Scalable Parameterised Algorithms for two Steiner Problems

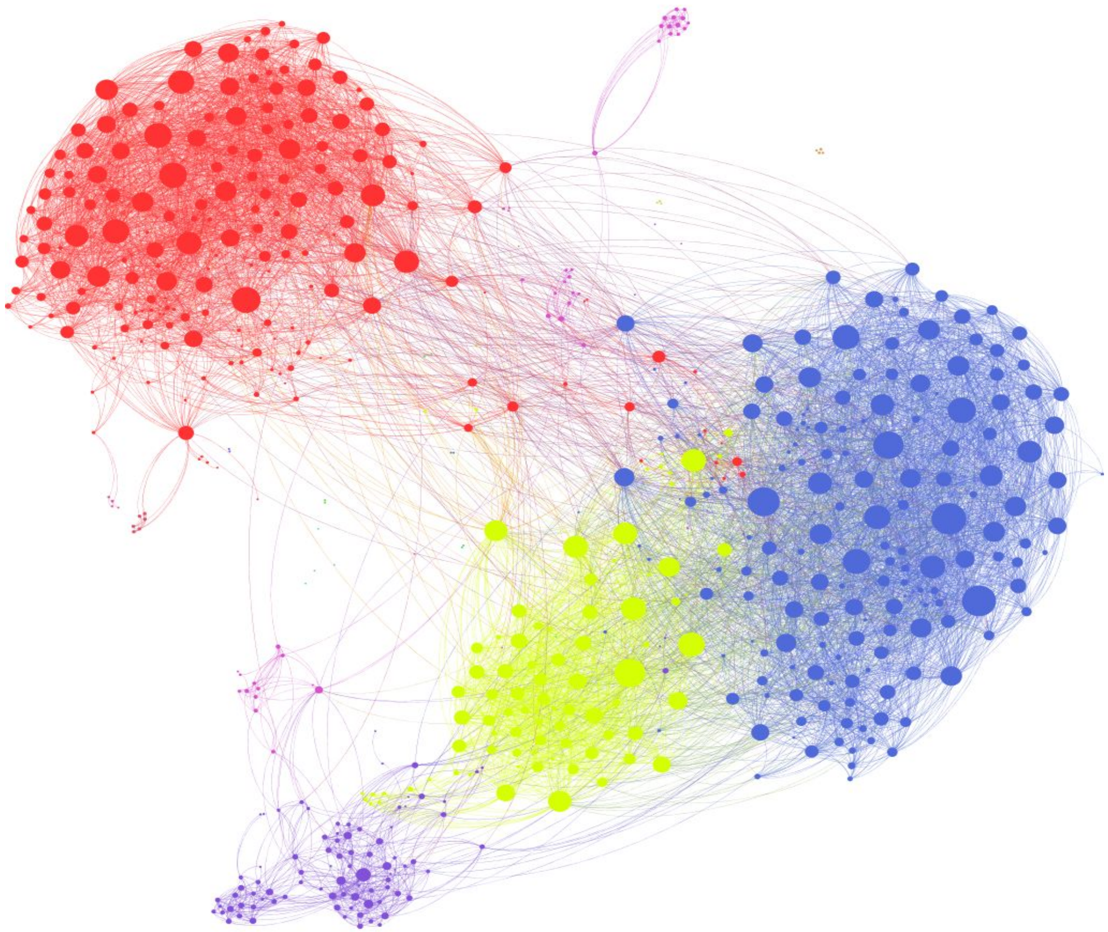
Suhas Thejaswi Muniyappa

suhas.muniyappa (at) aalto.fi

Aalto University
School of Science

Supervisor: Professor Petteri Kaski

A real-world problem



- Facebook network
- Each color represents a group.
- A minimum-weight subgraph connecting at least one vertex from each color.

Image source: https://griffsgraphs.wordpress.com/2012/07/02/a-facebook-network/

Outline

Introduction

- The Steiner problem
- The group Steiner problem

Algorithms

- Dreyfus–Wagner algorithm
- Erickson–Monma–Veinott algorithm
- Reduction for solving the group Steiner problem

