Minimum Spanning Tree Optimizations

Randy Cornell
Advised by: Martin Burtscher
Department of Computer Science

CS 4299: Undergraduate Research

Efficient Computing Laboratory

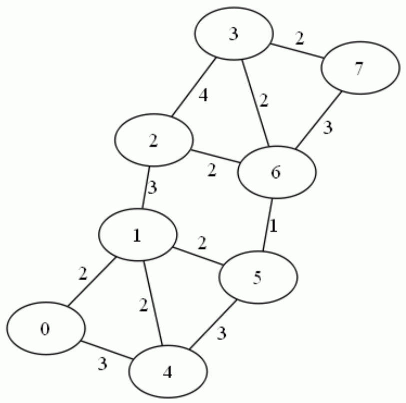
Presentation Structure

- Introduction
- Other people’s approaches
  - Classical approaches
  - Modern approaches
- Our approaches
- Result analysis
- Conclusion

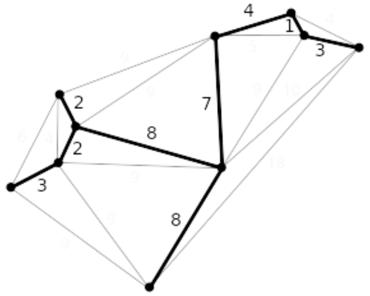
Introduction

- The Minimum Spanning Tree (MST) problem is a graph problem.
- The input graph has the following properties.
  - It must be a single connected component.
  - It must be an undirected graph.
  - It must have edge weights.
- Here are the properties of the output graph.
  - It must be a tree (no cycles).
  - It must contain a connection to all nodes from the initial graph.
  - It must be the lightest possible graph that follows the previous conditions.

Undirected Weighted Graph

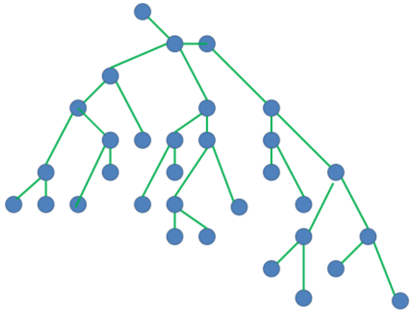


Minimum



Spanning

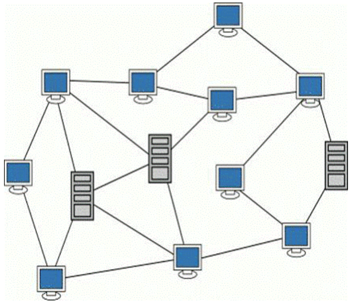
Tree



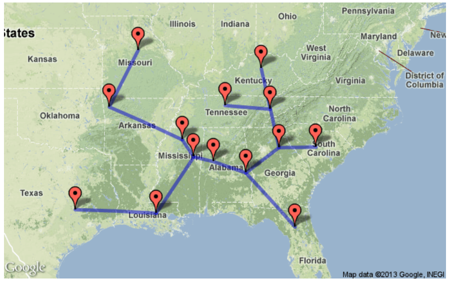
Introduction

- This problem has many important applications
  - Network design
    - Communication networks
    - Road networks
    - Utility networks
  - Approximation for NP-hard problems
    - Traveling Salesman Problem
    - Steiner Trees
  - Computer Science
    - Computer vision
    - Handwriting recognition
    - Image registration
    - Cluster analysis
  - Other various applications
    - Biology
    - Medical
    - Engineering
    - Finance

