# Recent results and open questions on memory-aware DAG scheduling

Loris Marchal
(CNRS & Univ. Lyon)

Solharis kickoff meeting
Bordeaux, ~~December 2019~~ January 2020

# Outline

Processing DAGs with Limited Memory (Bertrand Simon's PhD)
- Model and maximum parallel memory
- Coping with limited memory

Maximum memory with $p$ processors (Gabriel Bathie's internship)
- NP-completeness
- SP graphs
- p-MaxTopCut for SP graphs
- Refined algorithms on SP graphs

Available code for DAGs and memory

# Outline

Processing DAGs with Limited Memory (Bertrand Simon's PhD)

- Model and maximum parallel memory
- Coping with limited memory

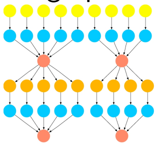
Maximum memory with $p$ processors (Gabriel Bathie's internship)

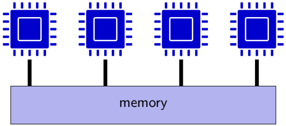
- NP-completeness
- SP graphs
- p-MaxTopCut for SP graphs
- Refined algorithms on SP graphs

Available code for DAGs and memory

# 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

# 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

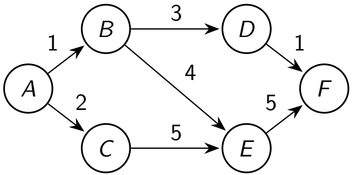
Task graphs with:

- 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

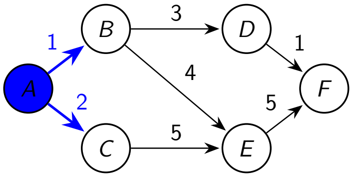
Task graphs with:

- 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

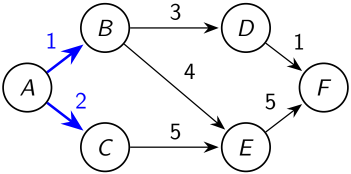
Task graphs with:

- 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

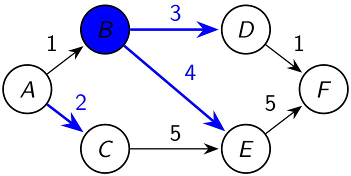
Task graphs with:

- 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

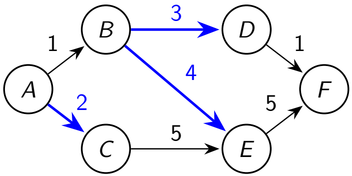
Task graphs with:

- 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

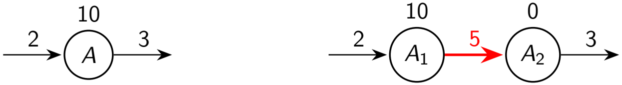
Simple memory model: at the beginning of a task

- Inputs are freed (instantaneously)
- Outputs are allocated

At the end of a task: outputs stay in memory

Emulation of other memory behaviours:

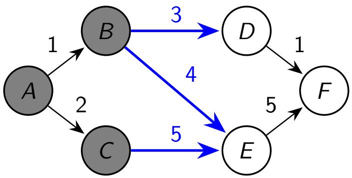
- Inputs + outputs allocated during task: duplicate nodes
  red edges represent memory during computations



# 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

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

# 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




$M = 10$

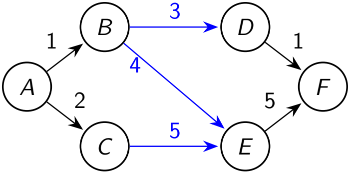
# Coping with limiting memory

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




$M = 10$

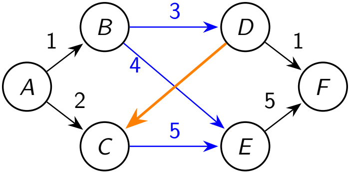
# Coping with limiting memory

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




$M = 10$

# Adding edges: 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.

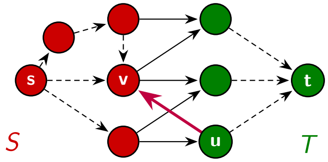
NB: stays NP-hard if we are given a sequential schedule $\sigma$ of $G$ which uses at most a memory $M$.

