Scheduling Task Graphs for Average Memory Consumption Reduction

Adrien Obrecht

ENS de Lyon

June 25, 2024

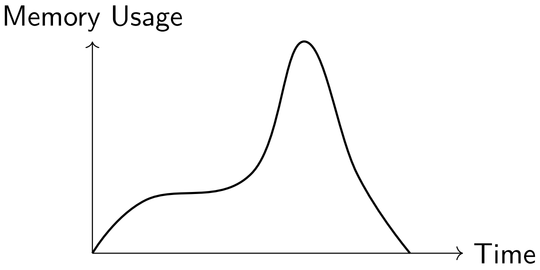
Overview

Introduction

- **Problem:** Scheduling task graphs to minimize memory consumption rather than execution speed.
- **Context:** High-performance computing, especially for applications with large memory footprints (e.g., machine learning).
- **Objective:** Reduce memory writes to external storage by minimizing memory usage during execution.

Intuition

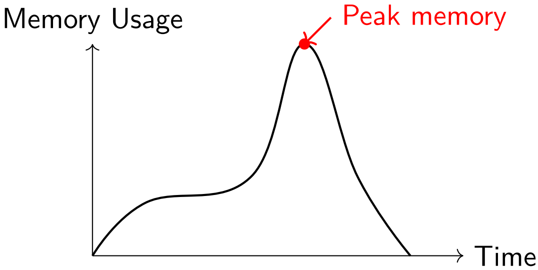
- Previous approaches target the minimization of the **peak memory**
- It may not be effective in shared memory environments
- We focus on reducing **average memory** consumption to improve overall performance



Potentially better for **parallel execution**

Intuition

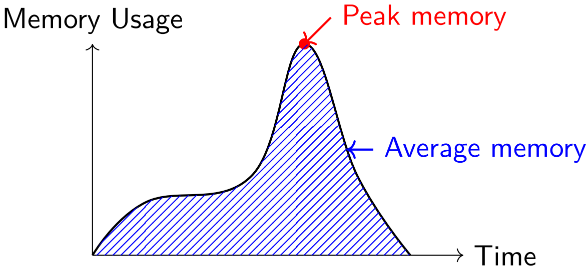
- Previous approaches target the minimization of the **peak memory**
- It may not be effective in shared memory environments
- We focus on reducing **average memory** consumption to improve overall performance



Potentially better for **parallel execution**

Intuition

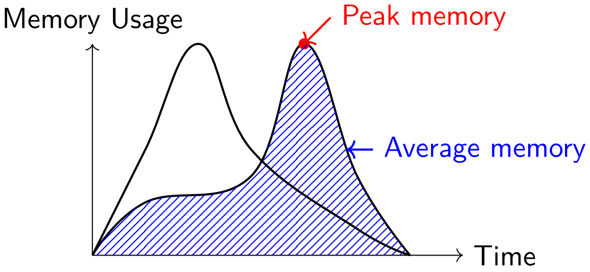
- Previous approaches target the minimization of the **peak memory**
- It may not be effective in shared memory environments
- We focus on reducing **average memory** consumption to improve overall performance



Potentially better for **parallel execution**

Intuition

- Previous approaches target the minimization of the **peak memory**
- It may not be effective in shared memory environments
- We focus on reducing **average memory** consumption to improve overall performance



Potentially better for **parallel execution**

2 Memory models

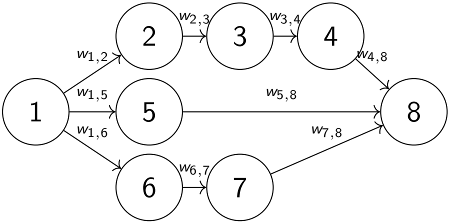


Figure: Pumpkin graph in the *multiple data* model

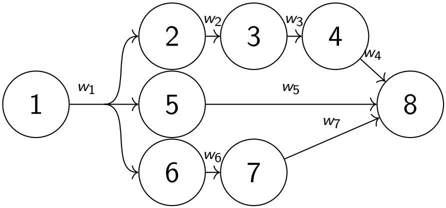


Figure: Pumpkin in the *single data* model

- **Task Graphs:** Represented by Directed Acyclic Graphs (DAGs).
- **Vertices (V):** Tasks.
- **Edges (E):** Data dependencies between tasks.
- **Goal:** Execute tasks to minimize memory consumption while respecting data dependencies and execution times.

