Branch and Bound Algorithms in Graph Coloring

Anna Dubovitskaya

CSC 591: Experimental Algorithms

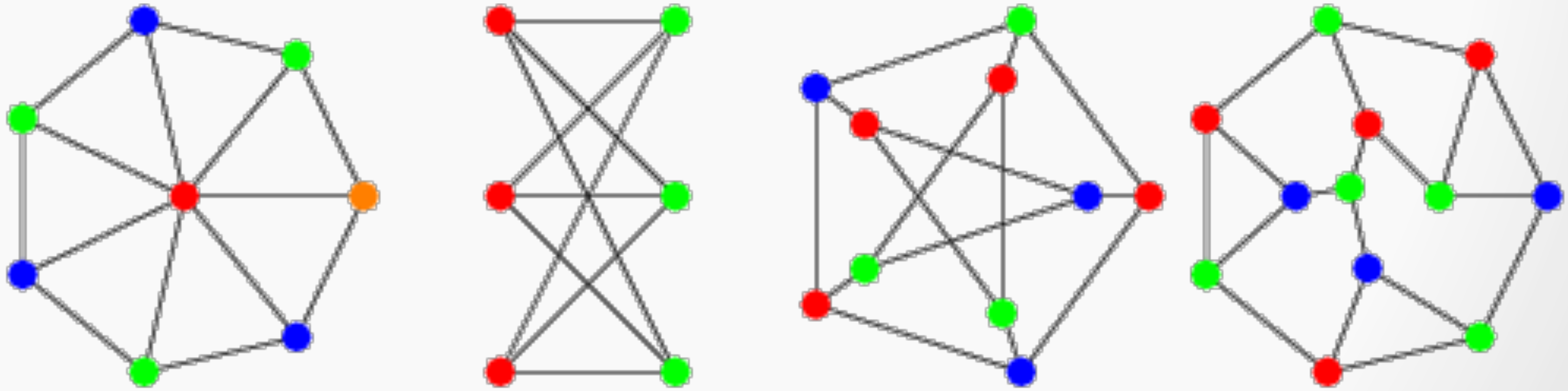
Content

- Graph Coloring
- Branch and Bound
- Programs
- Performance Comparisons
- Conclusions

Graph Coloring

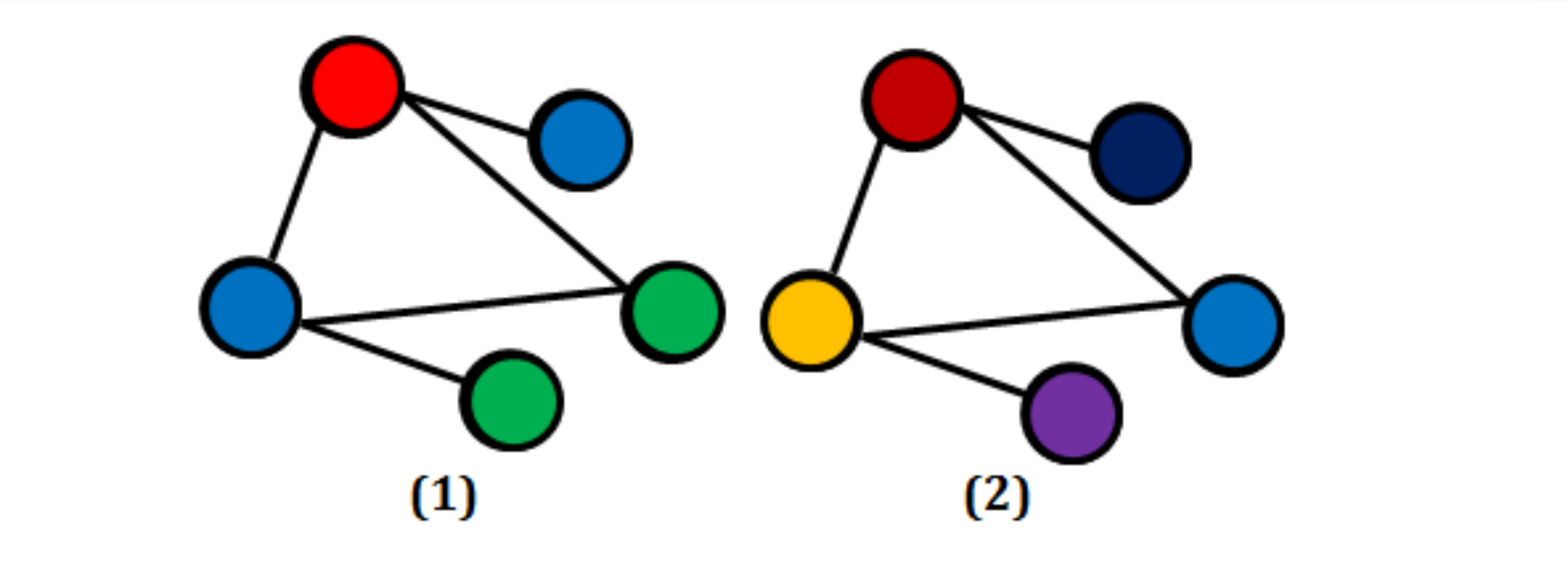
Definitions of Interest:

- **graph coloring**: coloring each vertex of a graph so that no two vertices that share an edge have the same color
- ***k*-coloring:** a coloring using at most *k* colors



Graph Coloring Algorithms

Goal: minimize number of colors used



Branch and Bound Algorithm

Branch:

- recursively generate search spaces

Bound:

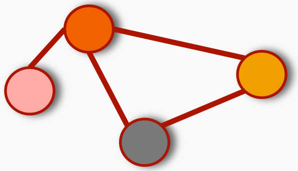
- evaluate these based on some metric

Branch and Bound Definitions

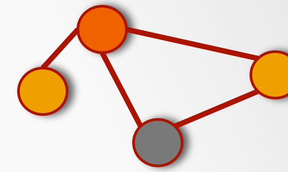
- node:
  - either the initial node, or a recursively generated child node
    - graph to be colored, lower bound, upper bound
- lower bound:
  - smallest k value for a given graph G
- upper bound:
  - largest k value for a given graph
- global upper bound:
  - current, best minimized coloring at a given stage of the branch and bound algorithm.