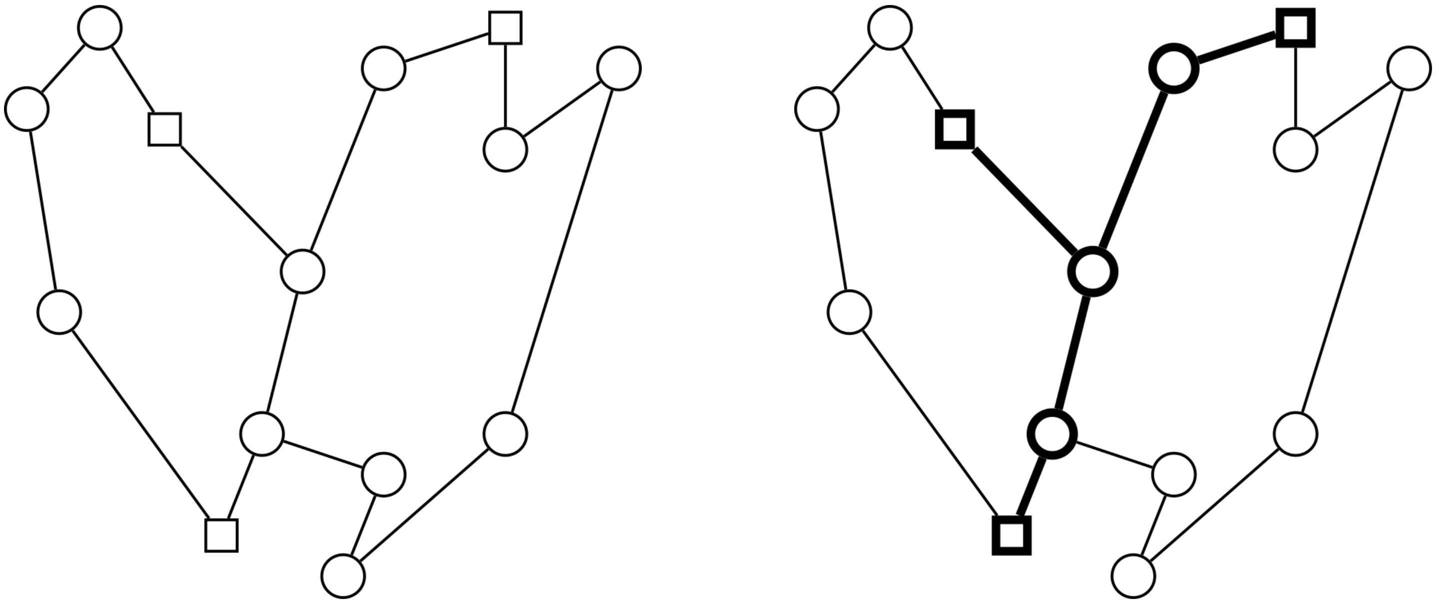
Implementation

- Challenges
- Implementation of Erickson–Monma–Veinott algorithm

Experimental results

- Scaling
- Parallelisation speed-up

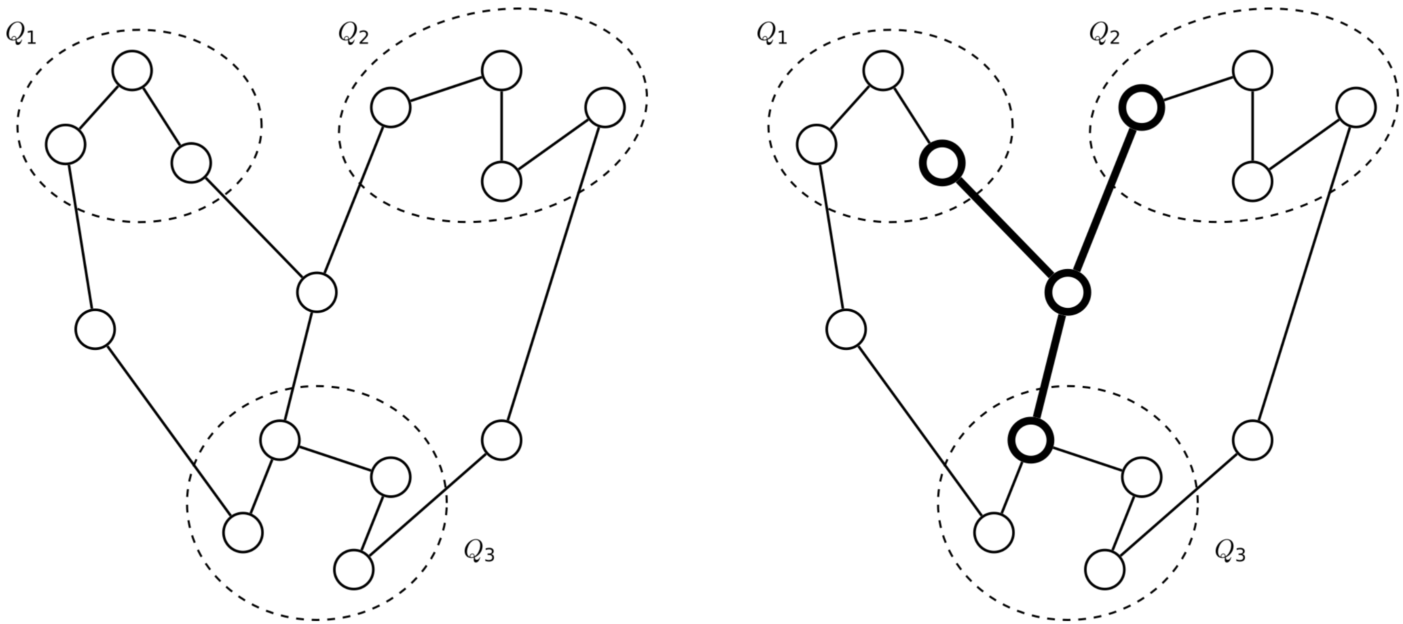
The Steiner problem



Rectangles are terminal vertices and circles are non-terminal vertices.

A minimum subgraph connecting all the terminals.

The group Steiner problem



A minimum subgraph connecting at least one vertex from each group.

Parameterised algorithms for the Steiner problem

NP-completeness

- The Steiner problem
  - Karp, 1972
- The group Steiner problem
  - Ihler, Discrete Applied Mathematics 1999

Parameterised algorithms

- Dreyfus and Wagner, 1972
- Erickson, Monma and Veinott, 1987

Dreyfus–Wagner algorithm

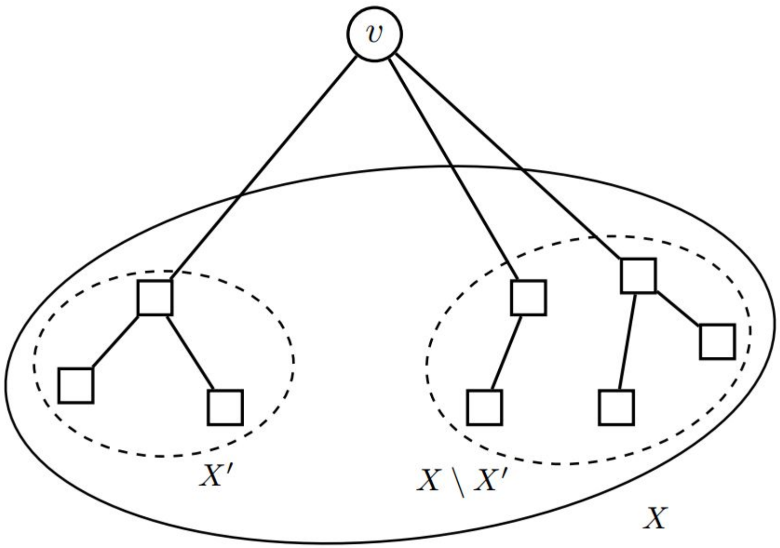
- Networks, 1972
- Dynamic-programing algorithm
- Fixed parameter tractable
- Optimal-decomposition property
- Exponential space complexity
- Time complexity, $O(3^k n + 2^k n^2 + n (n \log n + m))$

