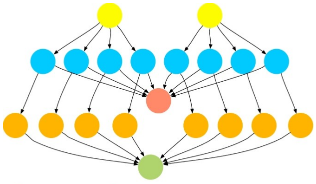
Parallel Scheduling of DAGs Under Memory Constraints

Loris Marchal, **Bertrand Simon** & Frédéric Vivien

Universität Bremen, Germany & ENS de Lyon, France

MAPSP, Renesse — 2019

Breaking down the title



DAGs of tasks

- Describe many applications
- Used by increasingly popular runtime schedulers
  (XKAAPI, StarPU, StarSs, ParSEC, ...)

Parallel scheduling

- Many tasks executed concurrently

Limited available memory (shared-memory platform)

- Simple breadth-first traversal may go out-of-memory

Objective

- Prevent dynamic schedulers from exceeding memory

Outline

Memory model

Task graph weights

- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

Memory model

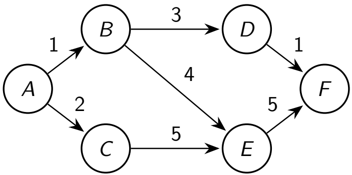
Task graph weights

- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior

- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory

$M_{used} = 0$



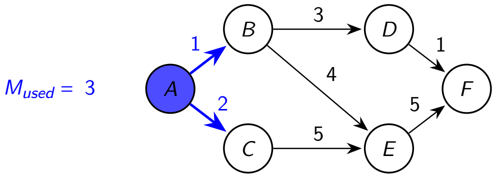
Memory model

Task graph weights

- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior

- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory



Memory model

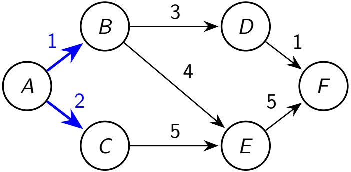
Task graph weights

- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior

- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory

$M_{used} = 3$



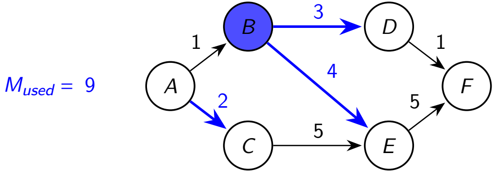
Memory model

Task graph weights

- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior

- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory



Memory model

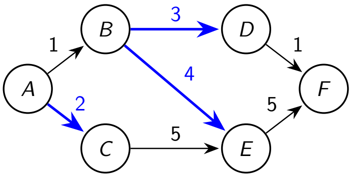
Task graph weights

- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

Memory behavior

- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory

$M_{used} = 9$



Memory model

Task graph weights

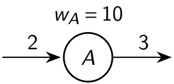
- Vertex w_i : estimated task duration
- Edge $m_{i,j}$: data size

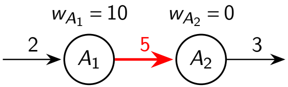
Memory behavior

- Task starts: free inputs (instantaneously)
  allocate outputs
- Task ends: outputs stay in memory

Emulation of other memory behaviours

- Inputs not freed, additional execution memory: duplicate nodes

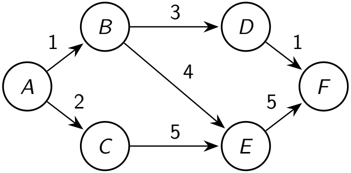




Maximum memory peak equivalent

Topological cut = partition of the vertices (S, T) with

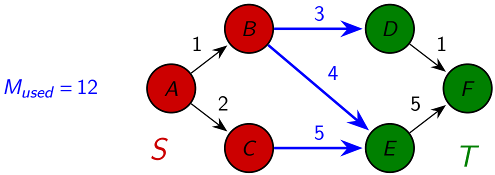
- Source $s \in S$ and sink $t \in T$
- No edge from T to S
- Weight of the cut $=$ sum of all edge weights from S to T



Maximum memory peak equivalent

Topological cut = partition of the vertices (S, T) with

- Source $s \in S$ and sink $t \in T$
- No edge from T to S
- Weight of the cut = sum of all edge weights from S to T

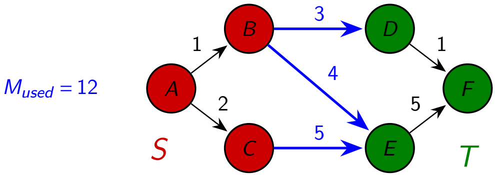


Topological cut $\longleftrightarrow$ execution state where T nodes are not started yet

Maximum memory peak equivalent

Topological cut = partition of the vertices (S, T) with

- Source $s \in S$ and sink $t \in T$
- No edge from T to S
- Weight of the cut $=$ sum of all edge weights from S to T



Topological cut $\longleftrightarrow$ execution state where T nodes are not started yet

Equivalence in our model between:

- Maximum memory peak (any parallel execution)
- Maximum weight of a topological cut

Computing the maximum topological cut

Literature

- Minimum cut is polynomial on graphs
- Maximum cut is NP-hard even on DAGs [Lampis et al. 2011]
- Not much for *topological* cuts

Theorem

Computing the maximum topological cut on a DAG is polynomial.