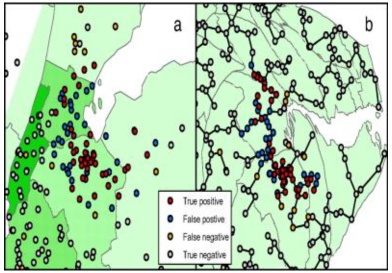
Computer Networks



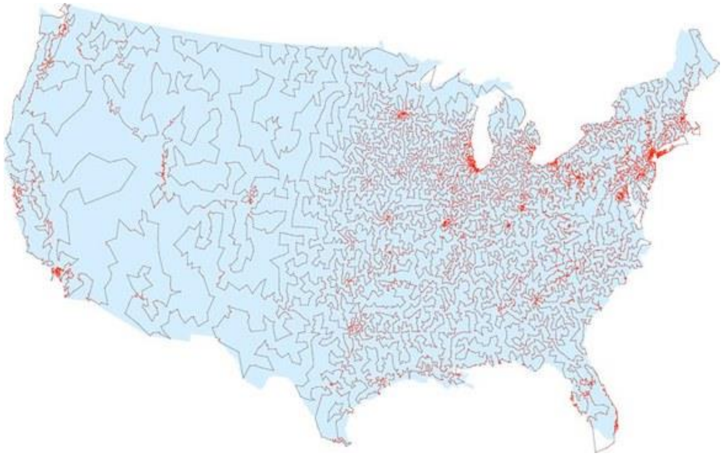
Road Networks



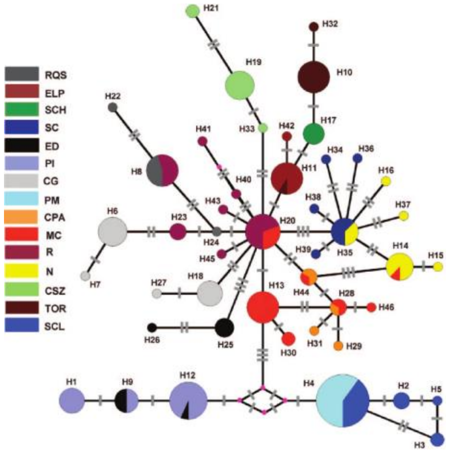
Medical Maps



TSP Approximation

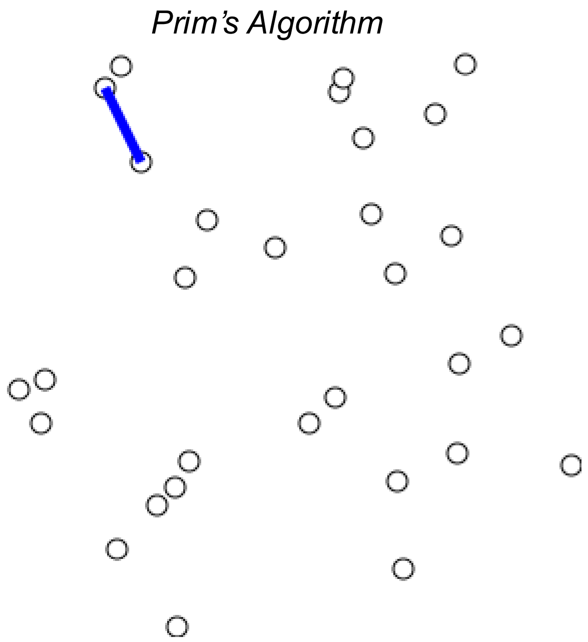


Biology



Classical Approaches: Prim's (Dijkstra's)

- For Prim's Algorithm you do the following
  1. Add any node to the MST
  2. Find the lightest edge leading outside of the MST
  3. Add the destination node to the MST
  4. Repeat until you have an MST

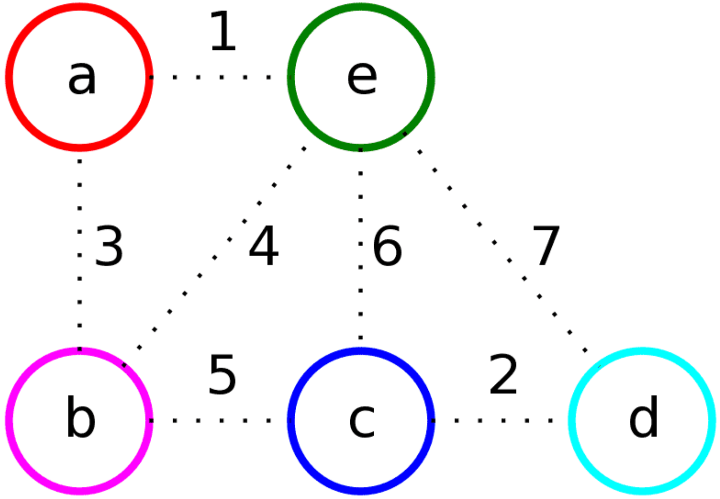


Classical Approaches: Kruskal's

- For Kruskal's Algorithm you do the following
  1. Sort edge list
  2. Check if the lightest edge creates a cycle
  3. If it does not create a cycle add it
  4. Repeat until you have an MST

Kruskal's Algorithm

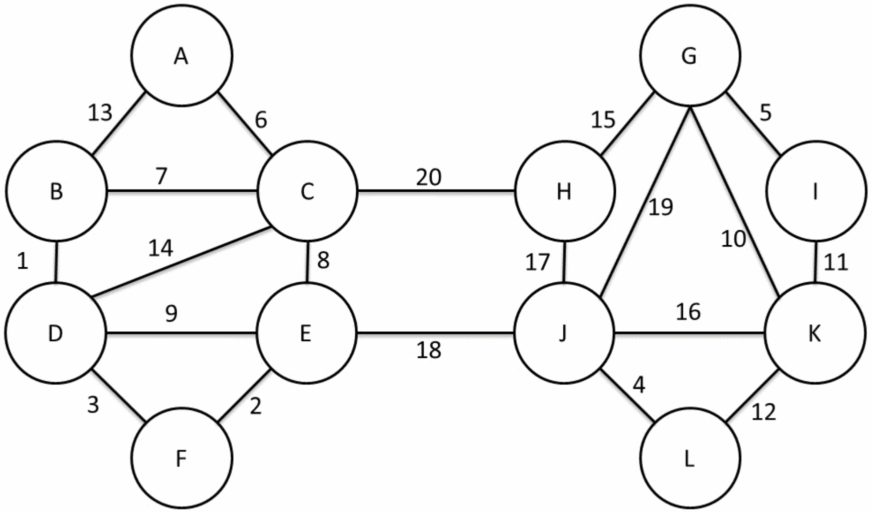
Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6