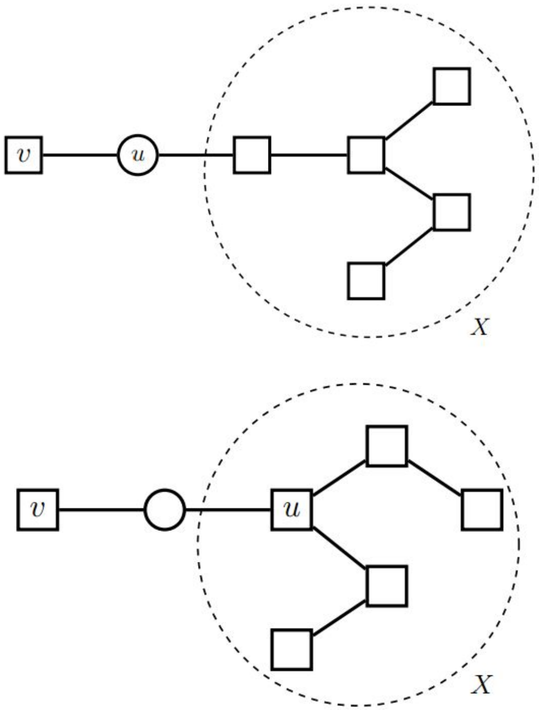
- Graph $N = (V, E, w)$
- Terminal set $K \subseteq V$
- Subset $X \subset K$
- Vertex $v \in V$

Cost of a Steiner tree connecting $X \cup \{v\}$

- $g_v(X)$, if $\deg(v) \geq 2$
- $f_v(X)$, otherwise

Optimal-decomposition property





$$g_v(X) = \min_{\emptyset \neq X' \subset X} \{f_v(X') + f_v(X \setminus X')\}$$

$$f_v(X) = \min\{ \min_{u \in X} \{d(v,u) + f_u(X \setminus \{u\})\}, \min_{u \in V \setminus X} \{d(v,u) + g_u(X)\}\}.$$

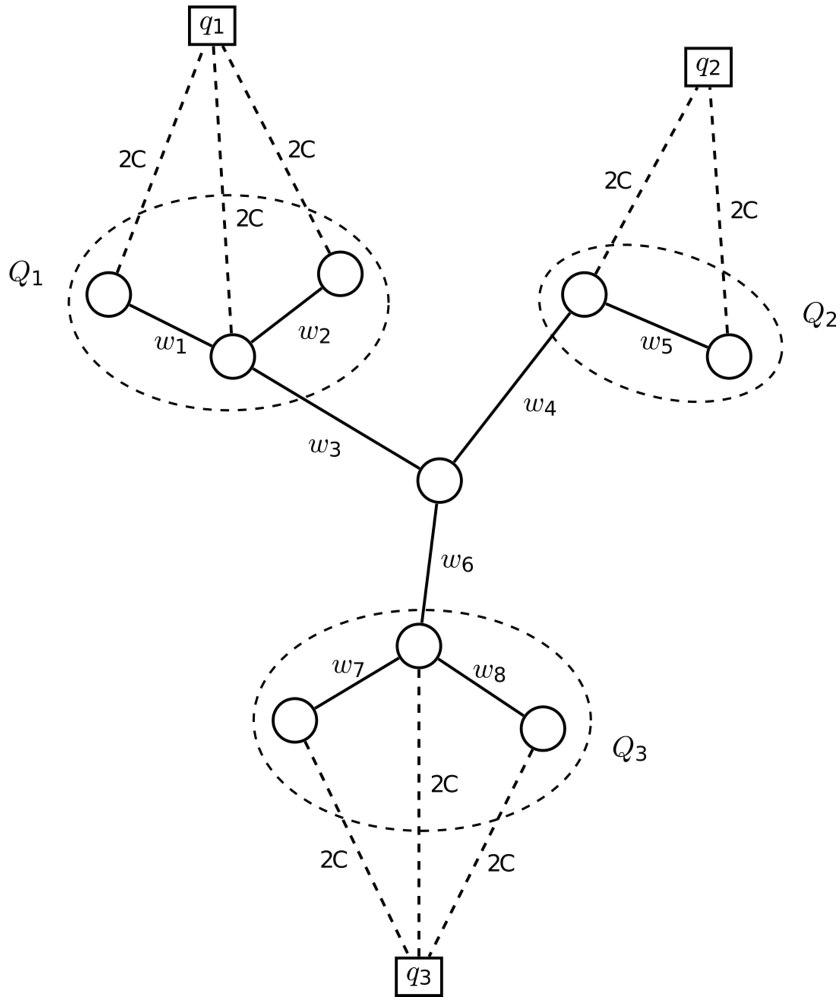
Erickson–Monma–Veinott Algorithm

- Mathematics of Operations Research, 1987
- Improvement of Dreyfus–Wagner algorithm
- Three improvements
  - Split set X only when $v \in V \setminus X$
  - Restrict computations to subset with terminals
  - Compute shortest path on demand
- Edge-linear time algorithm
- Time complexity, $O(3^k n + 2^k (n \log n + m))$
- Exponential space complexity, $O(2^k n + n + m)$

$$g_v(X) = \min_{\emptyset \neq X' \subset X} \{ f_v(X') + f_v(X \setminus X') \}$$

$$f_v(X) = \min_{u \in V \setminus X} \{ d(v, u) + g_u(X) \}.$$

Solving the group Steiner problem



Reducing the group Steiner problem to the Steiner problem.

- A network $N = (V, E, w)$
- $C = W(N)$
- Voß, Balkan conference on Operational Research, 1990.
- Restated by Duin *et al*. in 2004.
- Most algorithms of the Steiner problem can be used to solve the group Steiner problem.