# 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

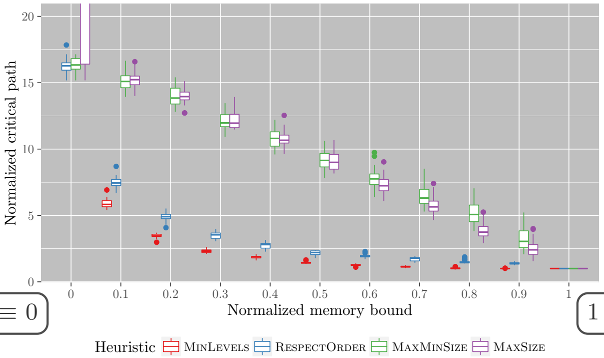
RespectOrder follows a precomputed memory-efficient schedule, always succeeds

MaxSize targets nodes dealing with large data

MaxMinSize variant of MaxSize

# Simulations – Pegasus workflows (LIGO 100 nodes)

*lower is better*

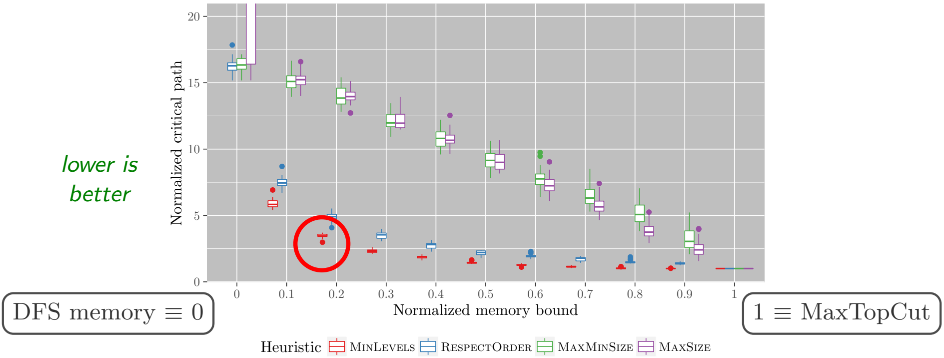



DFS memory $\equiv 0$

$1 \equiv$ MaxTopCut

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

# Simulations – Pegasus workflows (LIGO 100 nodes)




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

# Outline

Processing DAGs with Limited Memory (Bertrand Simon's PhD)

Model and maximum parallel memory

Coping with limited memory

Maximum memory with $p$ processors (Gabriel Bathie's internship)

NP-completeness

SP graphs

p-MaxTopCut for SP graphs

Refined algorithms on SP graphs

Available code for DAGs and memory

# 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.**

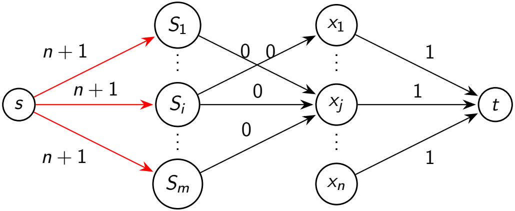
Computing the p-MaxTopCut is NP-complete

# NP-completeness of p-MaxTopCut

**Definition (Max-K-SubsetIntersection).**

Given a set $X$, subsets $S_i$ of $X$, find a collection $I$ of subsets such that $|I| = k$ and the intersection of $S_i$ for $i \in I$ covers at least $q$ elements of $X$.

(NP by reduction from Max-Edge-Biclique)




- edge $S_i \to x_j$ exists iff $x \notin S_i$
- look for a cut of weight $(n+1)k + q$
- $s \to S_i$ and $x_j \to t$ can be both in the cut only if $x_j \in S_i$
- other way to see it: $\overline{\bigcap S_i} = \bigcup \overline{S_i}$

# ILP for p-MaxTopCut

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

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

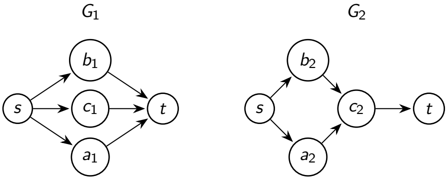
$$\sum_{(i,j)\in E} c_{i,j} d_{i,j} \leq p$$

$$\forall (i,j) \in E, \quad d_{i,j} \geq 0$$

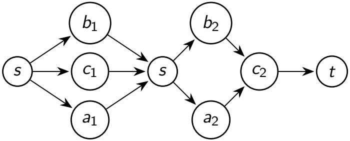
$$\forall i, p_i \in \{0,1\}, \;\; p_s = 1, \;\; p_t = 0$$

- Without constraints on $p$ red edges:
  LP Relaxation + rounding gives solution for MaxTopCut
- On Pegasus graphs, p-MaxTopCut only 1% smaller than MaxTopCut (small temporary data)
- On random graphs, p-MaxTopCut up to 3 times smaller (temporary data $\sim$ I/O data)

