Improving Kruskal's Algorithm

Vitaly Osipov, Peter Sanders, Johannes Singler

Universität Karlsruhe (TH)

Motivation

What is the best (practical) algorithm for Minimum Spanning Trees?

- Kruskal? for very sparse graphs
- Jarník–Prim? else
- Karger–Klein–Tarjan, Chazelle, Pettie–Ramachandran, . . . ? too complicated

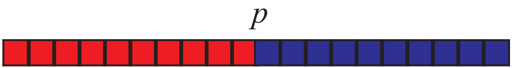
Kruskal's Algorithm

Procedure kruskal(E, T : Sequence **of** Edge, P : UnionFind)
  sort E by increasing edge weight
  foreach $\{u, v\} \in E$ **do**
    if u and v are in different components of P **then**
      add edge $\{u, v\}$ to T
      join the partitions of u and v in P

Time $O((n + m) \log m)$

Quick-Kruskal (Moret–Shapiro, Paredes–Navarro)

Procedure qKruskal(E, T : Sequence **of** Edge, P : UnionFind)
  if $m \leq$ kruskalThreshold($n, m, |T|$) **then** kruskal(E, T, P)
  else
    pick a pivot $p \in E$
    $E_{\leq} := \langle e \in E : e \leq p \rangle$
    $E_{>} := \langle e \in E : e > p \rangle$
    qKruskal($E_{\leq}, T, P$)
    if $|T| = n - 1$ **then** exit
    qKruskal($E_{>}, T, P$)



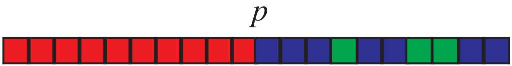
Time:

- $O\left(m + n\log^2 n\right)$ – random graphs, random edge weights
- $\Theta\left((n+m)\log m\right)$ if there is any heavy MST edge, e.g., Lollipop graph with random edge weights

Filter-Kruskal

Procedure filterKruskal(E, T : Sequence **of** Edge, P : UnionFind)
  if $m \leq$ kruskalThreshold($n, m, |T|$) **then** kruskal(E, T, P)
  else
    pick a pivot $p \in E$
    $E_{\leq} := \langle e \in E : e \leq p \rangle$
    $E_{>} := \langle e \in E : e > p \rangle$
    qKruskal($E_{\leq}, T, P$)
    if $|T| = n - 1$ **then** exit
    $E_{>} :=$ filter($E_{>}, P$)
    qKruskal($E_{>}, T, P$)

Function filter(E)
  return $\langle \{u, v\} \in E : u, v$ are in different components of $P \rangle$



Analysis – Arbitrary Graph, Random Weights

Lemma: It suffices to count edge comparisons

Chan's sampling lemma:

r lightest edges processed $\Rightarrow \mathbb{P}\left[e \in E_{>} \text{ survives filtering}\right] \leq \dfrac{n}{r}$

Optimistic Analysis:

Assume edge with rank i is filtered when $r = i$.

$$\mathbb{E}[\#\text{survivors}] \leq n + \sum_{i>n} \frac{n}{i} = \Theta\left(n \log \frac{m}{n}\right)$$

Partitioning m edges and sorting $n \log \frac{m}{n}$ edges:

$$\Omega(m + n \log n \log \frac{m}{n})$$

Analysis – Upper Bound (Outline)

Idea: Generalize textbook analysis of quicksort.
0–1-RV $X_{ij} := 1$ iff edges with ranks i and j are compared.

$$\mathrm{E}\left[\#\text{comparisons}\right] = \sum_{i \le m} \sum_{i < j \le m} \mathbb{P}\left[X_{ij} = 1\right] \quad .$$

strengthen bound on $\mathbb{P}\left[X_{ij} = 1\right]$ using Chan's sampling lemma.

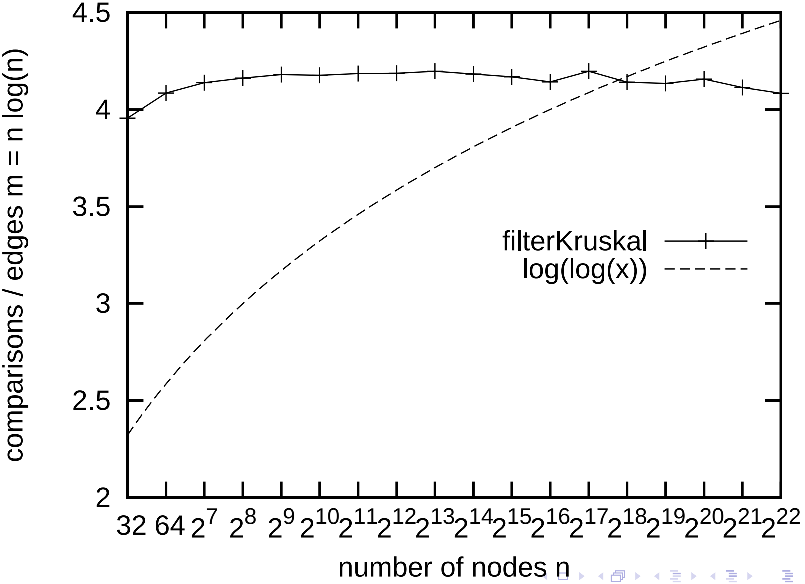
...

