By: Aaron Csetter, Dmytro Dobrynin, Nathan Davis, and Ian Peña

Comparing Sequential and Parallel Algorithm Performance for solving the Maximum Clique Problem

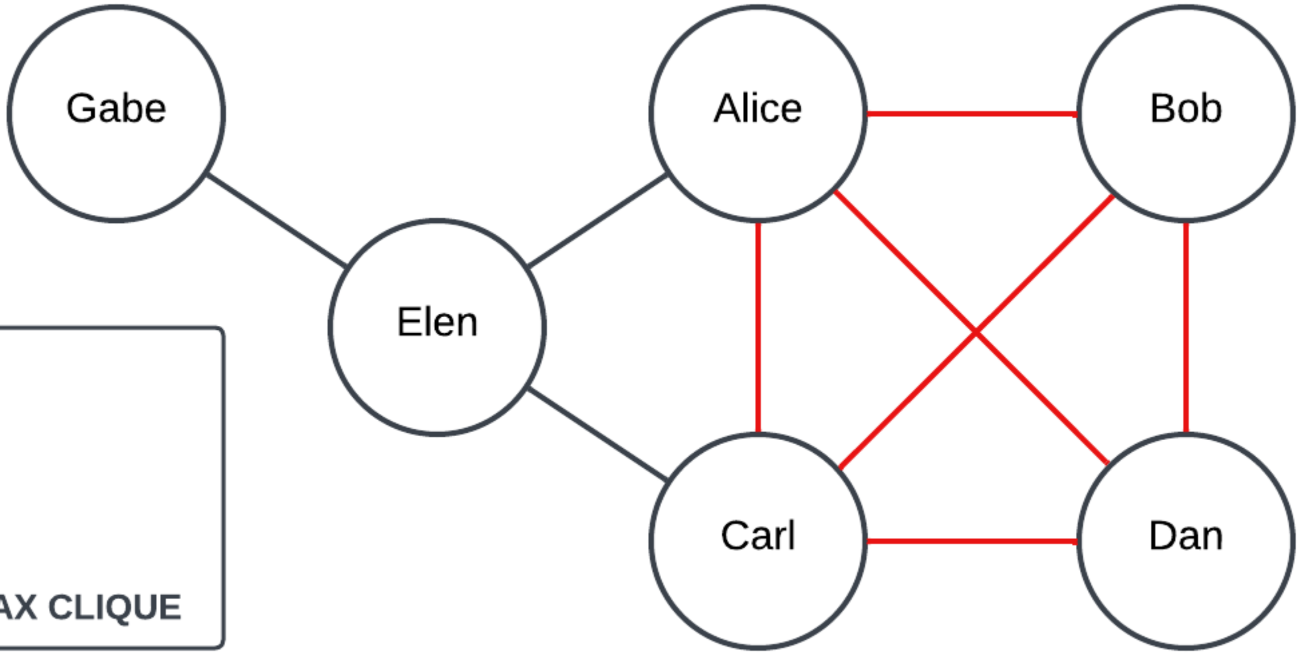
The Maximum Clique Problem

- Fundamental problem in computer science and graph theory.
- Involves finding the largest complete subgraph (i.e., a subgraph in which every vertex is connected to every other vertex) in a given graph.
- Has numerous applications in various fields:
  - including social networks (identify groups with similar behaviors)
  - Bioinformatics
  - Computer science
  - Data mining (identify patterns in large datasets)
  - Graph theory (operation research, transportation, etc.)

NP-Completeness of the MCP

- The Maximum Clique is an NP-Complete problem...
  - There is no known efficient algorithm that can solve it for all instances of the problem in polynomial time.
  - Therefore, finding the maximum clique in a graph is a computationally difficult problem.
- The time complexity:
  - $O(3^{n/3}) = O(1.4422^{n})$
- Using a recursive branch-and-bound strategy with backtracking:
  - $O(2^{n/3}) = O(1.2599^{n})$

MCP Visualization



Cliques (size 3+):
Alice, Bob, Carl
Alice, Bob, Dan
Alice, Carl, Dan
Alice, Carl, Elen
Bob, Carl, Dan
Alice, Bob, Carl, Dan <-- MAX CLIQUE

Imagine the graph as a group of people where each node is a person, and each edge is a friendship between two people.