Implementation challenges

Objective: Scaling for graphs with large number of edges

Memory consumption

- Space complexity $O(2^k n + n + m)$
- Natural-bit representation for subsets

- n = number of vertices
- m = number of edges
- k = number of terminals

Graph traversal and memory interface

- Arbitrary memory access pattern
- Less cache locality for large graphs
- Array of arrays representation for graphs

Parallel execution

- Single core cannot saturate the memory bandwidth
- Parallelisation over subsets of the terminal set K
- 2^k executions of Dijkstra subroutine

Implementation of Erickson–Monma–Veinott Algorithm

$$g_v(X) = \min_{\emptyset \neq X' \subset X} \{f_v(X') + f_v(X \backslash X')\}$$

$$f_v(X) = \min\{ \min_{u \in X}\{d(u,v) + f_u(X \setminus \{u\})\}, \min_{u \in V \setminus X}\{d(v,u) + g_u(X)\}\}.$$

- 2^k executions of Dijkstra subroutine
- Parallelisation over subsets
- Bit-twiddling hacks for subset generation
- Same memory used for $g_v(X)$ and $f_v(X)$
- One-dimensional array of size $2^k n$

```
index_t X     = X_a[i];
index_t *f_X = f_v + FV_INDEX(0, n, k, X);
index_t Xd    = 0;
// bit twiddling hacks: generating proper subsets of X
for(Xd = X & (Xd - X); Xd != X; Xd = X & (Xd - X)) {
    index_t X_Xd    = (X & ~Xd); // X - X'
    index_t *f_Xd   = f_v + FV_INDEX(0, n, k, Xd);
    index_t *f_X_Xd = f_v + FV_INDEX(0, n, k, X_Xd);
    for(index_t v = 0; v < n; v++)
        f_X[v] = MIN(f_X[v], f_Xd[v] + f_X_Xd[v])
}
// graph reconstruction
index_t s = n + th; index_t ps = pos[s];
index_t *adj_s = adj + (ps+1);
for(index_t u = 0; u < n; u++)
    adj_s[2*u+1] = f_X[u];'
for(index_t q = 0; q < k; q++) {
    if(!(X & (1<<q))) continue;
    index_t u = kk[q]; index_t X_u = (X & ~(1<<q));
    index_t i_X_u = FV_INDEX(u, n, k, X_u);
    adj_s[2*u+1]  = f_v[i_X_u];
}
dijkstra(s, n+nt, pos, adj, d_th, visit_th);
for(index_t v = 0; v < n; v++)
    f_X[v] = d_th[v];
```

Experiments

We measure the runtime, memory bandwidth and peak-memory usage of the experiments.

- Regular graphs
- n = number of vertices
- m = number of edges
- k = number of terminals
- d = degree

Dijkstra's algorithm

- Edge-linear scaling
- Binary versus Fibonacci heaps

Erickson, Monma and Veinott algorithm

- Edge-linear scaling (fixed k)
- Parallelisation speedup
- Scaling up to a billion edges
- Exponential scaling of number of terminals (fixed m)

Hardware

Mid-memory configuration

- 2 x 2.5 GHz Intel Xeon E5-2680v3 CPU (Haswell microarchitecture, 24 cores, 12 cores/CPU, no hyper-threading, 30 MiB L3 cache)
- 128 GiB of main memory (8 x 16 GiB DDR4-2133, ECC enabled)