$\rightsquigarrow$

...

$O(m + n \log n \log \frac{m}{n})$ expected comparisons

Getting rid of the $\log m/n$? E.g., random graphs



Filter-Kruskal+

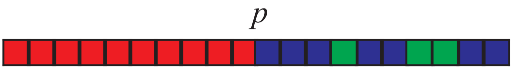
Function filter+(E, T, P)

$E' := \langle \{u, v\} \in E : u, v$ are in different components of $P\rangle$

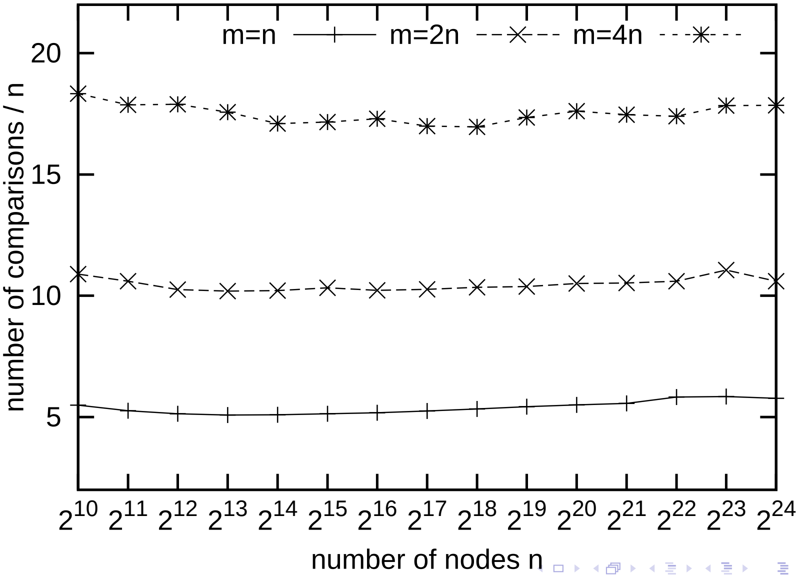
$T' := \langle \{u, v\} \in E' :$ u or v have degree one in comp. graph $\rangle$