A clique, then, is a group of people **who are all friends with eachother**.

The **max clique** would be the largest friend group.

Branch and Bound Algorithm

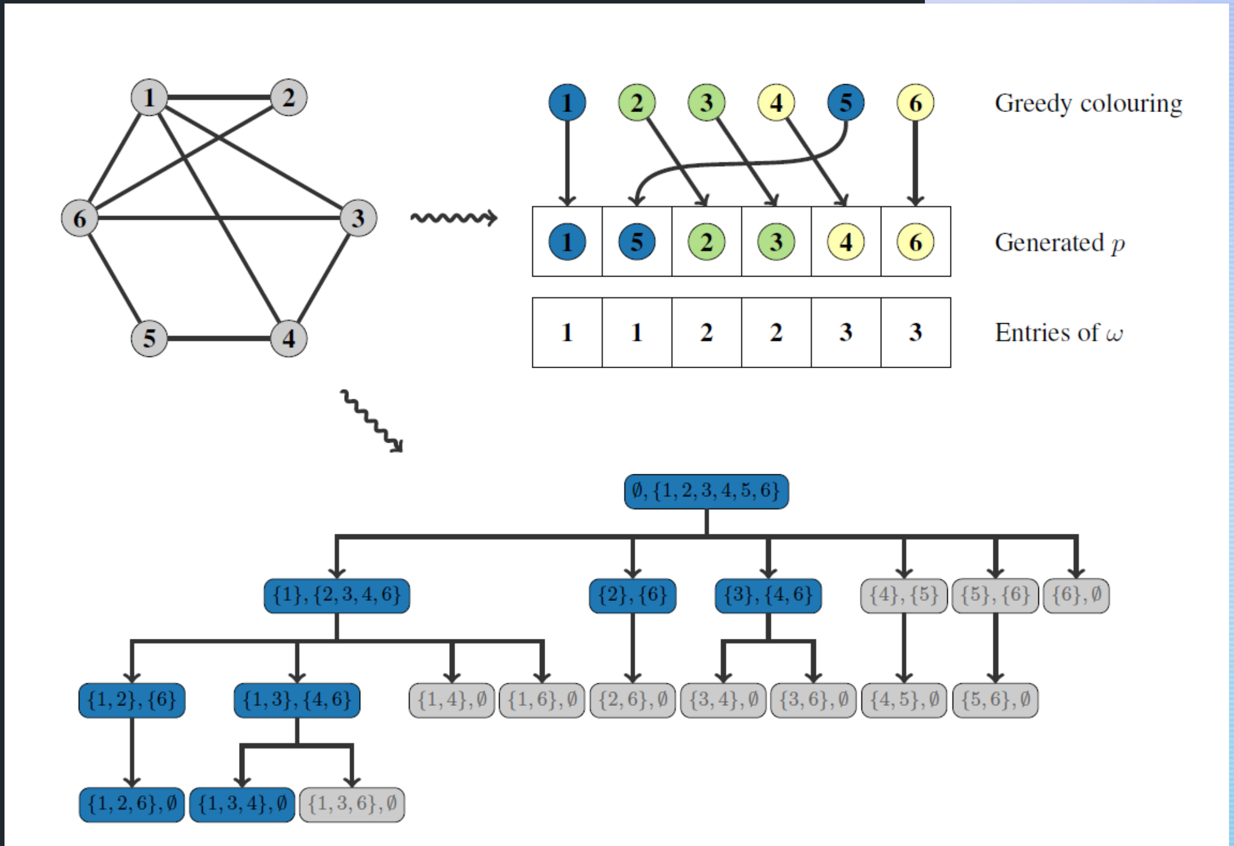
- Common solution for solving the maximum clique problem.
  - Set *P* is all the nodes in the graph, *C* is a candidate clique, and C_{max} is the max clique.
  - For every node in *P*:
    - If len(P) + len(C) > len(C_{max}): ← bound
      - Add the node to the candidate clique set *C*.
      - Put all the nodes adjacent to the current node in *P* to new set *P'* (p-prime).
      - Make *P' the new P* in a recursive call. ← branch
      - Do this until *P is empty and save C as C_{max}* if len(C) > len (C_{max}).