Model - *multiple data*

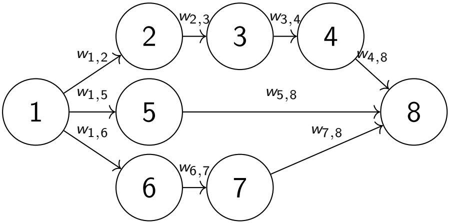


Figure: Pumpkin graph in the *multiple data* model

- Each edge (u, v) has an associated data of weight $w_{u,v}$
- The data stays in memory until all v has started its execution
- Equivalent to the Weighted Linear Arrangement problem

Definition (Weighted Linear Arrangement)

Given a valid schedule ϕ for a DAG G, we define the weighted linear arrangement cost of ϕ by:

$$WLA_G(\phi) = \sum_{(u,v)\in E} w_{u,v}(\phi(v) - \phi(u))$$

Model - *single data*

- Each vertex v has an associated outgoing data of weight w_v
- The data stays in memory until all childs of v have started being executed
- Equivalent to the Weighted Sum Cut problem

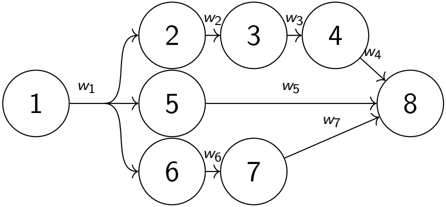


Figure: Pumpkin in the *single data* model

Definition (Weighted Sum Cut)

Given a valid schedule ϕ for a DAG G, we define the weighted sumcut of ϕ by:

$$WSC_G(\phi) = \sum_{u \in V} w_u \max_{v \in V^+(u)} (\phi(v) - \phi(u))$$

Previous works

Graph Type	Weighted	Unweighted
Directed	General : **NP**-C Out-tree : $\mathcal{O}(n\log(n))$ In-tree : $\mathcal{O}(n\log(n))$	General : **NP**-C Out-tree : $\mathcal{O}(n)$ In-tree : $\mathcal{O}(n)$
Undirected	General : **NP**-C	General : **NP**-C Tree : $\mathcal{O}(n^{1.58})$

Table: Summary of Linear Arrangement complexities (*multiple data* model)

Graph Type	Weighted	Unweighted
Directed	General : **NP**-C In-trees : $\mathcal{O}(n\log(n))$	General: **NP**-C In-tree : $\mathcal{O}(n)$ Out-tree : $\mathcal{O}(n)$
Undirected	General : **NP**-C	General : **NP**-C Tree : $\mathcal{O}(n)$

Table: Summary of SumCut complexities (*single data* model)

Linear Arrangement (*multiple data*) on pumpkins

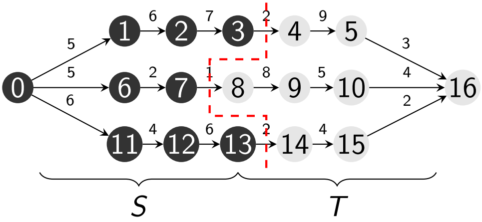


Figure: Min-cut (S, T) of a graph G

- We want to split the graph in half, to reuse algorithms on in-trees and out-trees

