Part 3: Memory-Aware DAG Scheduling

Minimize Memory for Trees

Minimize Memory for SP-Graphs

Minimize I/Os for Trees

Shared Memory of Parallel Processing

- Complexity and Space-Time Tradeoffs for Trees
- Processing DAGs with Limited Memory

Part 3: Memory-Aware DAG Scheduling

Minimize Memory for Trees

Minimize Memory for SP-Graphs

Minimize I/Os for Trees

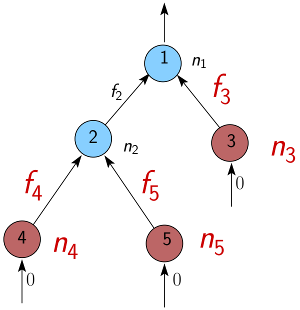
Shared Memory of Parallel Processing

- Complexity and Space-Time Tradeoffs for Trees
- Processing DAGs with Limited Memory

Part 3: Memory-Aware DAG Scheduling

Model for Parallel Tree Processing

- p uniform processors
- Shared memory of size M
- Task i has execution times p_i
- Parallel processing of nodes $\Rightarrow$ larger memory
- Trade-off time vs. memory



NP-Completeness in the Pebble Game Model

Background:

- Makespan minimization NP-complete for trees ($P|trees|C_{\max}$)
- Polynomial when unit-weight tasks ($P|p_i = 1, trees|C_{\max}$)
- Pebble game polynomial on trees

Pebble game model:

- Unit execution time: $p_i = 1$
- Unit memory costs: $n_i = 0, f_i = 1$
  (pebble edges, equivalent to pebble game for trees)

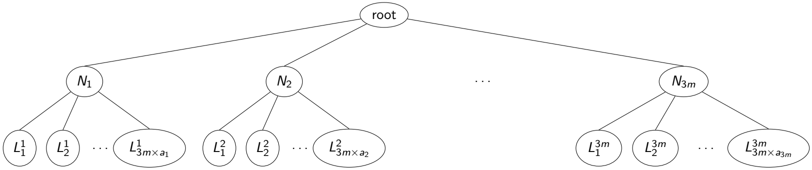
Theorem

Deciding whether a tree can be scheduled using at most B pebbles in at most C steps is NP-complete.

NP-Completeness – Proof

Reduction from 3-Partition:

- $3m$ integers a_i and B with $\sum ai = mB$,
- find m subsets S_k of 3 elements with $\sum_{i \in S_k} a_i = B$



Schedule the tree using:

- $p = 3mB$ processors,
- at most $B = 3m \times B + 3m$ pebbles,
- at most $C = 2m + 1$ steps.

Space-Time Tradeoff

Not possible to get a guarantee on both memory and time simultaneously:

Theorem 1

There is no algorithm that is both an α-approximation for makespan minimization and a β-approximation for memory peak minimization when scheduling tree-shaped task graphs.

Lemma

For a schedule with peak memory M and makespan $C_{\max}$,

$$M \times C_{\max} \geq 2(n-1)$$

Proof: each edge stays in memory for at least 2 steps.

Corollary: Lower Bound on Space-Time Product

For a schedule with peak memory M and makespan $C_{\max}$,

$$M \times C_{\max} \geq \sum_i mem\_needed\_for\_task_i \times p_i$$

Space-Time Tradeoff

Not possible to get a guarantee on both memory and time simultaneously:

Theorem 1

There is no algorithm that is both an α-approximation for makespan minimization and a β-approximation for memory peak minimization when scheduling tree-shaped task graphs.

Lemma

For a schedule with peak memory M and makespan $C_{\max}$,

$$M \times C_{\max} \geq 2(n-1)$$

Proof: each edge stays in memory for at least 2 steps.

Corollary: Lower Bound on Space-Time Product

For a schedule with peak memory M and makespan $C_{\max}$,

$$M \times C_{\max} \geq \sum_i mem\_needed\_for\_task_i \times p_i$$

Space-Time Tradeoff

Not possible to get a guarantee on both memory and time simultaneously:

Theorem 1

There is no algorithm that is both an α-approximation for makespan minimization and a β-approximation for memory peak minimization when scheduling tree-shaped task graphs.