Improvements on Branch and Bound

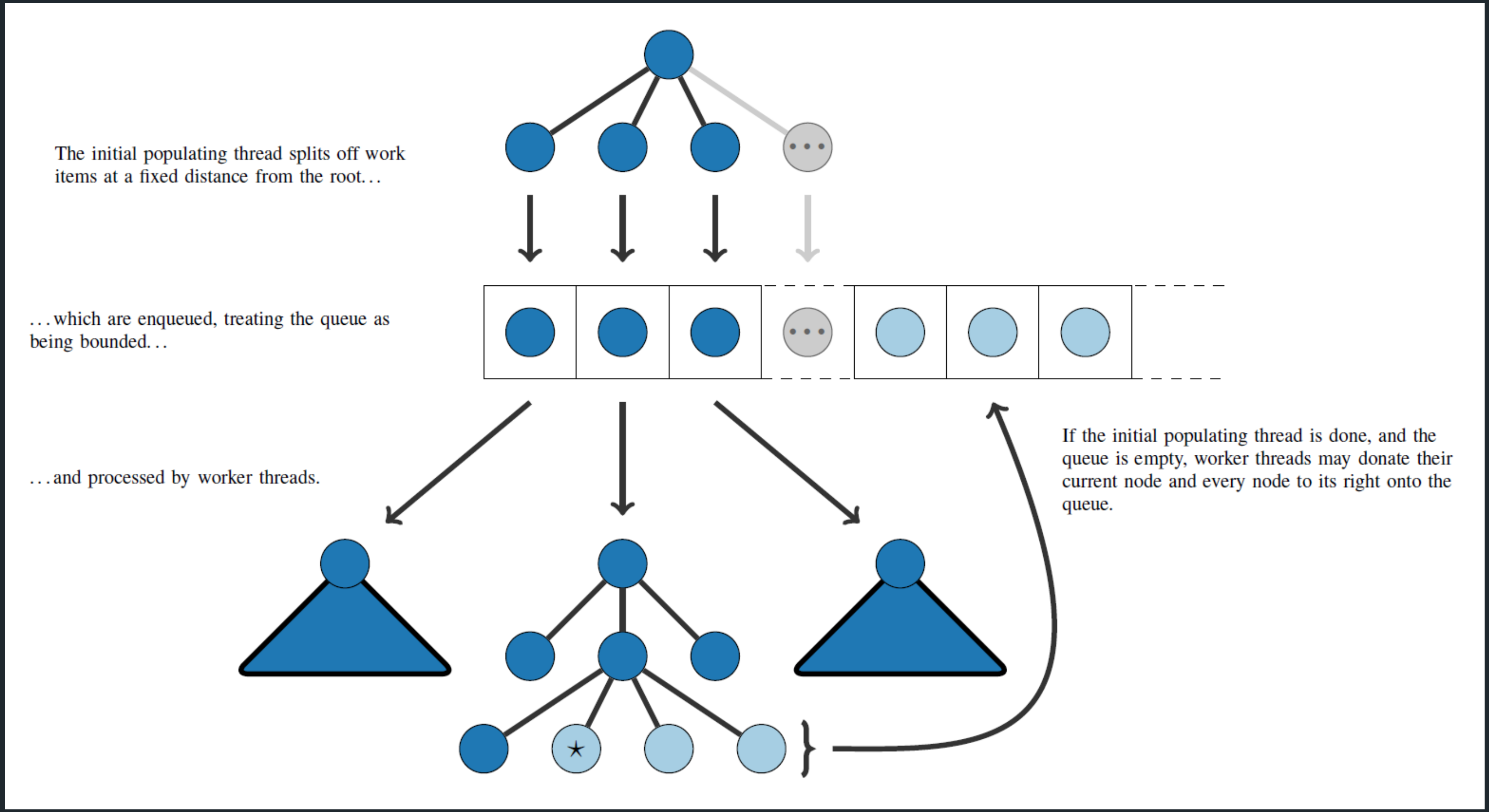
- We can prune more of the search space by using various heuristics and optimization techniques:
  - Vertex ordering, color coding, preprocessing...
- We decided to use the vertex ordering combined with a simple greedy graph coloring algorithm.
  - Helps to reduce the number of candidate cliques worth expanding.
  - This follows research by McCreesh and Proosser, whose ideas we based this research on.
- It is worth noting that graph coloring is also an NP-complete problem, but the greedy algorithm can be computed in polynomial time.