Classical Approaches: Boruvka's

- For Boruvka's Algorithm you do the following
  1. First make every node a tree
  2. Find the lightest edge leading outside of the tree
  3. Merge trees together that are connected via these lightest edges
  4. Repeat until you have an MST

Boruvka's Algorithm



Modern Approaches

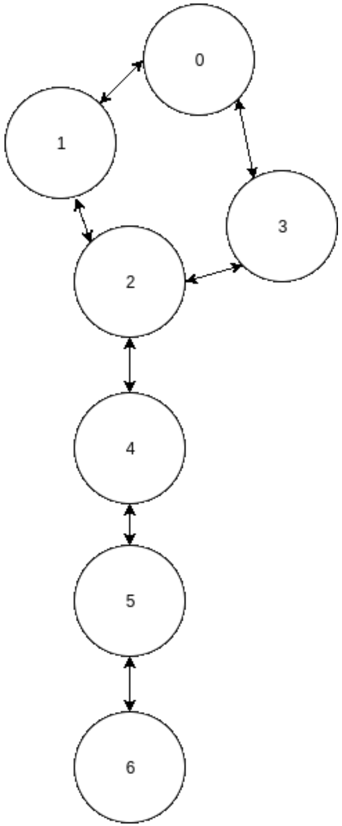
- Prim
  - Priority queues
    - Binary
    - Binomial
    - Fibonacci
  - Fredman and Tarjan
- Kruskal
  - Bucket sorting
  - Filter Kruskal
- Boruvka
  - Yao
  - MST verification

- Combinations
- Other various
  - Non-Greedy
  - Vertex removal

Our Approaches: Linked List Removal

- Most graphs have a large number of nodes that only connect to 1 edge
  - These edges have to be in the MST
    - Remember the spanning property
- If this edge leads to a node with 2 edges
  - These edges have to also be in the MST
  - This is due to the fact that it takes 2 edges that could form a cycle
    - We know that an edge leading to single node cannot form a cycle
    - So you can add the other edge that this node is connected to
- This leads to improvement
  - You do not have to track as many trees
  - Fewer nodes to sort (Kruskal)