Lemma

For a schedule with peak memory M and makespan $C_{\max}$,

$$M \times C_{\max} \geq 2(n-1)$$

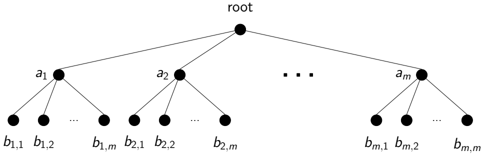
Proof: each edge stays in memory for at least 2 steps.

Corollary: Lower Bound on Space-Time Product

For a schedule with peak memory M and makespan $C_{\max}$,

$$M \times C_{\max} \geq \sum_i mem\_needed\_for\_task_i \times p_i$$

Space-Time Tradeoff – Proof



- With m^2 processors: $C^*_{\max} = 3$
- With 1 processor, sequentialize the a_i subtrees: $M^* = 2m$
- By contradiction, approximating both objectives: $C_{\max} \leq 3\alpha$ and $M \leq 2m\beta$
- But $M \times C_{\max} \geq 2(n-1) = 2m^2 + 2m$
- $2m^2 + 2m \leq 6m\alpha\beta$
- Contradiction for a sufficiently large value of m

Complexity – Summary

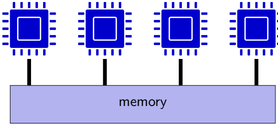
For task trees:

- Optimizing both makespan memory is NP-Complete
  $\Rightarrow$ Same for minimizing makespan under memory budget
- No scheduling algorithm can be a constant factor approximation on both memory and makespan

Part 3: Memory-Aware DAG Scheduling

Processing DAGs with Limited Memory

- Schedule general graphs
- On a shared-memory platform



First option: design good static scheduler:

- NP-complete, non-approximable
- Cannot react to unpredicted changes in the platform or inaccuracies in task timings

Second option:

- Limit memory consumption of *any dynamic scheduler*
  Target: runtime systems
- Without impacting too much parallelism

Part 3: Memory-Aware DAG Scheduling

Memory model

Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory

Memory model

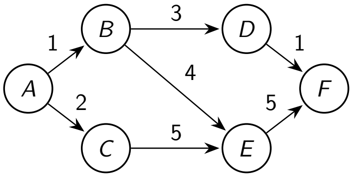
Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

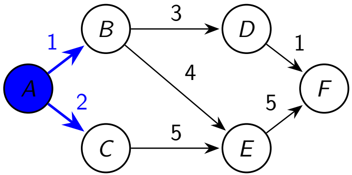
Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

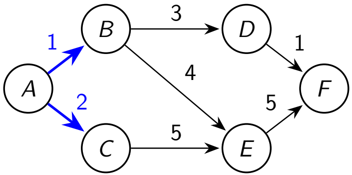
Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

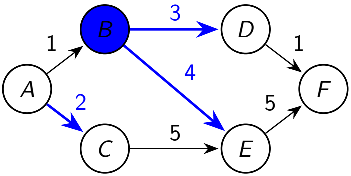
Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

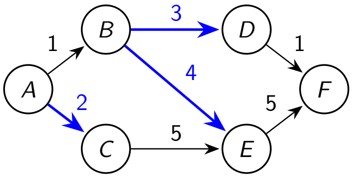
Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory



Memory model

Task graphs with:

- *Vertex weights* (w_i): task (estimated) durations
- *Edge weights* ($m_{i,j}$): data sizes

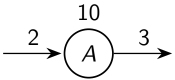
Simple memory model: at the beginning of a task

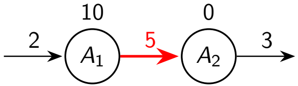
- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory

Emulation of other memory behaviours:

- Inputs + outputs allocated during task: duplicate nodes





Part 3: Memory-Aware DAG Scheduling

Minimize Memory for Trees

Minimize Memory for SP-Graphs

Minimize I/Os for Trees

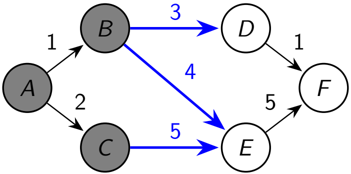
Shared Memory of Parallel Processing

- Complexity and Space-Time Tradeoffs for Trees
- Processing DAGs with Limited Memory
  - Model and maximum parallel memory
  - Maximum parallel memory/maximal topological cut
  - Efficient scheduling with bounded memory
  - Heuristics and simulations

Computing the maximum memory peak

Topological cut: (S, T) with:

- S include the source node, T include the target node
- No edge from T to S
- Weight of the cut = weight of all edges from S to T



Any topological cut corresponds to a possible state when all node in S are completed or being processed.

Two equivalent questions (in our model):

- What is the *maximum memory* of any parallel execution?
- What is the *topological cut with maximum weight*?

Computing the maximum topological cut

Literature:

- Lots of studies of various cuts in non-directed graphs ([Diaz,2000] on Graph Layout Problems)
- Minimum cut is polynomial on both directed/non-directed graphs
- Maximum cut NP-complete on both directed/non-directed graphs ([Karp 1972] for non-directed, [Lampis 2011] for directed ones)
- Not much for *topological* cuts

Theorem.

Computing the maximum topological cut of a DAG can be done in polynomial time.

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\min \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \quad p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut
- Then change the optimization direction (min $\rightarrow$ max)
- Draw w uniformly in $]0,1[$, define the cut such that $S_w = \{i \mid p_i > w\}, \quad T_w = \{i \mid p_i \leq w\}$
- Expected cost of this cut = M^* (opt. rational solution)
- All cuts with random w have the same cost M^*

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\min \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \quad p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut
- Then change the optimization direction (min $\rightarrow$ max)
- Draw w uniformly in $]0,1[$, define the cut such that $S_w = \{i \mid p_i > w\}, \quad T_w = \{i \mid p_i \leq w\}$
- Expected cost of this cut $= M^*$ (opt. rational solution)
- All cuts with random w have the same cost M^*

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\max \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} = p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \quad p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut
- Then change the optimization direction (min $\rightarrow$ max)
- Draw w uniformly in $]0,1[$, define the cut such that $S_w = \{i \mid p_i > w\}, \quad T_w = \{i \mid p_i \leq w\}$
- Expected cost of this cut $= M^*$ (opt. rational solution)
- All cuts with random w have the same cost M^*

Maximum topological cut – using LP

- Consider one classical LP formulation for finding a minimum cut:

$$\max \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$
$$\forall (i,j) \in E, \quad d_{i,j} = p_i - p_j$$
$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$
$$p_s = 1, \quad p_t = 0$$

- Integer solution $\Leftrightarrow$ topological cut
- Then change the optimization direction (min $\rightarrow$ max)
- Draw w uniformly in $]0,1[$, define the cut such that $S_w = \{i \mid p_i > w\}, \quad T_w = \{i \mid p_i \leq w\}$
- Expected cost of this cut $= M^*$ (opt. rational solution)
- All cuts with random w have the same cost M^*

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $maxdiff$ in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph → maximum topological cut

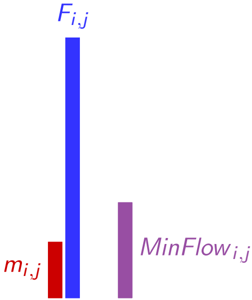
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow *maxdiff* in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph $\rightarrow$ maximum topological cut



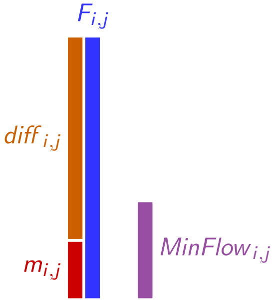
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow *maxdiff* in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph $\rightarrow$ maximum topological cut



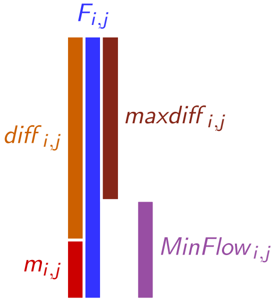
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow $maxdiff$ in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph → maximum topological cut