Linear Arrangement (*multiple data*) on pumpkins

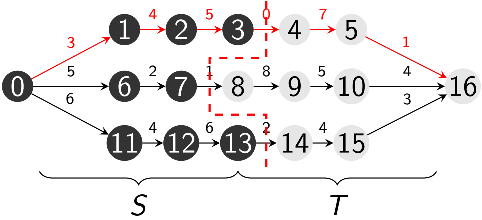


Figure: Min-cut (S, T) of a graph G

- We want to split the graph in half, to reuse algorithms on in-trees and out-trees

Reducing the cost along chain 1

Linear Arrangement (*multiple data*) on pumpkins

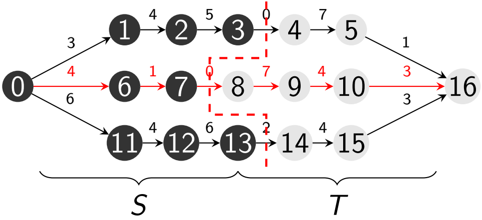


Figure: Min-cut (S, T) of a graph G

- We want to split the graph in half, to reuse algorithms on in-trees and out-trees

Reducing the cost along chain 2

Linear Arrangement (*multiple data*) on pumpkins

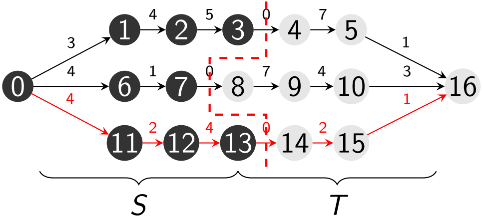


Figure: Min-cut (S, T) of a graph G

- We want to split the graph in half, to reuse algorithms on in-trees and out-trees

Reducing the cost along chain 3

Linear Arrangement (*multiple data*) on pumpkins

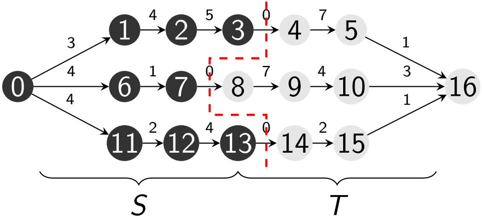
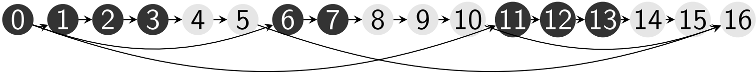


Figure: Min-cut (S, T) of a graph G



(a) Lexicographic schedule ϕ

Linear Arrangement (*multiple data*) on pumpkins

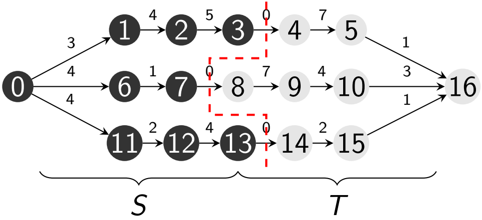
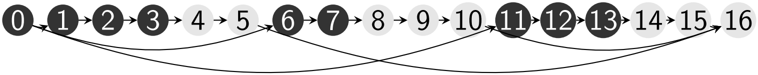
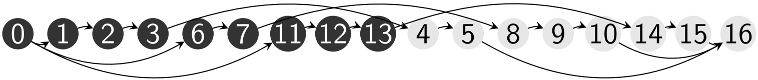


Figure: Min-cut (S, T) of a graph G



(a) Lexicographic schedule ϕ



(b) New schedule $\phi' = \phi[S] + \phi[T]$

Linear Arrangement (*multiple data*) on pumpkins

- Finding the min cut: $\mathcal{O}(n)$
- Solving Linear Arrangement on in-tree and out-tree: $\mathcal{O}(n \log(n))$
- Gives an $\mathcal{O}(n \log(n))$ algorithm