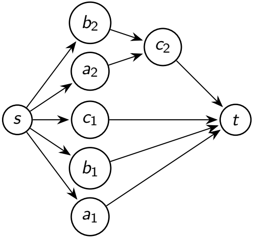
# Special case: Series-Parallel graphs




$Serie(G_1, G_2)$ :




$Par(G_1, G_2)$ :



# Computing Maximal Memory for SP graphs

Recursive algorithm to compute MaxTopCut on SP-graphs:

- For a 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)

# Computing p-MaxTopCut for SP graphs

Goal: compute maximum memory with $p$ red edges $M(G, p)$

- Adapt previous algorithm:
  Compute $M(G, k)$ for each $k = 1, \ldots, p$
- For a 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:

- Dynamic programming: $O(|E|p^2)$.

# Computing p-MaxTopCut for SP graphs

Goal: compute maximum memory with $p$ red edges $M(G, p)$

- Adapt previous algorithm:
  Compute $M(G, k)$ for each $k = 1, \ldots, p$
- For a 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:

- Dynamic programming: $O(|E|p^2)$.

# Computing p-MaxTopCut for SP graphs

Goal: compute maximum memory with $p$ red edges $M(G,p)$

- Adapt previous algorithm:
  Compute $M(G,k)$ for each $k = 1, \dots, p$
- For a 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:

- Dynamic programming: $O(|E|p^2)$.

# Computing p-MaxTopCut for SP graphs

Goal: compute maximum memory with $p$ red edges $M(G, p)$

- Adapt previous algorithm:
  Compute $M(G, k)$ for each $k = 1, \ldots, p$
- For a 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:

- Dynamic programming: $O(|E|p^2)$.

# Refined algorithms on SP graphs

Recent paper:
*Tim Kaler, William Kuszmaul, Tao B. Schardl, Daniele Vettorel: Cilkmem: Algorithms for Analyzing the Memory High-Water Mark of Fork-Join Parallel Programs. CoRR abs/1910.12340 (2019)*

- Better complexity for previous algorithm: $O(|E|p)$ (by restricting the search on each subgraph to $w(G)$, the maximum width of $G$, and with tighter analysis using potentials)
- 2-approximation with complexity $O(|E|)$

# Fast 2-approximation for p-MaxTopCut on SP graphs

**Definition (Dual Approximation).**

For a given guess $\lambda$, algorithm that answers YES if $M(G, p) \leq \lambda$ and NO if $M(G, p) > \lambda/2$.

Idea:

- Consider only edges whose weight is $> \lambda/2p$
- Apply SP algorithms without bound on $p$
- Return NO if $M(G, \infty) \geq \lambda/2$, YES otherwise

Using binary search: 2-approximation algorithm

# Summary

Results on maximum memory:

- Maximum parallel memory = MaxTopCut
- Two algorithms to compute MaxTopCut:
  - Linear program + rounding
  - Direct algorithm based on MaxFlow/MinCut
- Downsides of MaxTopCut:
  - Large running time ($O(|V|^2|E|)$)
  - Taking into account the bound on task being processed makes the problem NP complete: p-MaxTopCut

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 $O(|E|)$

# Open questions

- What to do if the graph is not Series-Parallel?
- And if the whole graph is not known in advance but dynamically uncovered?
- For now, we add (a tons of) edges to keep the (suposed stupid) runtime scheduler safe, but we could trust the scheduler more...
- Which information to give to the scheduler to avoid bad memory decision?
- What to do in distributed context?

# Outline

Processing DAGs with Limited Memory (Bertrand Simon's PhD)

Model and maximum parallel memory

Coping with limited memory

Maximum memory with $p$ processors (Gabriel Bathie's internship)

NP-completeness

SP graphs

p-MaxTopCut for SP graphs

Refined algorithms on SP graphs

Available code for DAGs and memory

# Available code for DAGs and memory

`https://gitlab.inria.fr/lmarchal/memdag`

Gathers most of the algorithms/codes produced on memory-aware scheduling of DAGs:

- Computing minimum memory (for sequential processing):
  - Liu's optimal algorithms (postorder and general)
  - Optimal algo. for SP graphs (with Enver, Thomas and Bora)
- Maximum parallel memory (MaxTopCut)
  and its limitation by adding new edges (with Bertrand)

# Available code for DAGs and memory

`https://gitlab.inria.fr/lmarchal/memdag`

Other useful algorithms:

- SP graph recognition: algo. by Valdes, Tarjan and Lawler
- SP-ization: custom algo. based on González-Escribano et al. (transformation into SP graph by adding synchronization vertices)

Graph formats:

- dot files
- list of nodes (trees)
- ask for more!

Feedback welcome !