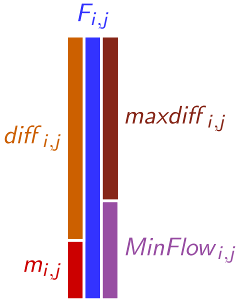
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Maximum topological cut – direct algorithm

- Dual problem: Min-Flow *(larger than all edge weights)*
- Idea: use an optimal algorithm for Max-Flow

Algorithm sketch

1. Build a large flow F on the graph G
2. Consider G^{diff} with edge weights $F_{i,j} - m_{i,j}$
3. Compute a maximum flow *maxdiff* in G^{diff}
4. $F - maxdiff$ is a minimum flow in G
5. Residual graph → maximum topological cut



Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

Summary 1

Predict the *maximal memory of any dynamic scheduling*

$\Leftrightarrow$

Compute the *maximal topological cut*

Two algorithms:

- Linear program + rounding
- Direct algorithm based on MaxFlow/MinCut

Downsides:

- Large running time: $O(|V|^2|E|)$ or solving a LP
- May include edges corresponding to the computing of more than p tasks

Faster Max. Memory Computation for SP Graphs

Recursive algorithm to compute maximum topological cut on SP-graphs:

- Single edge $i \rightarrow j$:
  $M(G) = m_{i,j}$
- Series combination:
  $M(G) = max(M(G_1), M(G_2))$
- Parallel combination:
  $M(G) = M(G_1) + M(G_2)$

Complexity: $O(|E|)$
Proof:

- consider tree of compositions: (full) binary tree
- $|E|$ leaves
- $|E| - 1$ internal nodes (compositions)

Maximum memory with p processors

Change in the model:

- Black (regular) edges
- Red edges corresponding to computations

Definition.

P-MaxTopCut Given a graph with black/red edges and a number p of processor, what is the maximal weight of a topological cut including at most p red edges ?

Theorem.

P-MaxTopCut is NP-complete

Special Case of SP Graphs – Exact Algorithm

Compute the maximum memory with p red edges $M(G, p)$:

- Adapt previous algorithm:
  Compute $M(G, k)$ for each $k = 1, \dots, p$
- Single edge $i \to j$:
  $M(G, k) = \begin{cases} m_{i,j} & \text{if edge is black or } k \geq 0 \\ -\infty & \text{otherwise} \end{cases}$
- Series combination:
  $M(G, k) = max(M(G_1, k), M(G_2, k))$
- Parallel combination:
  $M(G, k) = max_{j=0,\dots k} M(G_1, j) + M(G_2, k - j)$

Complexity:

- Simple Dynamic Programming algorithm: $O(|E|p^2)$.
- By restricting the search on each subgraph to $w(G)$ (maximum width), and with tighter analysis: $O(|E|p)$.

Special Case of SP Graphs – Exact Algorithm

Compute the maximum memory with p red edges $M(G, p)$:

- Adapt previous algorithm:
  Compute $M(G, k)$ for each $k = 1, \ldots, p$
- Single edge $i \to j$:
  $M(G, k) = \begin{cases} m_{i,j} & \text{if edge is black or } k \geq 0 \\ -\infty & \text{otherwise} \end{cases}$
- Series combination:
  $M(G, k) = max(M(G_1, k), M(G_2, k))$
- Parallel combination:
  $M(G, k) = max_{j=0,\ldots k} M(G_1, j) + M(G_2, k - j)$

Complexity:

- Simple Dynamic Programming algorithm: $O(|E|p^2)$.
- By restricting the search on each subgraph to $w(G)$ (maximum width), and with tighter analysis: $O(|E|p)$.

Special Case of SP Graphs – Exact Algorithm

Compute the maximum memory with p red edges $M(G,p)$:

- Adapt previous algorithm:
  Compute $M(G,k)$ for each $k = 1,\dots,p$
- Single edge $i \to j$:
  $M(G,k) = \begin{cases} m_{i,j} & \text{if edge is black or } k \geq 0 \\ -\infty & \text{otherwise} \end{cases}$
- Series combination:
  $M(G,k) = max(M(G_1,k), M(G_2,k))$
- Parallel combination:
  $M(G,k) = max_{j=0,\dots k} M(G_1,j) + M(G_2,k-j)$

Complexity:

- Simple Dynamic Programming algorithm: $O(|E|p^2)$.
- By restricting the search on each subgraph to $w(G)$ (maximum width), and with tighter analysis: $O(|E|p)$.