Sum Cut (*single data*) on pumpkins

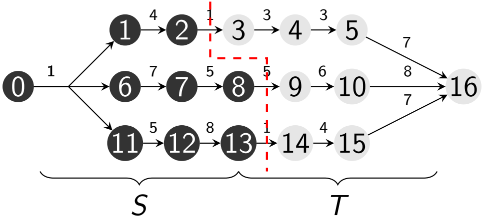


Figure: Cut (S, T) of a graph G

- We can't split at the min-cut like before

Sum Cut (*single data*) on pumpkins

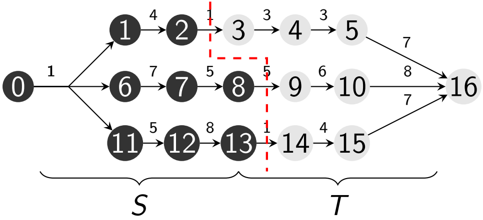


Figure: Cut (S, T) of a graph G

- We can't split at the min-cut like before
- Transform our graph into another pumpkin in the *multiple data* model

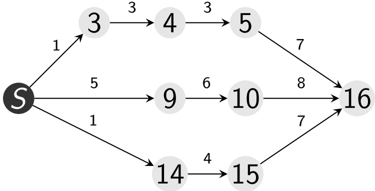


Figure: G'

Sum Cut (*single data*) on pumpkins

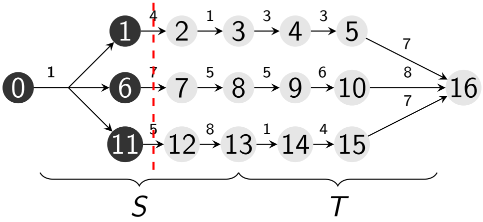


Figure: Cut (S, T) of a graph G

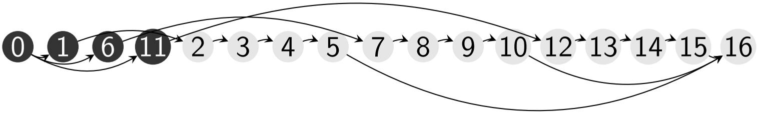


Figure: Schedule transformation with a cut

Sum Cut (*single data*) on pumpkins

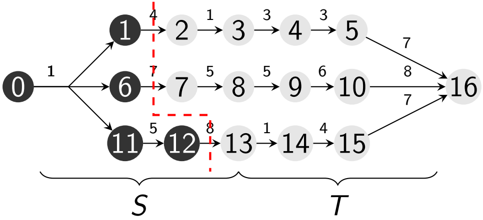


Figure: Cut (S, T) of a graph G

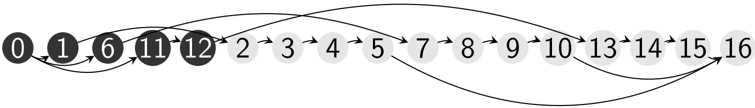


Figure: Schedule transformation with a cut

Sum Cut (*single data*) on pumpkins

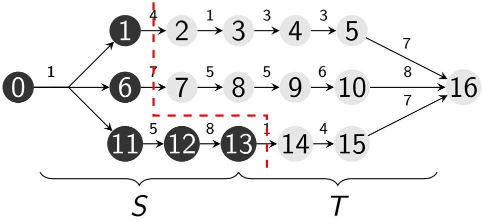


Figure: Cut (S, T) of a graph G

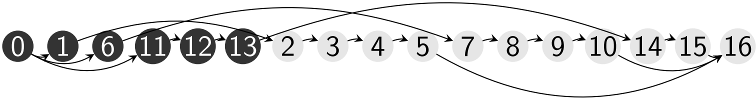


Figure: Schedule transformation with a cut

Sum Cut (*single data*) on pumpkins

- $\mathcal{O}\left(n^k\right)$ cuts
- Solving Linear Arrangement on in-tree and pumpkin: $\mathcal{O}(n \log(n))$
- Gives an $\mathcal{O}\left(n^k n \log(n)\right)$ algorithm