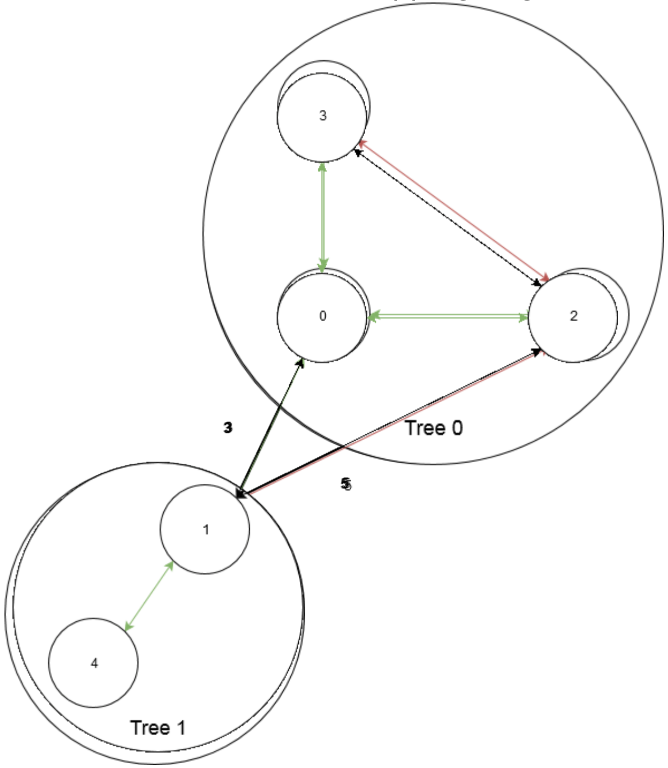
Linked List Removal



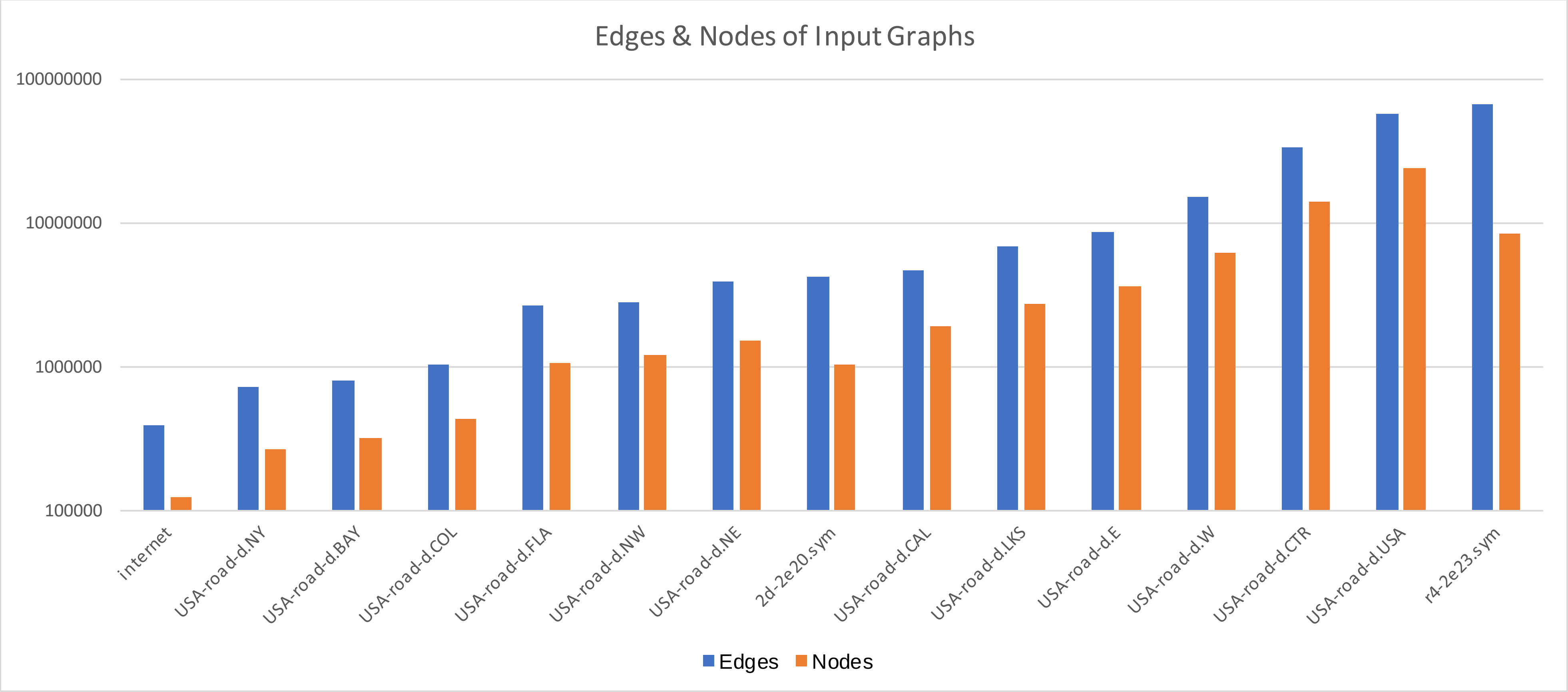
Our Approaches: Skipping Edges

- With Boruvka you can determine that certain edges cannot be in the MST
  - The edge creates a cycle
  - The non-lightest edge connecting the same trees
- You can also determine the same thing about nodes as well
  - If there are no viable edges