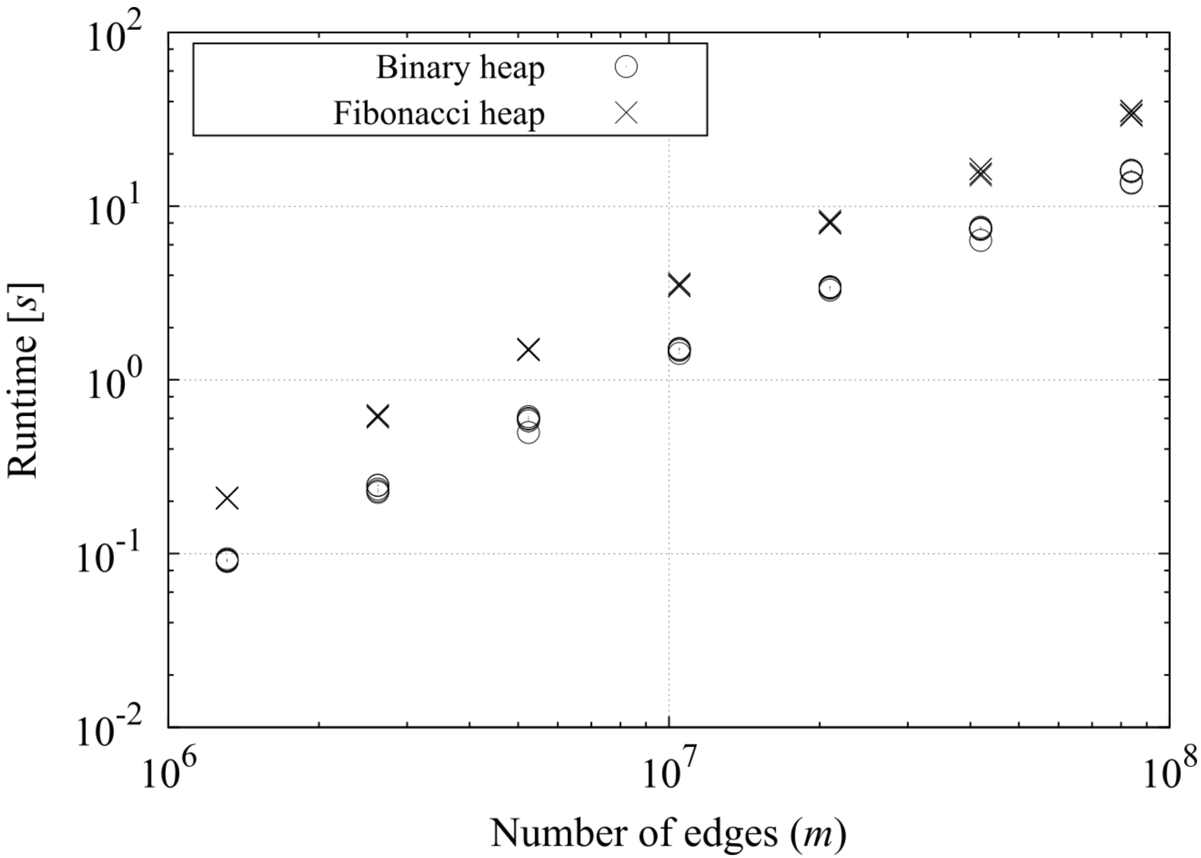
Large-memory configuration

- 2 x 2.5 GHz Intel Xeon E5-2680v3 CPU (Haswell microarchitecture, 24 cores, 12 cores/CPU, no hyper-threading, 30 MiB L3 cache)
- 256 GiB of main memory (16 x 16 GiB DDR4-2133, ECC enabled)

Huge-memory configuration

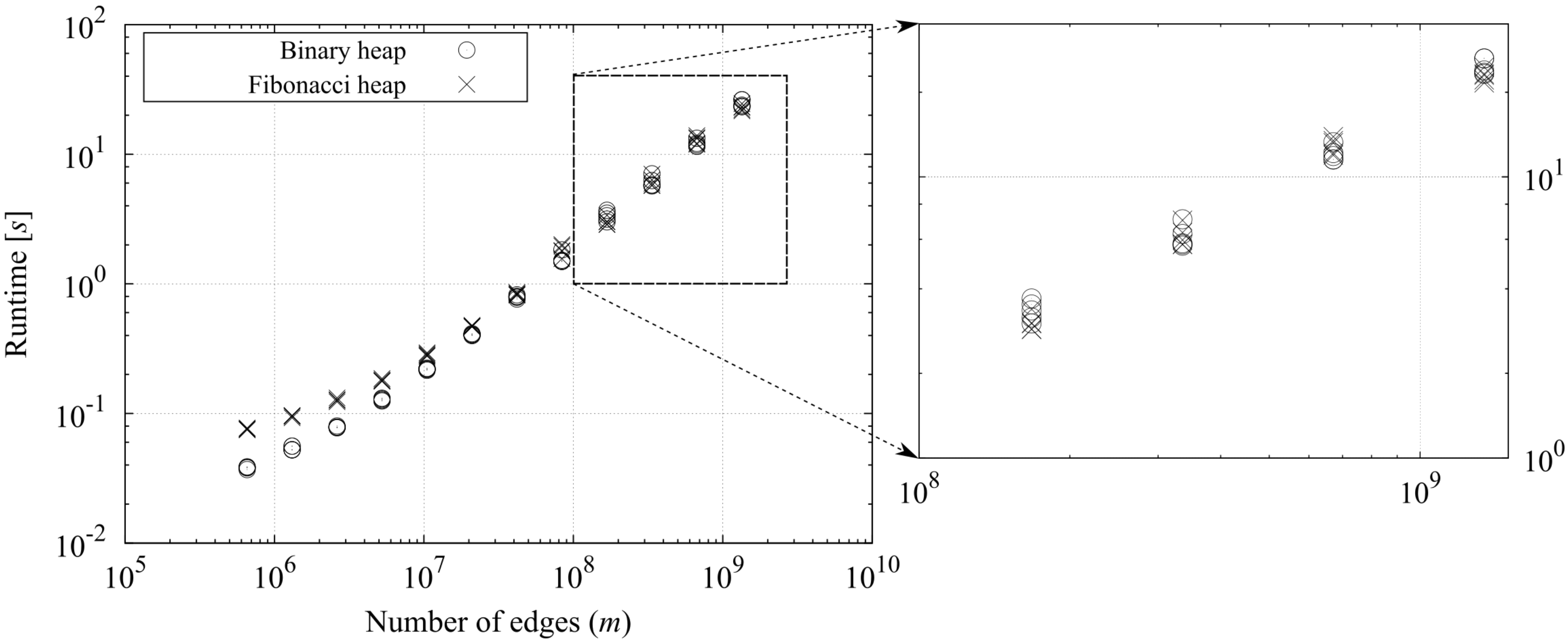
- 4 x 2.8 GHz Intel Xeon E7-8891v3 CPU (Haswell microarchitecture, 40 cores, 10 cores/CPU, no hyper-threading, 45 MiB L3 cache)
- 1536 GiB of main memory (96 x 16 GiB DDR4-2133, ECC enabled)