Special Case of SP Graphs – Exact Algorithm

Compute the maximum memory with p red edges $M(G, p)$:

- Adapt previous algorithm:
  Compute $M(G, k)$ for each $k = 1, \ldots, p$
- Single edge $i \rightarrow j$:
  $M(G, k) = \begin{cases} m_{i,j} & \text{if edge is black or } k \geq 0 \\ -\infty & \text{otherwise} \end{cases}$
- Series combination:
  $M(G, k) = max(M(G_1, k), M(G_2, k))$
- Parallel combination:
  $M(G, k) = max_{j=0,\ldots k} M(G_1, j) + M(G_2, k - j)$

Complexity:

- Simple Dynamic Programming algorithm: $O(|E|p^2)$.
- By restricting the search on each subgraph to $w(G)$ (maximum width), and with tighter analysis: $O(|E|p)$.

Special Case of SP Graphs – Approximation

Definition (Dual Approximation).

For a given guess λ, algo. that answers "1" if $M(G, p) \leq \lambda$ and "0" if $M(G, p) > \lambda/2$.

Idea:

- Consider only edges whose weight is $> \lambda/2p$
- Apply SP algorithms for without bound on p
- Return 1 iff $M(G, \infty) \geq \lambda/2$

Using binary search: 2-approximation algorithm

Summary 2

Predict the *maximal memory of any dynamic scheduling*

$\Leftrightarrow$

Compute the *maximal topological cut*

Two algorithms:

- Linear program + rounding
- Direct algorithm based on MaxFlow/MinCut

Downsides:

- Large running time ($O(|V|^2|E|)$)
- Taking into account the bound on task being processed makes the problem NP complete

Special case of SP graphs:

- Max. Top. cut computed in $O(|E|)$
- Max. Top. cut with p procs computed in $O(|E|p)$
- Max. Top. cut with p procs: 2-approximation in $O(|E|)$

Part 3: Memory-Aware DAG Scheduling

Coping with limiting memory

Problem:

- Limited available memory M
- Allow use of dynamic schedulers
- Avoid running out of memory
- Keep high level of parallelism (as much as possible)

Our solution:

- Add edges to guarantee that any parallel execution stays below M
  fictitious dependencies to reduce maximum memory
- Minimize the obtained *critical path*

$M = 10$

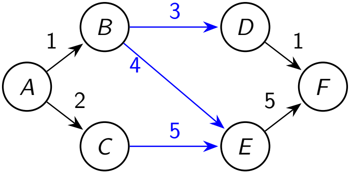
Coping with limiting memory

Problem:

- Limited available memory M
- Allow use of dynamic schedulers
- Avoid running out of memory
- Keep high level of parallelism (as much as possible)

Our solution:

- Add edges to guarantee that any parallel execution stays below M
  fictitious dependencies to reduce maximum memory
- Minimize the obtained *critical path*



$M = 10$

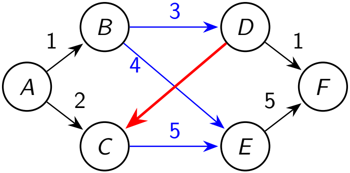
Coping with limiting memory

Problem:

- Limited available memory M
- Allow use of dynamic schedulers
- Avoid running out of memory
- Keep high level of parallelism (as much as possible)

Our solution:

- Add edges to guarantee that any parallel execution stays below M
  fictitious dependencies to reduce maximum memory
- Minimize the obtained *critical path*



$M = 10$

Problem definition and complexity

Definition (PartialSerialization).

Given a DAG $G = (V, E)$ and a bound M, find a set of new edges E' such that $G' = (V, E \cup E')$ is a DAG, $MaxMem(G') \leq M$ and $CritPath(G')$ is minimized.