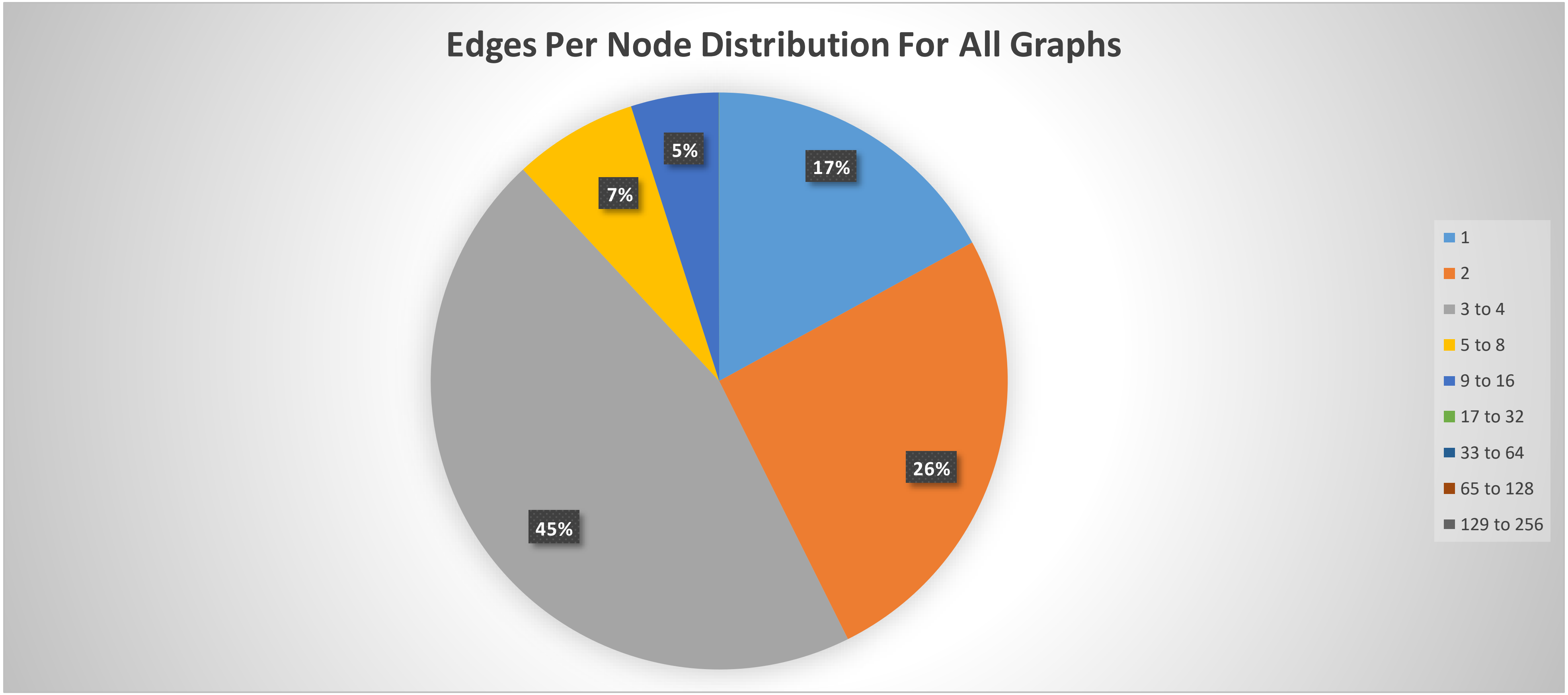
Different Cases with Skipping Edges



Input Graphs: Edge & Nodes



Input Graphs: Degree Distribution



ECL Codes Comparison

- Serial Version
  - Boruvka
    - Baseline
    - Edge Skipping
    - Linked List Removal
  - Kruskal
    - Baseline
    - Bucket Sorting
    - Linked List Removal
  - Prim
    - Edge based
    - Node based
- OpenMP
  - Boruvka
  - Kruskal
  - Prim