Edge scaling of Dijkstra's algorithm



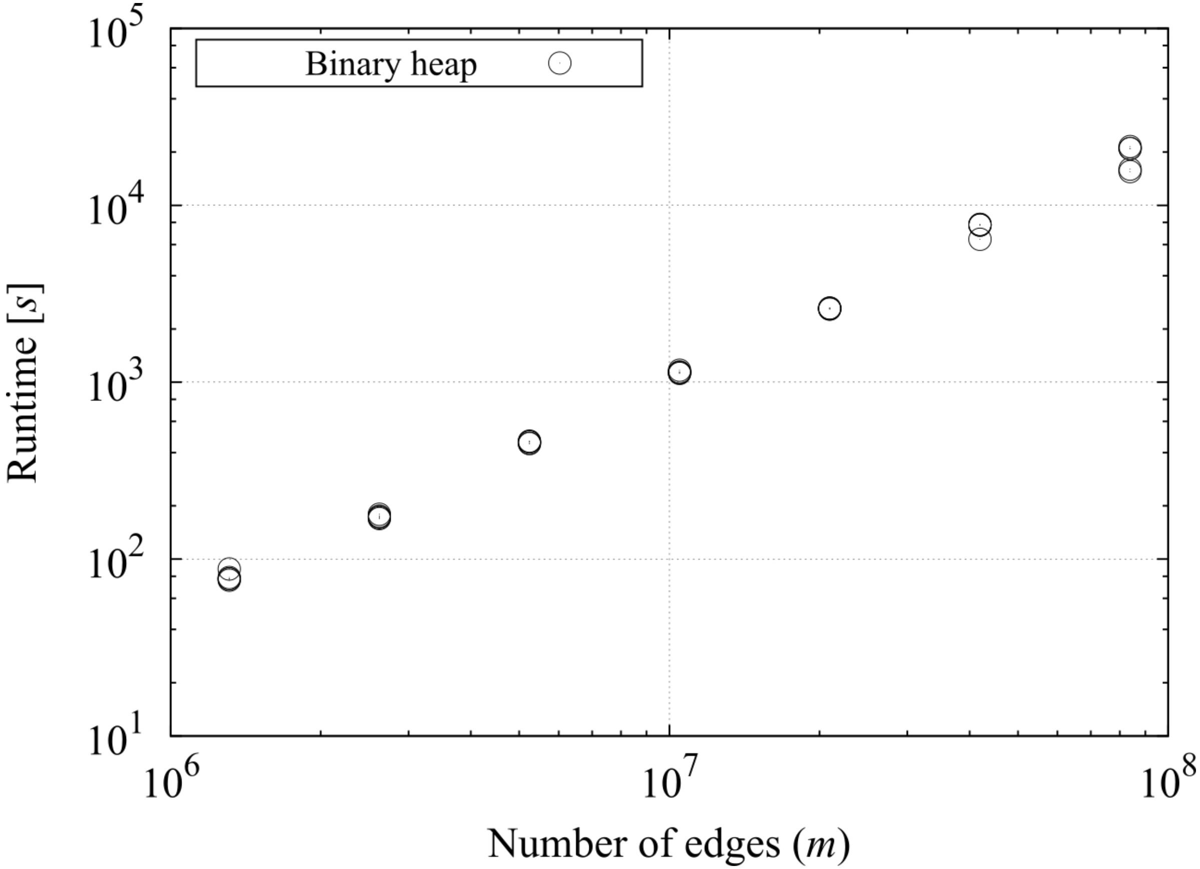
- Five random (d-regular) graphs for each value of m
- $d = 20$ (fixed)
- Logarithmic axis
- Mid-memory configuration

Binary heaps versus Fibonacci heaps



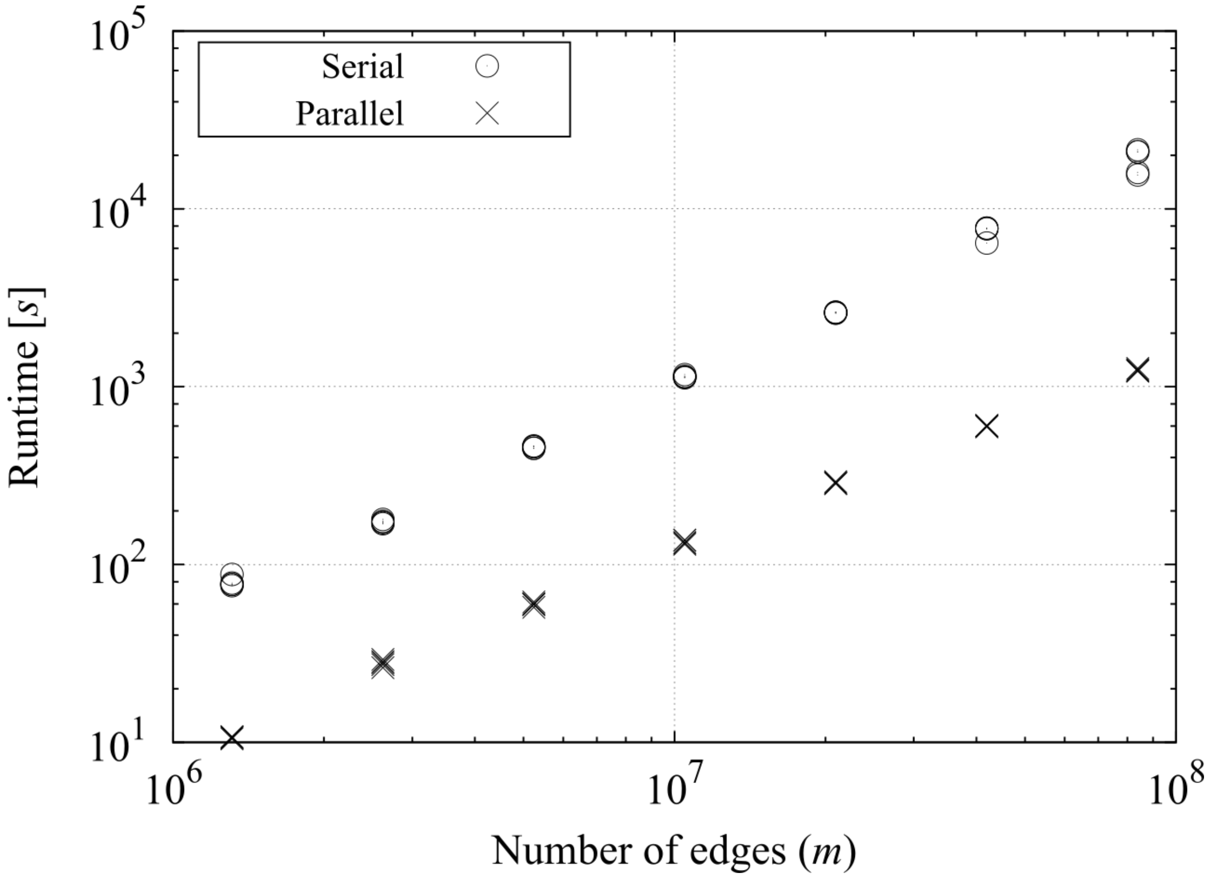
- Five random (d-regular) graphs for each value of m
- Mid-memory configuration
- n = 65536 (fixed)
- d = 20, 40, 80, …, 40960
- Logarithmic axis