$T := T \cup T'$

return $E' \setminus T'$



Linear Time? E.g., random graphs

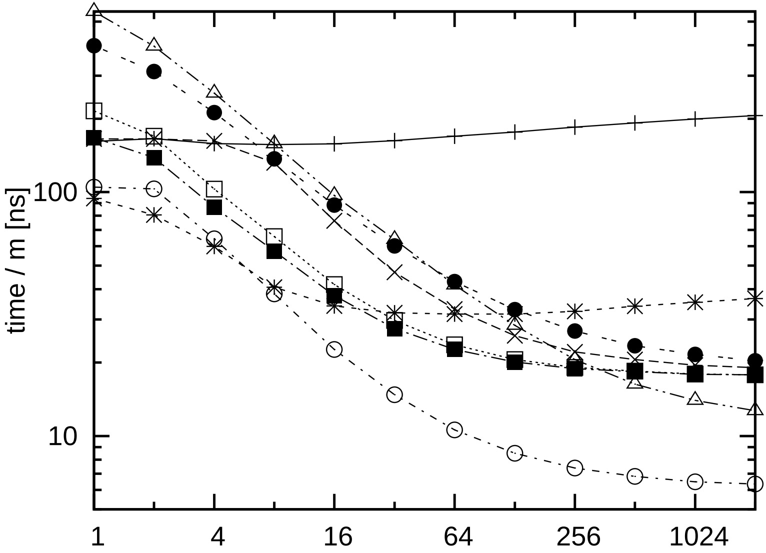


Parallelization

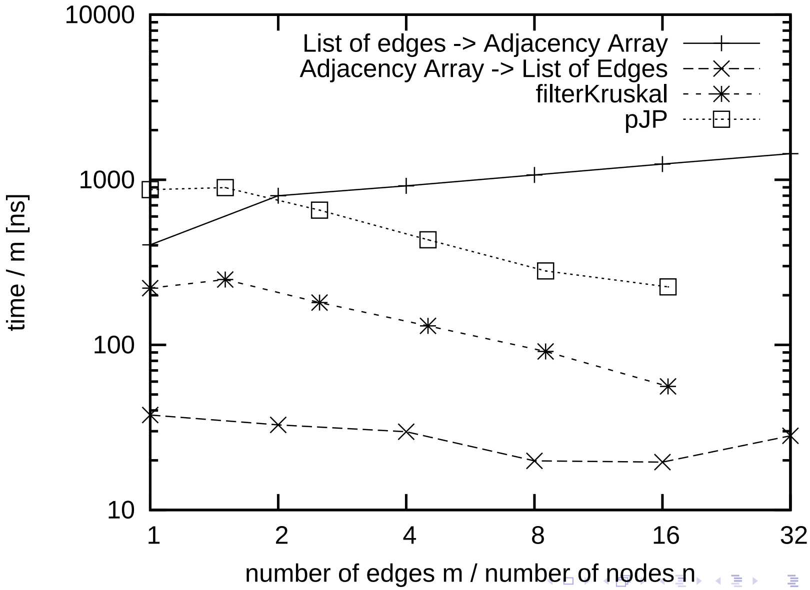
Procedure filterKruskal(E, T : Sequence **of** Edge, P : UnionFind)
  if $m \le$ kruskalThreshold($n, m, |T|$) **then**
    kruskal(E, T, P) // parallel sort
  else
    pick a pivot $p \in E$
    $E_{\le} := \langle e \in E : e \le p \rangle$ // parallel
    $E_{>} := \langle e \in E : e > p \rangle$ // partitioning
    qKruskal($E_{\le}, T, P$)
    if $|T| = n - 1$ **then** exit
    $E_{>} :=$ filter($E_{>}, P$) // parallel removeIf
    qKruskal($E_{>}, T, P$)

Easy: available in the Multi-Core Parallel STL (e.g. g++)

Running Time: Random graph with 2^{16} nodes



Graph Formatting: Random graph with 2^{22} nodes



Conclusions

filterKruskal improves on Kruskal and Jarník–Prim

- very simple
- often faster
- parallelizable
- needs only edge sequence

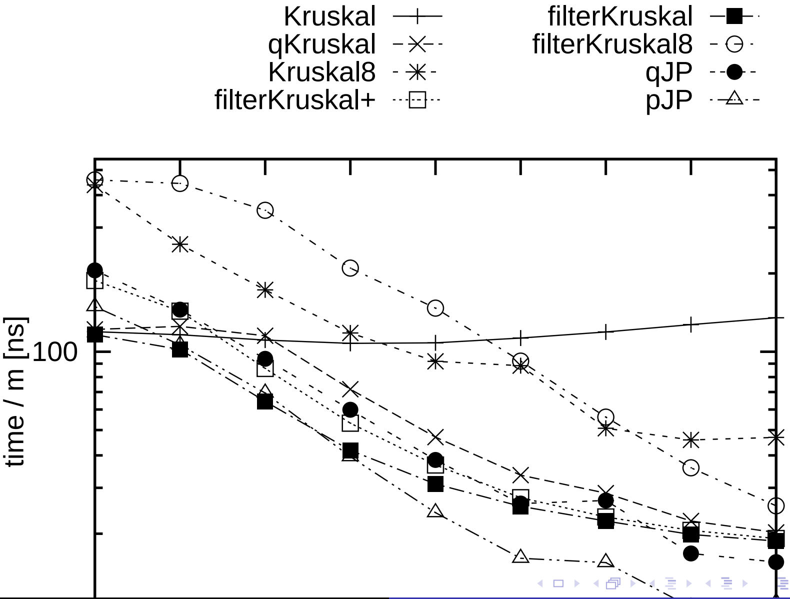
Todo: More real world inputs, tight analysis,...