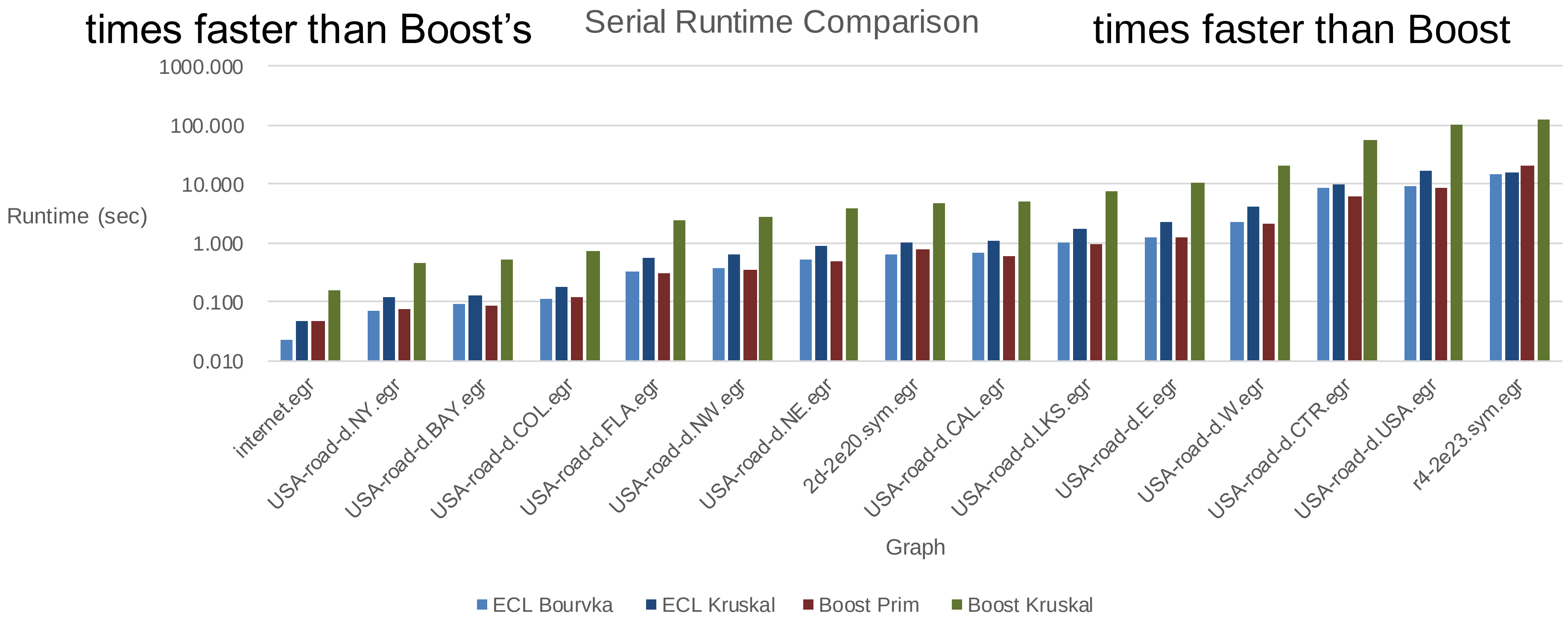
Result Analysis: Comparison Code

- Serial
  - Boost
    - Industry standard
    - Well documented
    - General graph solving C++ library
    - Kruskal and Prim version

Result Analysis: Serial Runtime Comparison

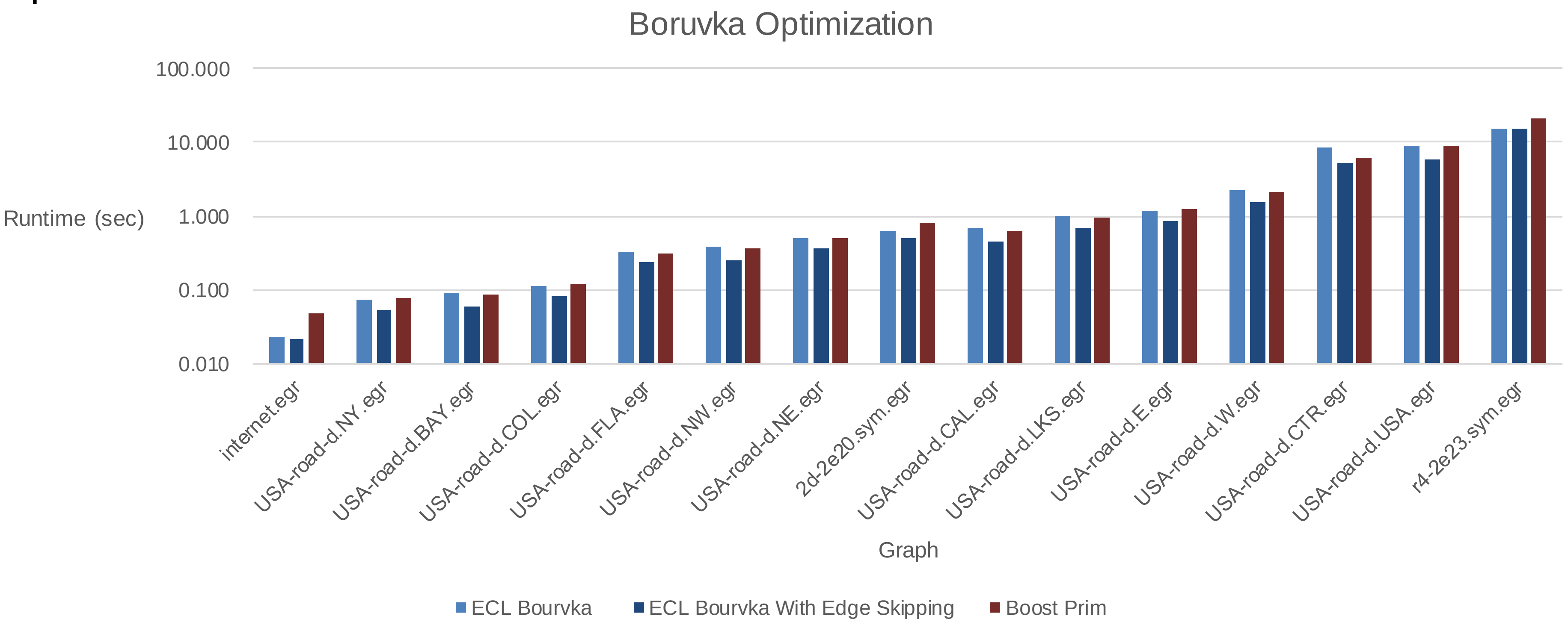
On average our Kruskal baseline code was 4.5 times faster than Boost's

