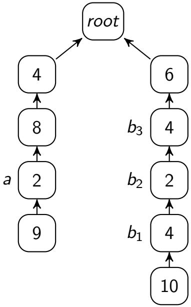


MINMEMALGO:

peak = 11

I/Os = 1

I/Os enforced: 2

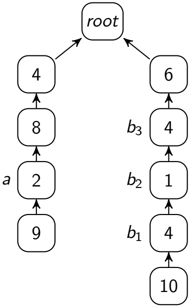


MINMEMALGO:

peak = 10

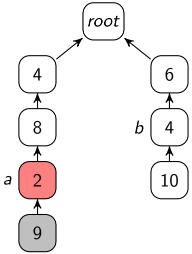
I/Os = 0

I/Os enforced: 3

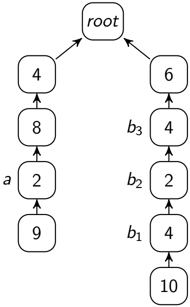


Example of RecExpand, M=10

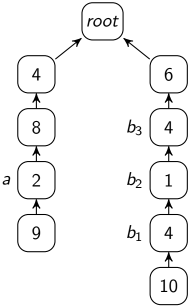
MinMemAlgo:
peak = 12
I/Os = 4
I/Os enforced: 0



MinMemAlgo:
peak = 11
I/Os = 1
I/Os enforced: 2



MinMemAlgo:
peak = 10
I/Os = 0
I/Os enforced: 3



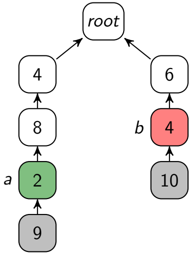
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

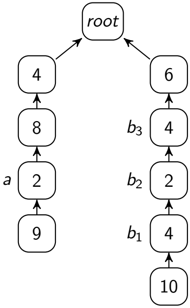


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

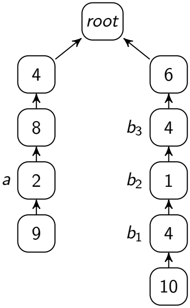


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3



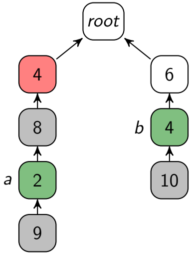
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

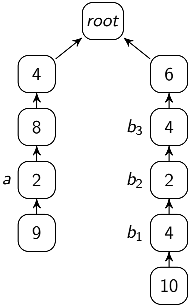


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

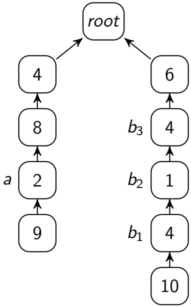


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3



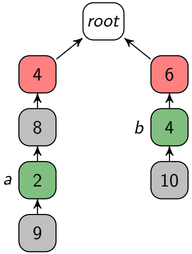
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

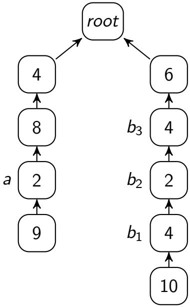


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

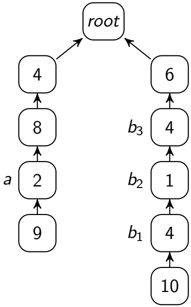


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3



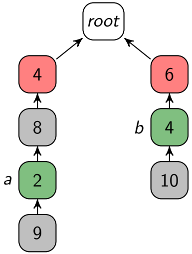
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

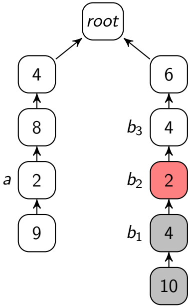


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

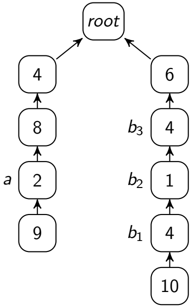


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3



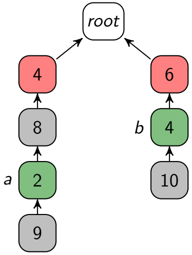
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

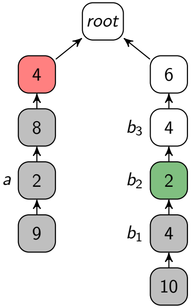


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

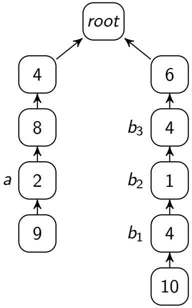


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3



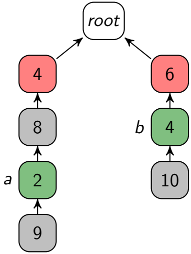
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