Open Problem: What about filterKruskal+

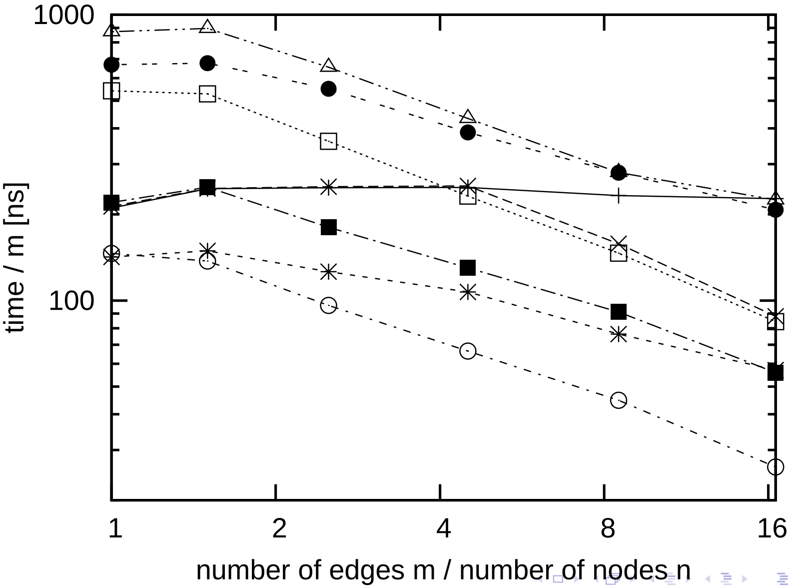
- provably linear time?
- scalable parallelization? Sequential component $\mathcal{O}\left(n^{0.6}\right)$?
- more efficient implementation

Thank you!