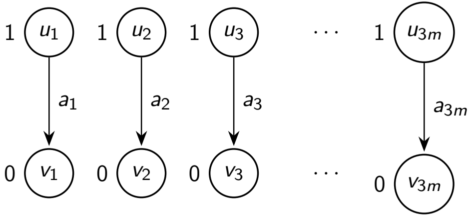
Theorem.

PartialSerialization is NP-hard in the stronge sense.

NB: stays NP-hard if we are given a sequential schedule σ of G which uses at most a memory M.

NP-completeness – proof sketch

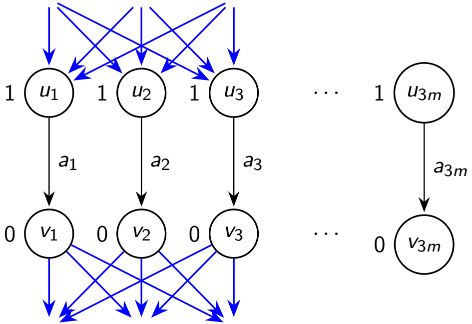
- Reduction from 3-Partition: a_i s.t. $\sum a_i = mB$, solution: m sets of 3 a_i's summing to B



- Set the memory bound to B
- Bound on the critical path: m
- Solution to PartialSerialization $\Leftrightarrow$ group edges by 3 s.t. $\sum a_i = B$

NP-completeness – proof sketch

- Reduction from 3-Partition: a_i s.t. $\sum a_i = mB$, solution: m sets of 3 a_i's summing to B



- Set the memory bound to B
- Bound on the critical path: m
- Solution to PartialSerialization $\Leftrightarrow$ group edges by 3 s.t. $\sum a_i = B$

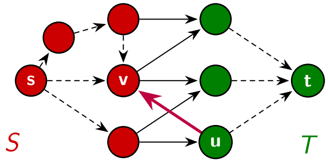
Part 3: Memory-Aware DAG Scheduling

Heuristic solutions for PartialSerialization

Framework:

(inspired by [Sbîrlea et al. 2014])

1. Compute a max. top. cut (S, T)
2. If weight $\leq M$: succeeds
3. Add edge (u, v) with $u \in T,\ v \in S$ without creating cycles; or fail
4. Goto Step 1



Several heuristic choices for Step 3:

MinLevels does not create a large critical path

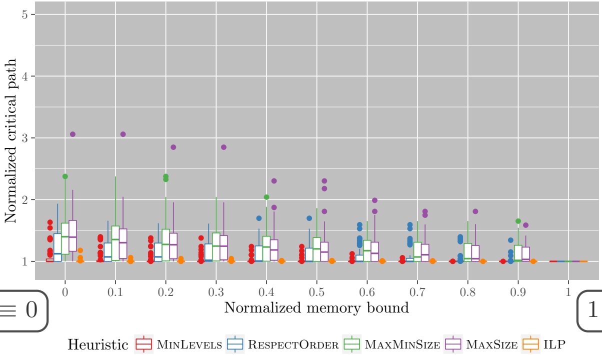
RespectOrder follows a precomputed memory-efficient schedule, always succeeds

MaxSize targets nodes dealing with large data

MaxMinSize variant of MaxSize

Simulations: dense random graphs (25, 50, 100 nodes)

lower is better



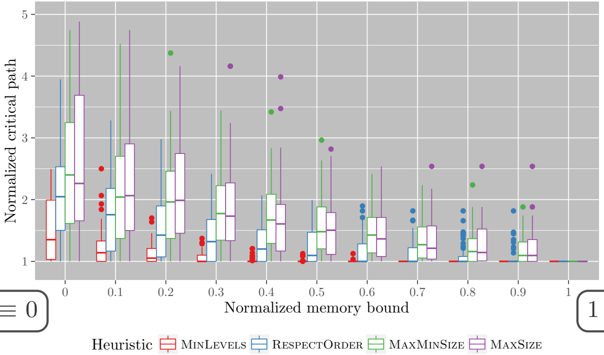
DFS memory $\equiv 0$

$1 \equiv$ MaxTopCut

- *x: memory (0 = DFS, 1 = MaxTopCut)*
  median ratio MaxTopCut / DFS ≈ 1.3
- *y: CP / original CP → lower is better*
- MinLevels performs best