Edge scaling



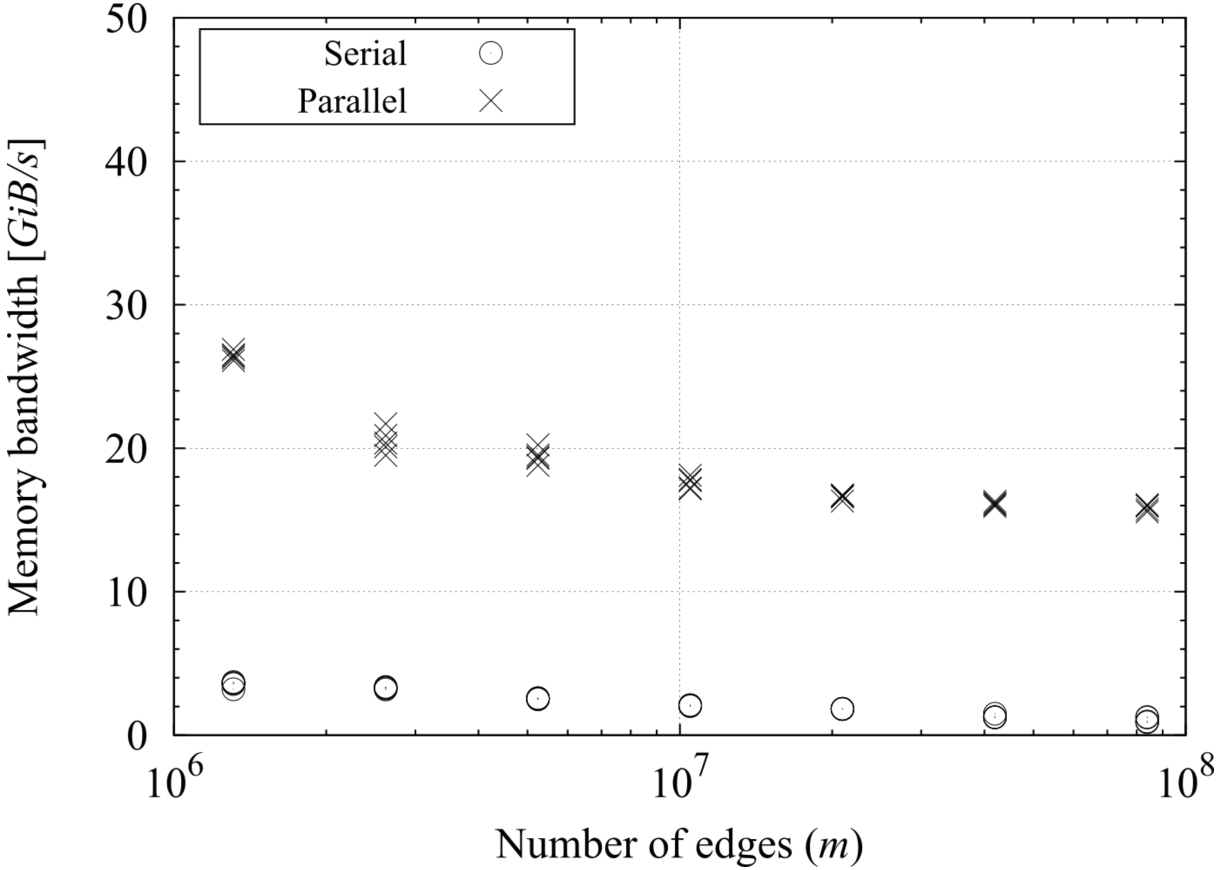
- Five random (d-regular) graphs for each value of m
- $d = 20$ (fixed)
- $k = 10$ (fixed)
- Logarithmic axis
- Mid-memory configuration

Parallelisation speed-up



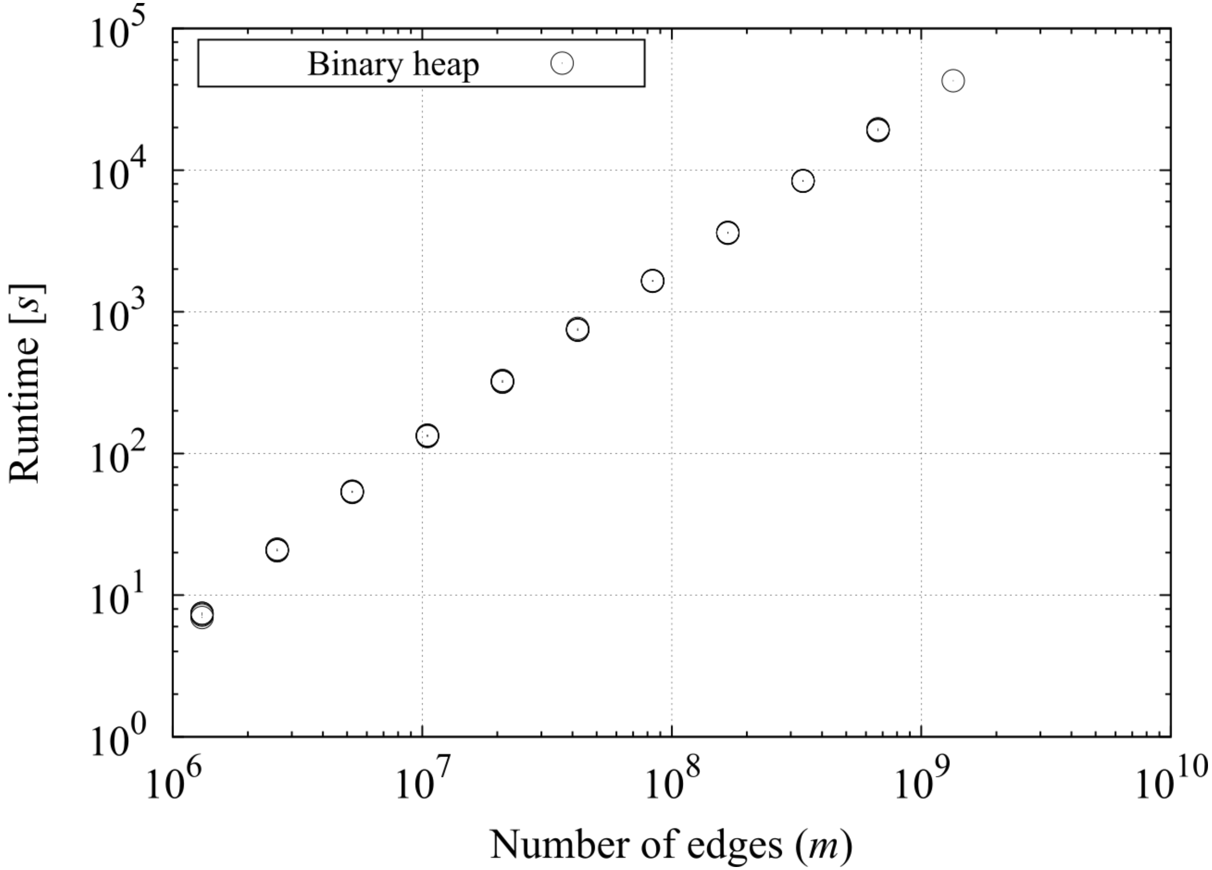
- Five random (d-regular) graphs for each value of m
- $d = 20$ (fixed)
- $k = 10$ (fixed)
- Logarithmic axis
- Speed-up is fifteen times for $m = 10^8$
- Mid-memory configuration

