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
Kruskal’s Algorithm

**Procedure** `kruskal(E, T : Sequence of Edge, P : UnionFind)`

1. sort $E$ by increasing edge weight
2. **foreach** $(u, v) \in E$ **do**
   1. **if** $u$ and $v$ are in different components of $P$ **then**
      1. add edge $(u, v)$ to $T$
      2. join the partitions of $u$ and $v$ in $P$

Time $O((n + m) \log m)$
Quick-Kruskal

(Moret–Shapiro, Paredes–Navarro)

**Procedure** qKruskal($E, T : \text{Sequence of Edge}, P : \text{UnionFind}$)

if $m \leq \text{kruskalThreshold}(n, m, |T|)$ 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($E_{>}, T, P$)

$\text{Time:}$

- $O\left(m + n \log^2 n\right)$ – random graphs, random edge weights
- $\Theta((n + m) \log m)$ 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 $m \leq \text{kruskalThreshold}(n, m, |T|)$ 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
    $E_{>} := \text{filter}(E_{>}, P)$
    `qKruskal(E_{>}, T, P)`

**Function** `filter(E)`

return $\langle \{u, v\} \in E : u, v \text{ are in different components of } P \rangle$
Analysis – *Arbitrary* Graph, Random Weights

**Lemma:** It suffices to count edge comparisons

**Chan’s sampling lemma:**
r lightest edges processed ⇒ \( \mathbb{P}[e \in E_{>} \text{ survives filtering}] \leq \frac{n}{r} \)

**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:

\[
\Omega(m + n \log n \log \frac{m}{n})
\]
Idea: Generalize textbook analysis of quicksort.

0–1-RV $X_{ij} := 1$ iff edges with ranks $i$ and $j$ are compared.

$$E[\text{#comparisons}] = \sum_{i \leq m} \sum_{i < j \leq m} \mathbb{P}[X_{ij} = 1].$$

Strengthen bound on $\mathbb{P}[X_{ij} = 1]$ using Chan’s sampling lemma.

$\ldots$

$\ldots$

$O(m + n \log n \log \frac{m}{n})$ expected comparisons
Getting rid of the log $m/n$? E.g., random graphs

![Diagram showing the comparison of \( m = n \log(n) \) and \( \log(\log(x)) \) with increasing number of nodes. The diagram compares two lines: one representing \( m = n \log(n) \) and the other representing \( \log(\log(x)) \). The \( x \) axis represents the number of nodes, and the \( y \) axis represents the comparisons / edges. The graph shows that \( \log(\log(x)) \) grows more slowly than \( m = n \log(n) \) as \( n \) increases.

Vitaly Osipov, Johannes Singler, Peter Sanders

Improving Kruskal's Algorithm
Filter-Kruskal+

Function $\text{filter+}(E, T, P)$

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

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

$T := T \cup T'$

return $E' \setminus T'$
Linear Time? E.g., random graphs

Vitaly Osipov, Johannes Singler, Peter Sanders
Improving Kruskal's Algorithm
Parallelization

Procedure filterKruskal($E, T : \text{Sequence of Edge}, P : \text{UnionFind}$)

if $m \leq \text{kruskalThreshold}(n, m, |T|)$ then
    kruskal($E, T, P$)  // parallel sort
else
    pick a pivot $p \in E$
    $E_\leq := \langle e \in E : e \leq p \rangle$
    $E_\gt := \langle e \in E : e > p \rangle$
    qKruskal($E_\leq, T, P$)
    if $|T| = n - 1$ then exit
    $E_\gt := \text{filter}(E_\gt, P)$
    qKruskal($E_\gt, T, P$)  // parallel removeIf

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

- List of edges -> Adjacency Array
- Adjacency Array -> List of Edges
- filterKruskal
- pJP

Vitaly Osipov, Johannes Singler, Peter Sanders

Improving Kruskal’s Algorithm
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 $O(n^{0.6})$?
- more efficient implementation
Thank you!
Random graph with $2^{10}$ nodes

Vitaly Osipov, Johannes Singler, Peter Sanders

Improving Kruskal’s Algorithm
Random graph with $2^{22}$ nodes

Vitaly Osipov, Johannes Singler, Peter Sanders
Improving Kruskal’s Algorithm
Random graph, $n = 2^{16}$, anti-Prim weights
Random geometric graph with $2^{16}$ nodes

Vitaly Osipov, Johannes Singler, Peter Sanders

Improving Kruskal's Algorithm
Lollipop graph with $2^{10}$ nodes

Vitaly Osipov, Johannes Singler, Peter Sanders
Improving Kruskal’s Algorithm
Lollipop graph with $2^{16}$ nodes

Vitaly Osipov, Johannes Singler, Peter Sanders  
Improving Kruskal's Algorithm
Lollipop graph with $2^{22}$ nodes

Graph showing the comparison of various algorithms (Kruskal, qKruskal, Kruskal8, filterKruskal+, filterKruskal, filterKruskal8, qJP, pJP) in terms of time (in nanoseconds) vs. the number of edges divided by the number of nodes.
Image Segmentation Application

Improving Kruskal’s Algorithm