Simulations: sparse random graphs (25, 50, 100 nodes)

lower is better



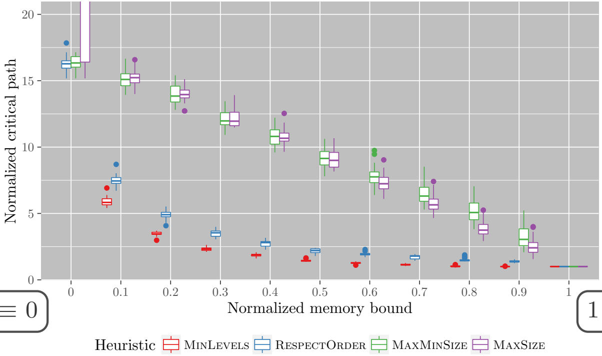
DFS memory ≡ 0

1 ≡ MaxTopCut

- *x: memory (0 = DFS, 1 = MaxTopCut)*
  median ratio MaxTopCut / DFS ≈ 2
- *y: CP / original CP → lower is better*
- MinLevels performs best, but might fail

Simulations – Pegasus workflows (LIGO 100 nodes)

lower is better



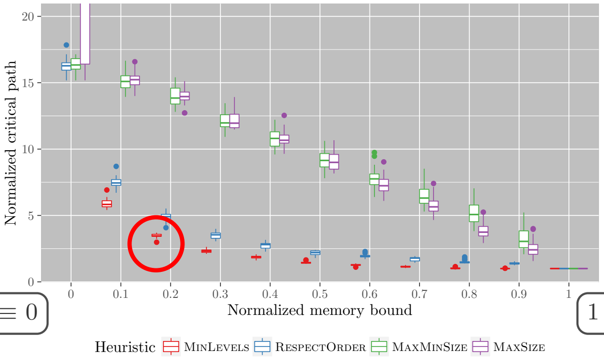
DFS memory ≡ 0

1 ≡ MaxTopCut

- *Median ratio MaxTopCut / DFS ≈ 20*
- MinLevels performs best, RespectOrder always succeeds
- Memory divided by 5 for CP multiplied by 3

Simulations – Pegasus workflows (LIGO 100 nodes)

lower is better



DFS memory ≡ 0

1 ≡ MaxTopCut

- *Median ratio MaxTopCut / DFS ≈ 20*
- MinLevels performs best, RespectOrder always succeeds
- Memory divided by 5 for CP multiplied by 3

Summary – Memory-Aware DAG Scheduling

Several models:

1. Memory weights on edges and nodes,
   inputs+outputs+tmp needed to compute tasks
2. Memory weights only on edges
   Processing tasks ⇔ replace inputs by outputs
3. (Memory increment on nodes)
   - Model 2 emulates 1, Model 3 emulates 1 and 2, ...
   - Choose the right model to solve each problem
   - Same for in-trees vs. out-trees

Results:

- One processor: optimal algorithms for trees (postorder or not)
- Several processors: NP-complete problem, no (α, β)-approx.
- Dynamic scheduling with memory bound:
  - Compute the worst memory: polynomial (linear for SP-graphs)
  - Limit memory: NP-complete, heuristic solutions

Summary – Memory-Aware DAG Scheduling

Several models:

1. Memory weights on edges and nodes,
   inputs+outputs+tmp needed to compute tasks
2. Memory weights only on edges
   Processing tasks $\Leftrightarrow$ replace inputs by outputs
3. (Memory increment on nodes)
   - Model 2 emulates 1, Model 3 emulates 1 and 2, ...
   - Choose the right model to solve each problem
   - Same for in-trees vs. out-trees

Results:

- One processor: optimal algorithms for trees (postorder or not)
- Several processors: NP-complete problem, no (α, β)-approx.
- Dynamic scheduling with memory bound:
  - Compute the worst memory: polynomial (linear for SP-graphs)
  - Limit memory: NP-complete, heuristic solutions