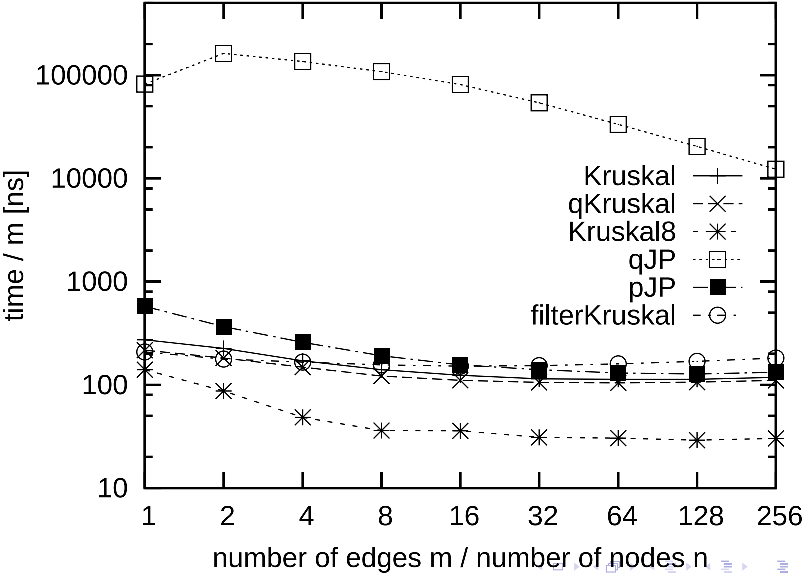
Random graph with 2^{10} nodes



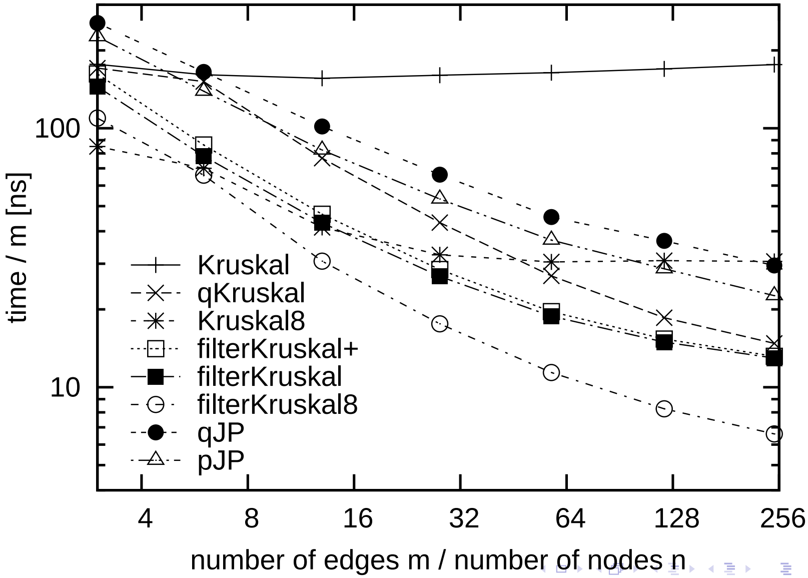
Random graph with 2^{22} nodes



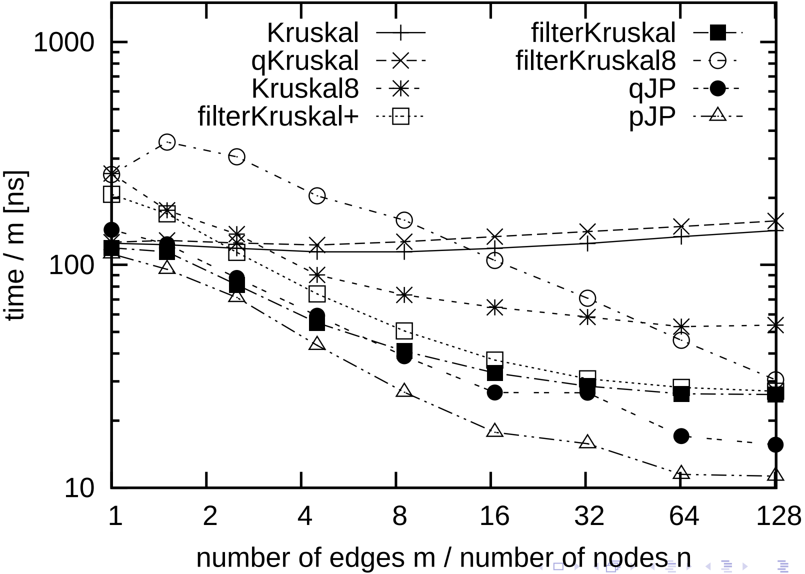
Random graph, $n = 2^{16}$, anti-Prim weights



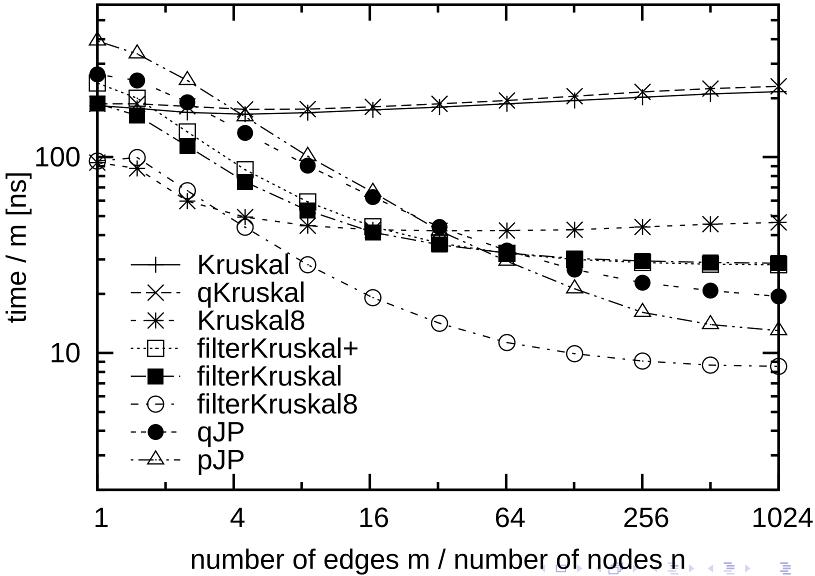
Random geometric graph with 2^{16} nodes



Lollipop graph with 2^{10} nodes



Lollipop graph with 2^{16} nodes



Lollipop graph with 2^{22} nodes

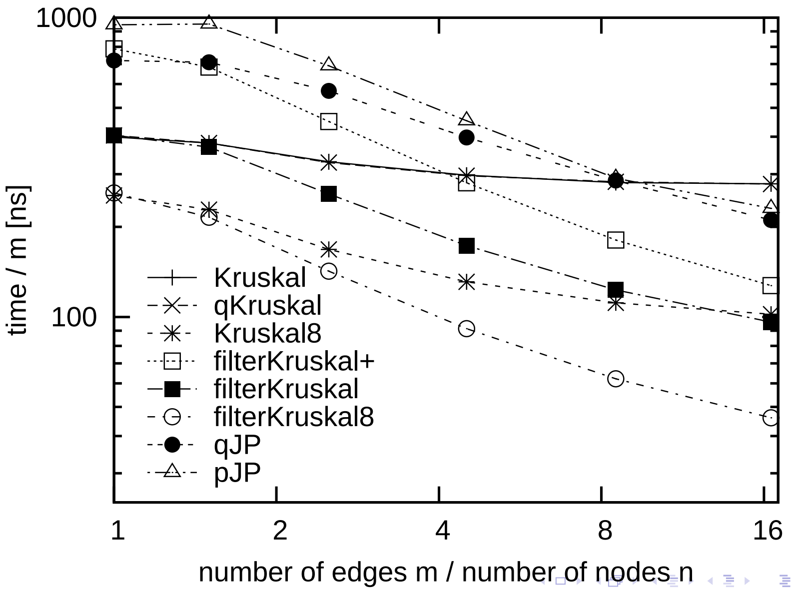


Image Segmentation Application