Graph Coloring Algorithm

- A set of nodes is the input.
- All nodes not adjacent to each other are colored the same, starting with the node of highest degree and going down from there.
- The output is the enumerated colors and order of the nodes that the algorithm will use to branch on.

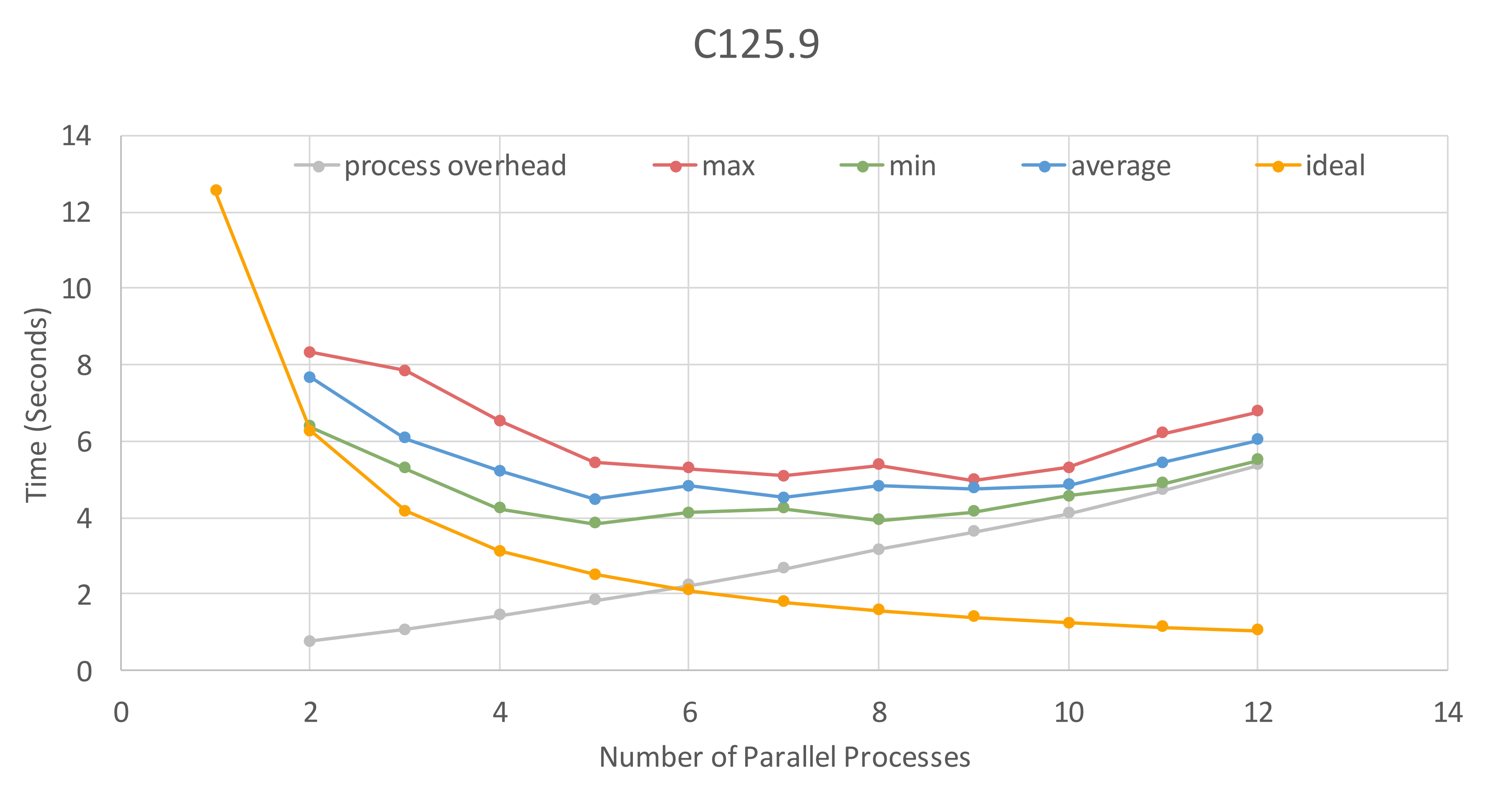


Parallel Adaptation using 'Work Donation'