Memory bandwidth



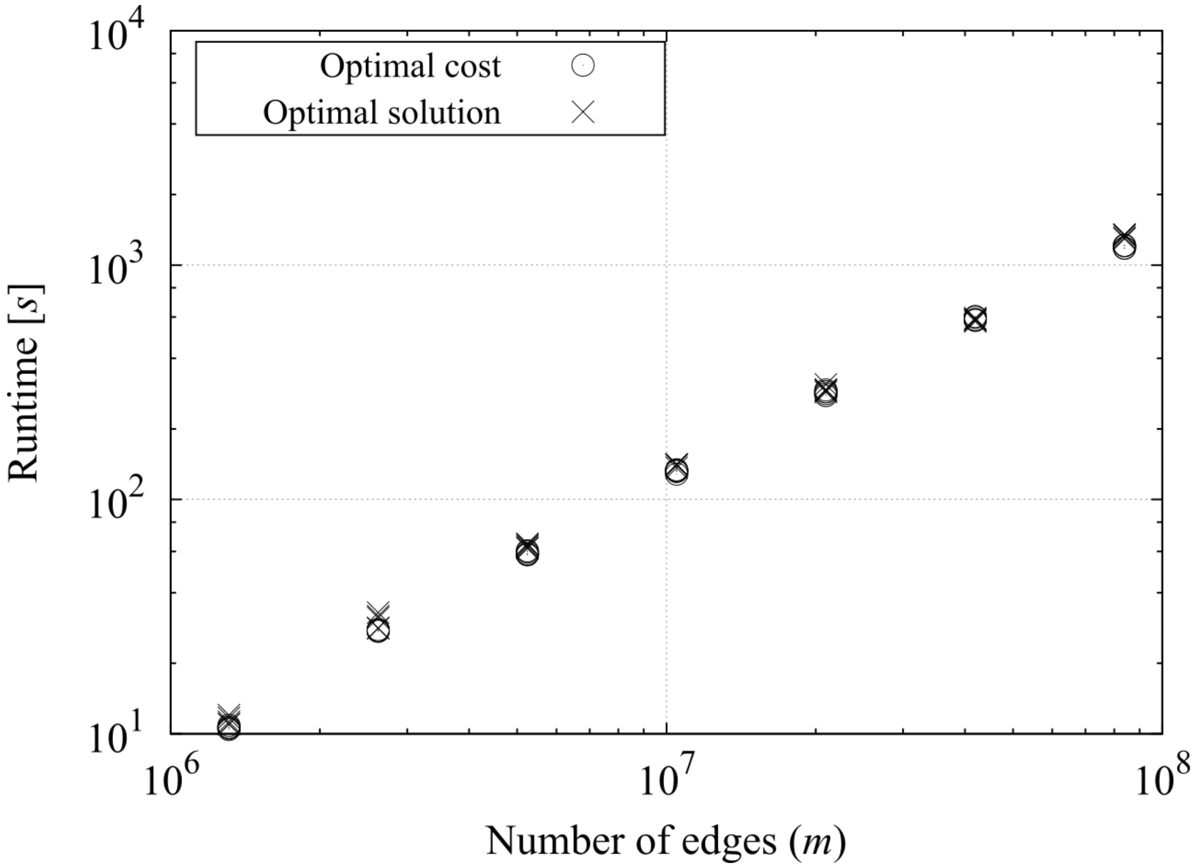
- Five random (d-regular) graphs for each value of m
- $d = 20$ (fixed)
- $k = 10$ (fixed)
- Logarithmic x-axis
- Bandwidth speed-up is fifteen times for $m = 10^8$
- Mid-memory configuration

Scaling up to a billion edges



- Five random (d-regular) graphs for each value of m
- $d = 20$ (fixed)
- $k = 10$ (fixed)
- Logarithmic axis
- Huge-memory configuration

Optimal cost versus optimal solution



- Five random (d-regular) graphs for each value of m
- $d = 20$ (fixed)
- $k = 10$ (fixed)
- Logarithmic axis
- Binary heap
- Large-memory configuration

Summary

Scaling on a single compute node

- Up to a billion edges for small number of terminals
- Up to twenty terminals for small number of edges

Parallel implementation

- Fifteen times faster than its serial counterpart for large graphs
- Memory bandwidth is twice the read random cache lines experiment

Heap implementations

- Binary heap perform better than Fibonacci heap
- Fibonacci heap can compete for dense graphs

Future work

- Performance is limited by memory bandwidth
- Using GPUs to achieve better memory bandwidth

Questions ?

Thank you :)