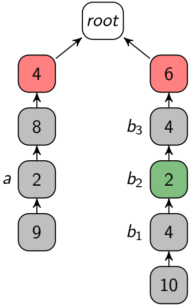


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

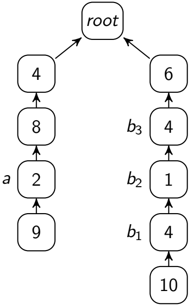


MinMemAlgo:

peak = 10

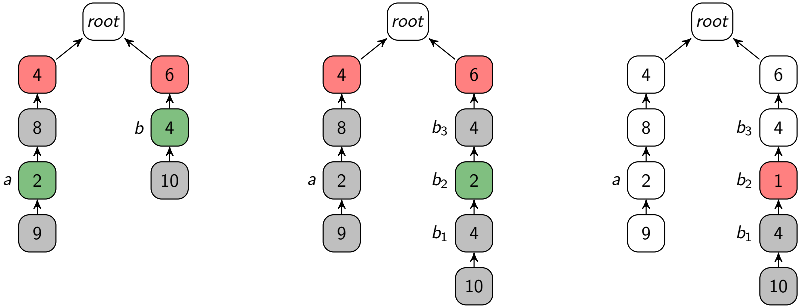
I/Os = 0

I/Os enforced: 3



Example of RecExpand, M=10

MinMemAlgo:	MinMemAlgo:	MinMemAlgo:
peak = 12	peak = 11	peak = 10
I/Os = 4	I/Os = 1	I/Os = 0
I/Os enforced: 0	I/Os enforced: 2	I/Os enforced: 3



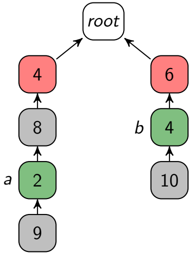
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

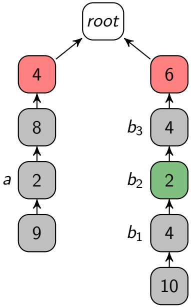


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

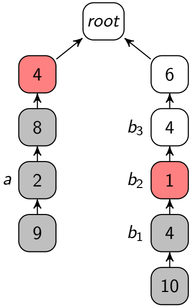


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3



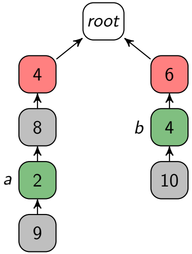
Example of RecExpand, M=10

MinMemAlgo:

peak = 12

I/Os = 4

I/Os enforced: 0

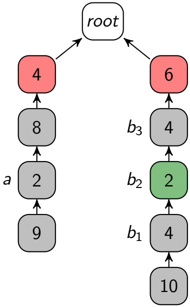


MinMemAlgo:

peak = 11

I/Os = 1

I/Os enforced: 2

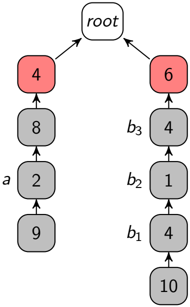


MinMemAlgo:

peak = 10

I/Os = 0

I/Os enforced: 3

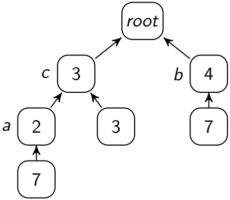


Example where RecExpand is not optimal, $M = 7$

PostOrderMinIO:

peak $= 10$

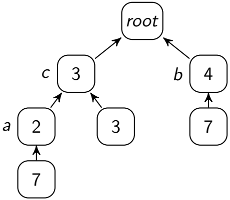
I/Os $= 3$



MinMemAlgo:

peak $= 9$

I/Os $= 4$

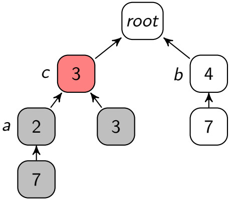


Example where RecExpand is not optimal, $M = 7$

PostOrderMinIO:

peak $= 10$

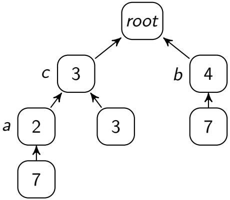
I/Os $= 3$



MinMemAlgo:

peak $= 9$

I/Os $= 4$

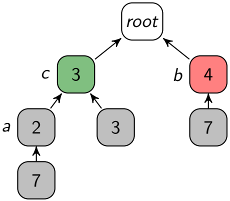


Example where RecExpand is not optimal, $M = 7$

PostOrderMinIO:

peak $= 10$

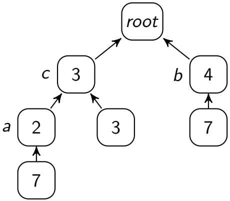
I/Os $= 3$



MinMemAlgo:

peak $= 9$

I/Os $= 4$

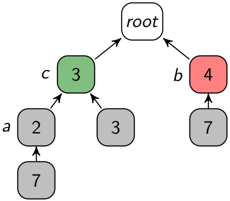


Example where RecExpand is not optimal, $M = 7$

PostOrderMinIO:

peak $= 10$

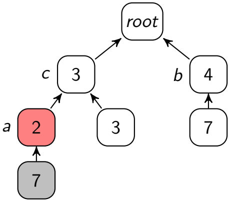
I/Os $= 3$



MinMemAlgo:

peak $= 9$

I/Os $= 4$

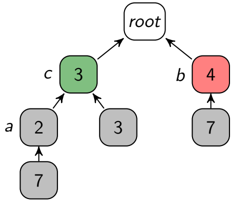


Example where RecExpand is not optimal, $M = 7$

PostOrderMinIO:

peak $= 10$

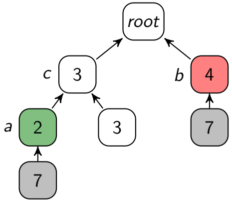
I/Os $= 3$



MinMemAlgo:

peak $= 9$

I/Os $= 4$

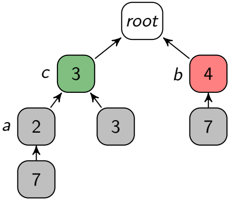


Example where RecExpand is not optimal, $M = 7$

PostOrderMinIO:

peak $= 10$

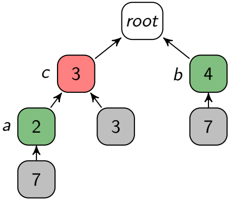
I/Os $= 3$



MinMemAlgo:

peak $= 9$

I/Os $= 4$



Outline

Experimental setup

Two datasets

- SYNTH: 330 synthetic binary trees of 3000 nodes uniformly drawn, memory weight uniform in $[1; 100]$
- TREES: 330 elimination trees of actual sparse matrices from 2000 to 40000 nodes (University of Florida Sparse Matrix Collection)
- Main memory size (M): mean of
  - Minimum memory for which a solution exists
  - Maximum memory for which I/Os are needed

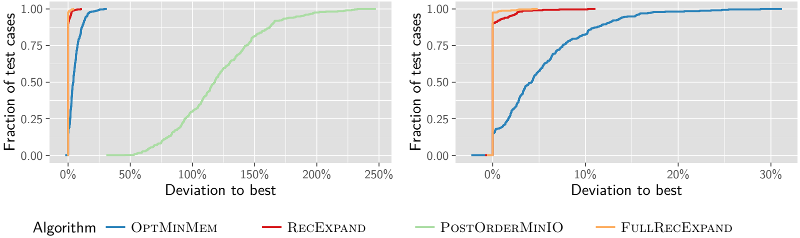
Heuristics

- MinMemAlgo, RecExpand, PostOrderMinIO, FullRecExpand

Performance

- If k I/Os are performed, performance is $1 + \dfrac{k}{M}$
- Objective: take into account the size of the main memory

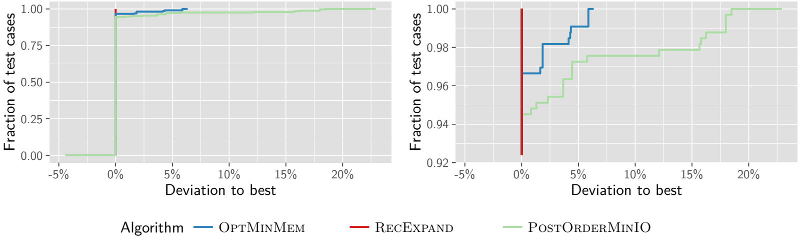
Results on SYNTH (right graph: zoom)



Analysis ***(Performance profiles: best is top-left)***

- Left: POSTORDERMINIO performs poorly ($>$ 100% deviation in 3/4 of the cases)
- Right: RECEXPAND significantly better than MINMEMALGO:
  - RECEXPAND best in $\approx$ 90% of the cases
  - MINMEMALGO best in $\approx$ 13% of the cases
- RECEXPAND is comparable to FULLRECEXPAND

Results on TREES (right graph: zoom)



Analysis *(best is top-left)*

- Smaller differences (right graph: zoom of the top-left part)
- Most of the graphs have *"easy"* solutions (cannot ensure optimality)
- RecExpand is always the best heuristic
- MinMemAlgo outperforms PostOrderMinIO

Outline

Conclusion

The MinIO problem

- Complexity still open
- Finding σ or τ suffices

Optimal solutions on subclasses

- Optimal postorder algorithm was already known
- PostOrderMinMem optimal for homogeneous trees

Heuristics

- MinMemAlgo performances are not bad
- RecExpand successfully combines the concepts of MinMemAlgo and the memory limit

Perspectives

- Recall: only concerns sequential schedules
- Next step: study I/O efficient parallel schedules (e.g., via memory booking)