Minimum Spawning Tree
-
DPHPC

Th. Cambier R. Dang-Nhu Th. Dardinier C. Trassoudaine

ETH Zürich

October 2018

1. Problem definition
   - Concepts
   - Use cases
2. Algorithms
   - Prim
   - Kruskal
   - Borůvka (Sollin)
   - Others
3. Environment
4. Benchmarking
   - Reference, baseline, tools

Problem definition

The MST problem

Concepts

(Somewhat) realistic use-cases and input sets?

- $G(n, p)$
- Preferential attachment
    - Social networks

Algorithms

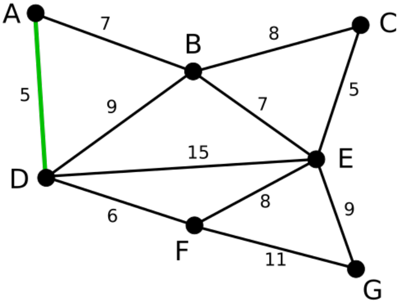
Prim

Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.

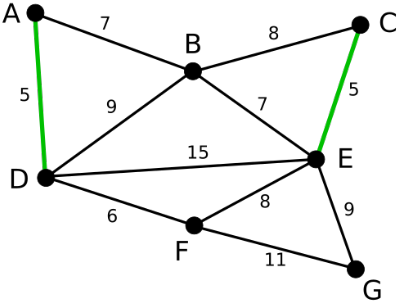
Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.



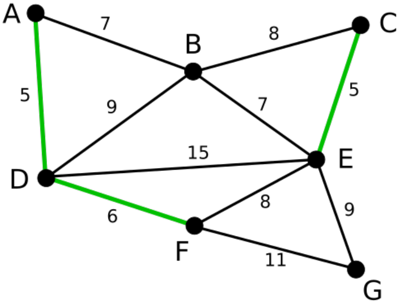
Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.



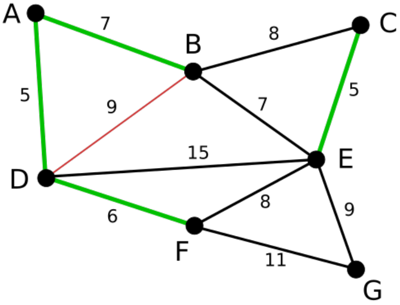
Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.



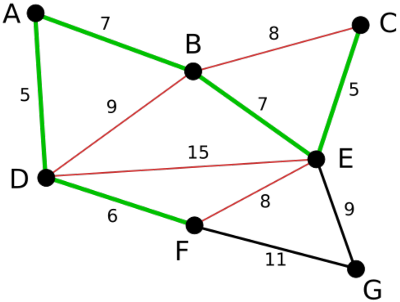
Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.



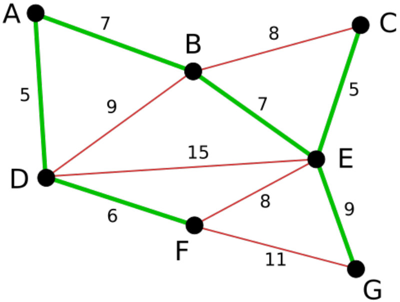
Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.



Sequential Kruskal

- Sort all edges by growing weight.
- For each edge: Add it to the MST if it doesn't create a cycle.



Data structure: Union-find

Represents the connected components of the graph given our MST.
Implementation with an array *parent* and 3 operations:

- Create: Every vertex is in its own component
    1. $parent[x] = x$.

Data structure: Union-find

Represents the connected components of the graph given our MST.
Implementation with an array *parent* and 3 operations:

- Create: Every vertex is in its own component
  1. $parent[x] = x$.
- Find(x): Find the component of this vertex.
  1. If $parent[x] \neq x$, then $parent[x] = find(parent[x])$
  2. Return $parent[x]$.

Data structure: Union-find

Represents the connected components of the graph given our MST.
Implementation with an array *parent* and 3 operations:

- Create: Every vertex is in its own component
  1. $parent[x] = x$.
- Find(x): Find the component of this vertex.
  1. If $parent[x] \neq x$, then $parent[x] = find(parent[x])$
  2. Return $parent[x]$.
- Union(x, y): Unite two components.
  1. $parent[find(x)] = find(y)$.

Sequential and parallel Kruskal

Sequential complexity:

- $O(E \log E)$: Sort all edges by growing weight.
- $O(E)$ (in practice): For each edge: Add it to the MST if it doesn't create a cycle.

We can parallelize the sort on $O(\log E)$ processors: $O(E)$ (in practice).

A better parallel approach: Filter-Kruskal

Borůvka (Sollin)

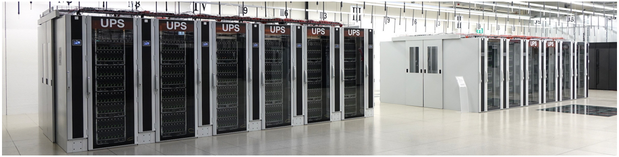
A few ideas

Correctness

How to verify correctness of the parallelization?

Environment

Architecture



EULER Cluster

Xeon Ex, $x \in \{3, 5, 7\}$
x86_64 architecture

Source : `https://scicomp.ethz.ch/wiki/Euler`

Tools

- CMake
  v3.3+
- C++11
  GCC v4.9.2+
- OpenMPI (shared memory)
  v1.6.5+

Benchmarking

Tools

- **Measures** : LibSciBench library
- **Interpretation** :
  - LibSciBench's R scripts
  - (Custom python scripts)

Ref : https://spcl.inf.ethz.ch/Research/Performance/LibLSB/

Baseline

https://en.wikipedia.org/wiki/Otakar_Bor%C5%AFvka

Borůvka's serial algorithm
$O(E \cdot log(V))$