Maximum topological cut – using LP

A classical min-cut LP formulation

$$\min \sum_{(i,j)\in E} m_{i,j} d_{i,j}$$

$$\forall (i,j) \in E, \quad d_{i,j} \geq p_i - p_j$$

$$d_{i,j} \geq 0$$

$$p_s = 1, \quad p_t = 0$$

- Any graph: integer solution $\iff$ cut

Maximum topological cut – using LP

A classical min-cut LP formulation

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

- Any graph: integer solution $\iff$ cut
- Modify LP: 'min' $\rightarrow$ 'max' ; '$\geq$' $\rightarrow$ '$=$'

Maximum topological cut – using LP

A classical min-cut LP formulation

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

- Any graph: integer solution $\Longleftrightarrow$ cut
- Modify LP: 'min' $\rightarrow$ 'max' ; '$\geq$' $\rightarrow$ '$=$'

In a DAG, any (non-integer) optimal solution $\Longrightarrow$ max. top. cut

- Any rounding of the p_i's works (large $\in S$, small $\in T$)

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

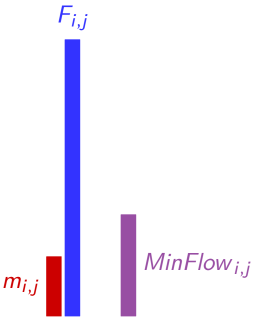
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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



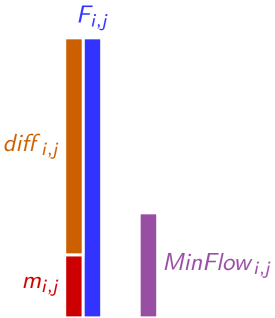
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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



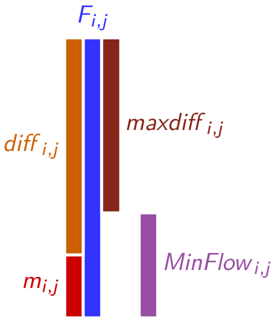
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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



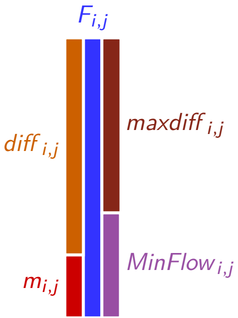
Complexity: same as maximum flow, e.g., $O(|V|^2|E|)$

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., $O(|V|^2|E|)$

Outline

Coping with limited memory

Problem

- Allow use of dynamic schedulers
- Limited available memory M
- Keep high level of parallelism

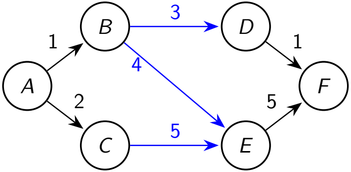
Coping with limited memory

Problem

- Allow use of dynamic schedulers
- Limited available memory M
- Keep high level of parallelism

Our solution

- Add edges to guarantee that any parallel execution stays below M
- Minimize the obtained *critical path*



$M_{\text{available}} = 10$

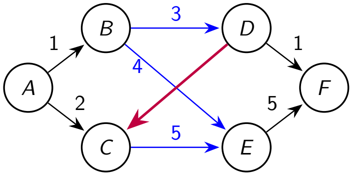
Coping with limited memory

Problem

- Allow use of dynamic schedulers
- Limited available memory M
- Keep high level of parallelism

Our solution

- Add edges to guarantee that any parallel execution stays below M
- Minimize the obtained *critical path*



$M_{\text{available}} = 10$

Problem definition and complexity

Definition (PartialSerialization of a DAG G under a memory M)

Compute a set of new edges E' such that:

- $G' = (V, E \cup E')$ is a DAG
- $MaxTopologicalCut(G') \leq M$
- $CritPath(G')$ is minimized

Theorem (Sethi 1975)

Computing a schedule that minimizes the memory usage is NP-hard.