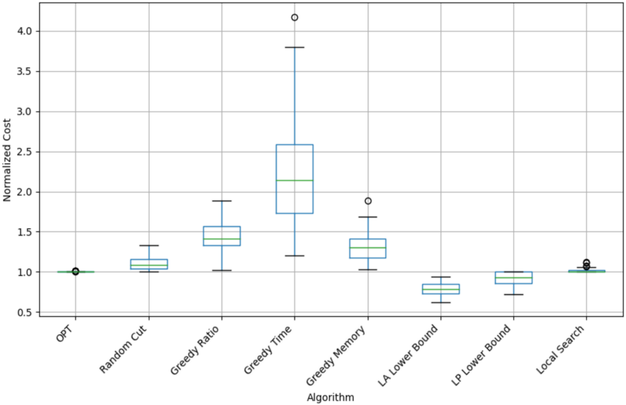
We look for heuristics, as we are competing against an $\mathcal{O}(n \log(n))$ algorithm.

- Greedy algorithms (greedy memory, greedy time)
- Random cut
- Local search

Experiments

- `C++` implementation of all the algorithms using the Boost Graph Library
- TaGaDa tool for graph generation
- Some other graphs from real world applications (trees)
- Compared with the state of the art algorithm for reducing the Peak memory usage

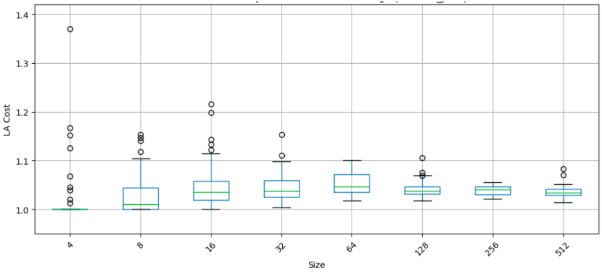
Experiments - Heuristics for WSC on pumpkins



Comparison of heuristics for WSC (*single data*) on random pumpkins

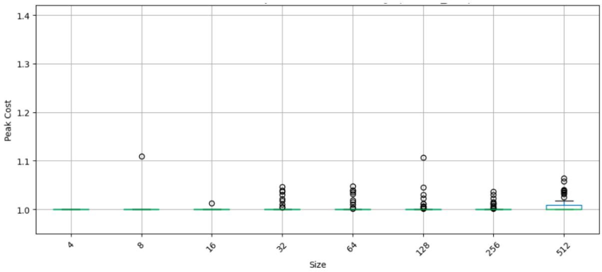
- Local search is on average within 5% of the optimal cost

Experiments - Comparing average and peak



Average memory cost of a schedule minimizing the peak memory

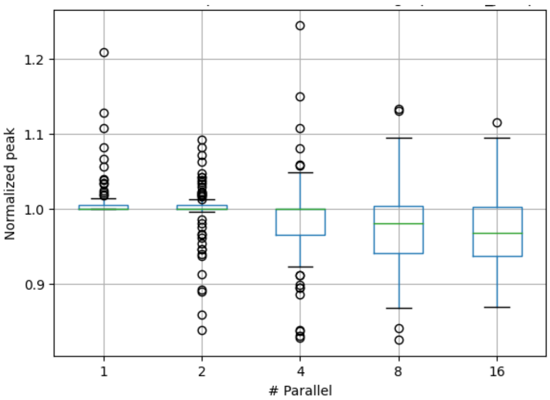
- 5% worse cost on average



Peak memory cost of a schedule minimizing the average memory

- Almost always the optimal cost

Experiments - Parallel Execution



- Multiple task graphs in parallel
- Minimizing the peak memory for each graph compared to minimizing the average memory for each graph
- The overall peak is up to 5% lower by using average rather than peak!

Conclusion

- Novel approach for scheduling task graphs to reduce average memory consumption.
- New exact algorithms and heuristics.
- Improved performance in parallel processing setups compared to focusing on the peak.

Questions

Thank you for your attention!
Any questions?