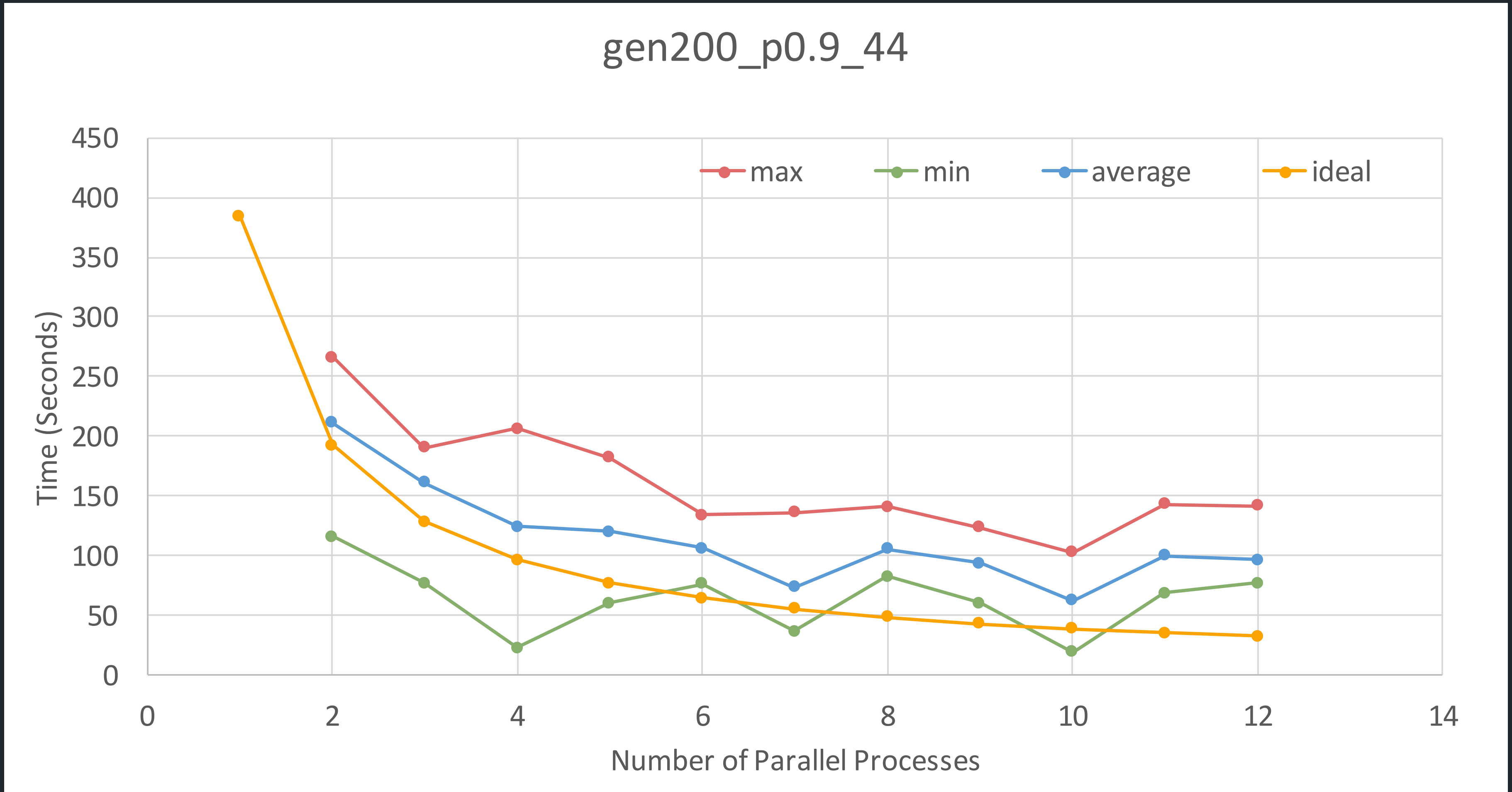
Results

- 125 vertices
- 6,963 edges
- 0.898 density
- Sequential: 12.53 seconds



Results

- 200 vertices
- 17910 edges
- 0.900 density
- Sequential: 384.37 seconds



Thank you!