On average our Boruvka baseline code was 1.07 times faster than Boost



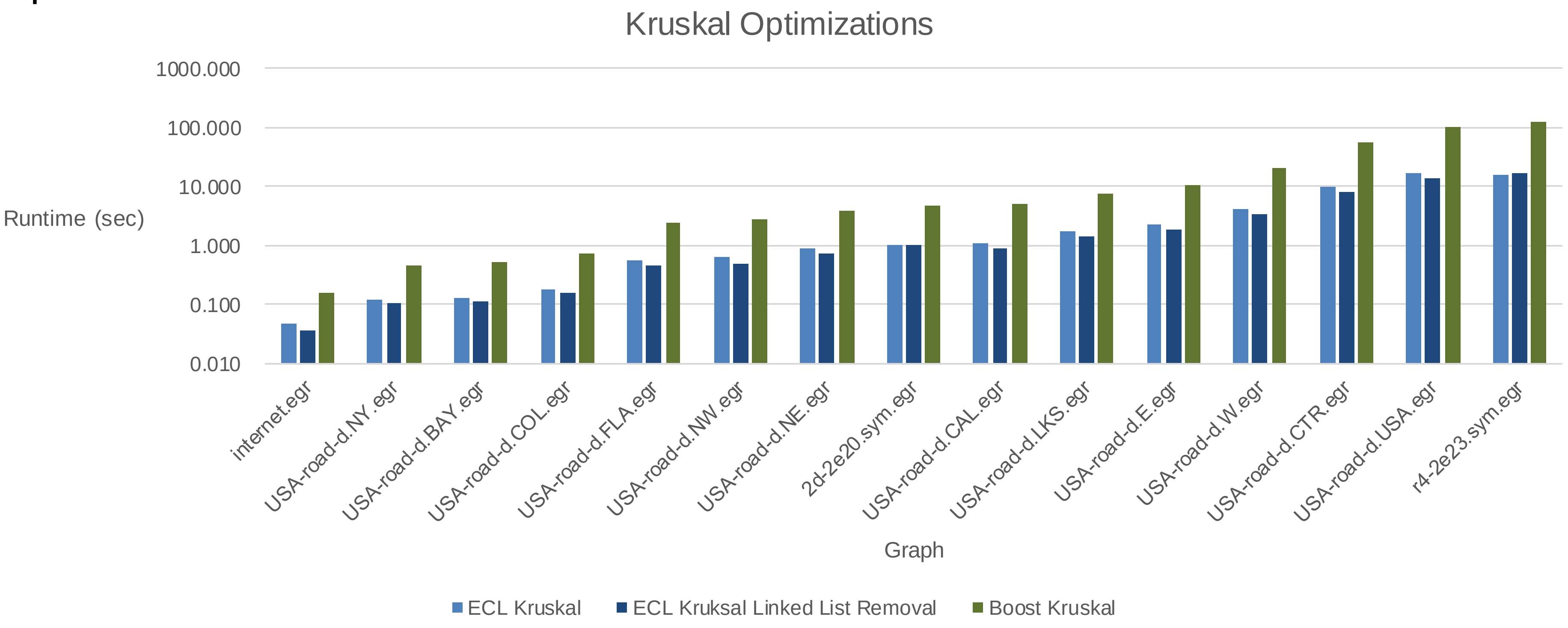
Result Analysis: Serial Boruvka Comparison

This led to a 1.5 times improvement over Boost



Result Analysis: Serial Kruskal Comparison

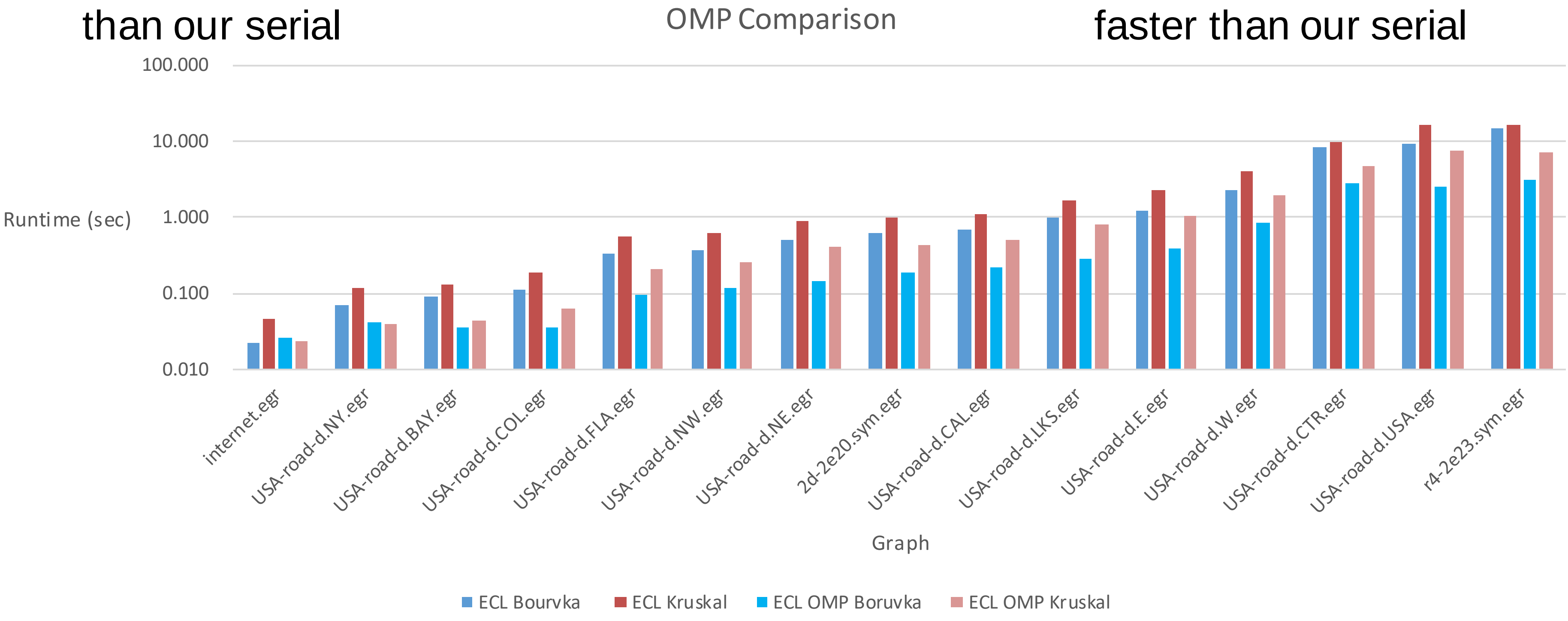
This led to a 5.5 times improvement over Boost