$\Longrightarrow$ finding a DAG G' with $MaxTopologicalCut(G') \leq M$ is NP-hard

Theorem

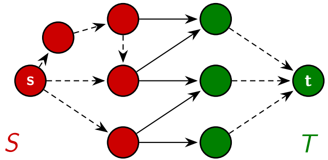
PartialSerialization *is NP-hard given a memory-efficient schedule.*

Optimal solution computable by an ILP (builds transitive closure)

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



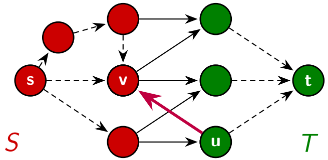
Several heuristic choices for Step 3

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

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

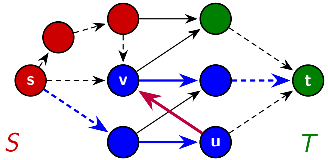
MaxSize targets nodes dealing with large data

MaxMinSize variant of MaxSize

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



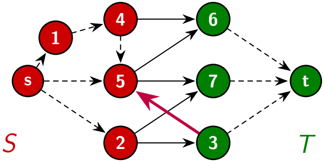
Several heuristic choices for Step 3

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

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

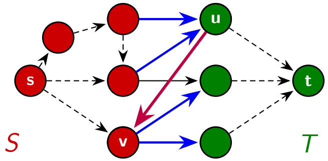
MaxSize targets nodes dealing with large data

MaxMinSize variant of MaxSize

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

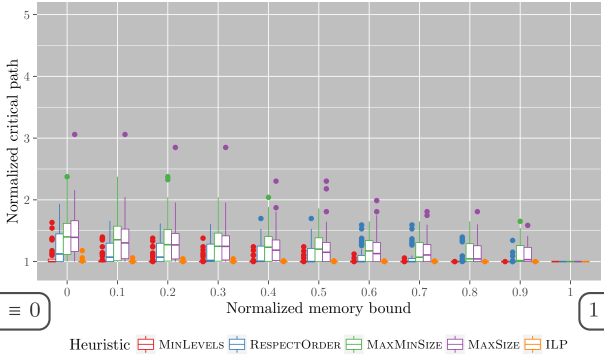
MaxSize targets nodes dealing with large data

MaxMinSize variant of MaxSize

Outline

Dense DAGGEN random graphs (25, 50, and 100 nodes)

lower is better



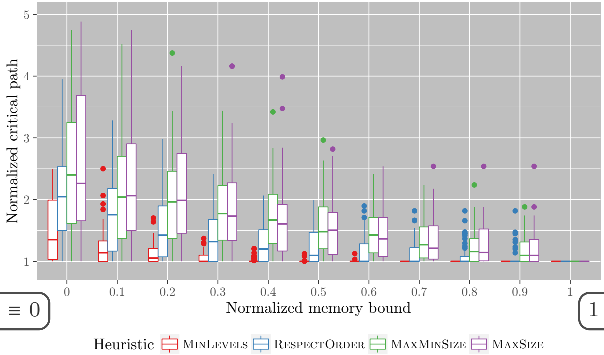
DFS memory ≡ 0

1 ≡ MaxTopCut

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

Sparse DAGGEN random graphs (25, 50, and 100 nodes)

lower is better



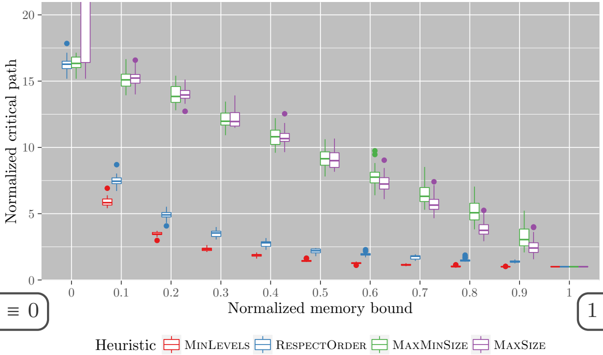
DFS memory ≡ 0

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



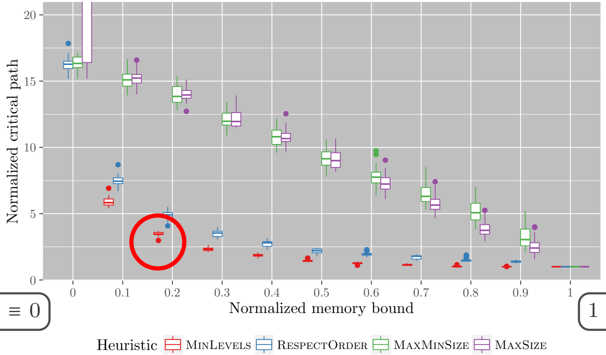
DFS memory ≡ 0

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

Outline

Conclusion

Memory model proposed

- Simple but expressive
- Explicit algorithm to compute maximum memory

Prevent dynamic schedulers from exceeding memory

- Adding fictitious dependences to limit memory usage
- Critical path as a performance metric
- Several heuristics (+ ILP)

Perspectives

- Reduce heuristic complexity
- Adapt performance metric to a platform
- Consider more *clever* schedulers
- Distributed memory