Result Analysis: OpenMP Runtime Comparison

On average our Kruskal OMP was 2.3 times faster than our serial

On average our Boruvka OMP code was 3 times faster than our serial



Conclusion

- Our code out performs Boost
  - Our Boruvka is 1.5 times faster
  - Our Kruskal is 5.5 times faster
- Be careful when relying on big O
- Don’t forget that smaller optimizations can still lead to improvements
  - Edge Skipping
  - Linked List Removal
- Sometimes it is better to break your problem into individual parts if you can
  - For example: The connectivity and the weights can be broken into separate problems

Questions?

References

- https://stackoverflow.com/questions/8025342/undirected-graph-conversion-to-tree
- http://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/
- http://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/
- https://www.researchgate.net/figure/236240895_fig2_Fig-2-Minimum-spanning-tree-of-46-mitochondrial-DNA-haplotypes-of-Ctenomys-talarum-is
- http://cs-pages.blogspot.com/2011/10/what-is-real-world-scenario-where.html
- http://www.pnas.org/content/104/22/9404/F2.expansion.html
- https://www.wired.com/2013/01/traveling-salesman-problem/
- https://commons.wikimedia.org/wiki/File%3APrimAlgDemo.gif
- https://commons.wikimedia.org/wiki/File%3AKruskalDemo.gif
- https://commons.wikimedia.org/wiki/File%3ABoruvka's_algorithm_(Sollin's_algorithm)_Anim.gif
- https://upload.wikimedia.org/wikipedia/commons/c/cd/Boost.png
- http://www.boost.org/doc/libs/1_59_0/libs/graph/doc/graph_theory_review.html
- https://www.cs.unm.edu/~moret/mst.ps
- http://www.cs.princeton.edu/~wayne/cs423/lectures/fibonacci-4up.pdf
- https://www.cs.princeton.edu/courses/archive/fall03/cs528/handouts/fibonacci%20heaps.pdf
- http://ac.els-cdn.com/0167642383900114/1-s2.0-0167642383900114-main.pdf?_tid=f200c8f4-425b-11e7-95d3-00000aab0f27&acdnat=1495835012_236fc24bc29774a56e01a8ba89a593ea
- http://algo2.iti.kit.edu/documents/algo1-2014/alenex09filterkruskal.pdf
- https://www.cc.gatech.edu/~bader/papers/MST-JPDC.pdf
- http://leeds-faculty.colorado.edu/glover/221%20-%20An_in-depth_empirical_investigation_of_non-greedy_approaches_to_the_MST.pdf
- http://ac.els-cdn.com/S1877050916316325/1-s2.0-S1877050916316325-main.pdf?_tid=c3b1986e-464a-11e7-9cef-00000aacb35e&acdnat=1496267438_7e5a27fcc1cb7355a9d540720f4c7c73
- https://archive.org/details/oeloglogvalgorit691yaoa
- http://algo2.iti.kit.edu/documents/algo1-2014/alenex09filterkruskal.pdf
- https://pdfs.semanticscholar.org/2574/a87cb4e36d0bf8994ce2bf02243a5eefb278.pdf
- http://ac.els-cdn.com/0167642383900114/1-s2.0-0167642383900114-main.pdf?_tid=f200c8f4-425b-11e7-95d3-00000aab0f27&acdnat=1495835012_236fc24bc29774a56e01a8ba89a593ea