Branch and Bound Steps

1. Initialize an initial node containing the graph G for which we are trying to determine the minimized k-coloring.
2. Determine the upper and lower bound of the node using some process
3. If the upper bound is smaller than the global upper, set the global upper bound to be equal to the smaller upper bound.
4. If the lower bound is higher than the upper bounds, the node is killed.
5. Recursively generate child nodes by choosing two vertices in graph G that are not connected by an edge, and:
    1. draw an edge between them in a new graph G', forcing the two vertices to be different colors.
    2. collapse them in a new graph G', forcing the two vertices to be the same color.
6. Place the generated nodes onto the priority queue.
7. Pop the first node off of the queue and evaluate using steps 2-7.

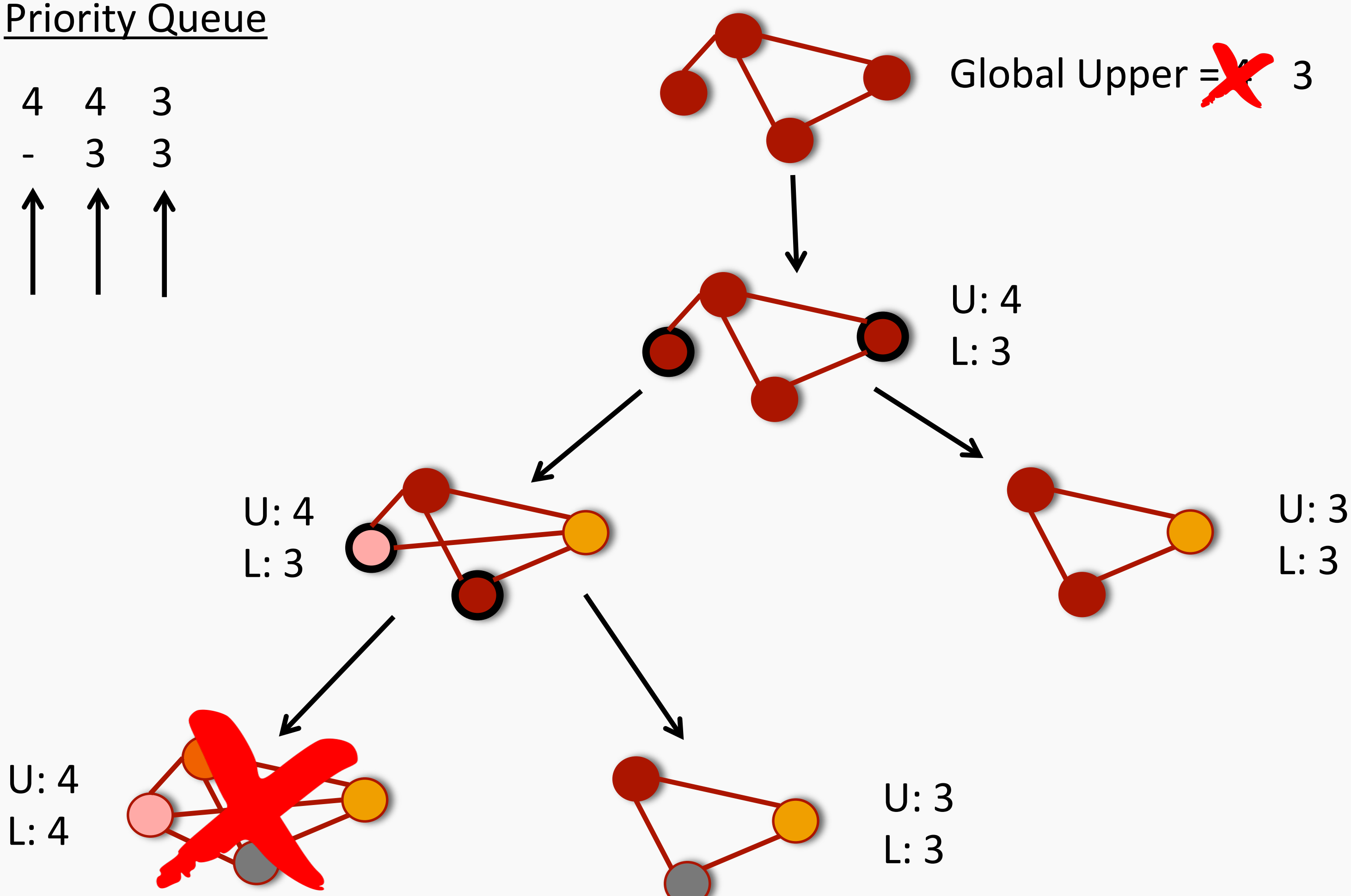


Example



Priority Queue

4	4	3
-	3	3



Experimental Question

What adjustments or decisions can we make to the branch and bound algorithm in order to improve its performance and efficiency in minimizing the k value for some given graph G?

Experimental Details

- Python
- pypy
- networkx module
- DIMACS coloring instances

Experimental Measurements

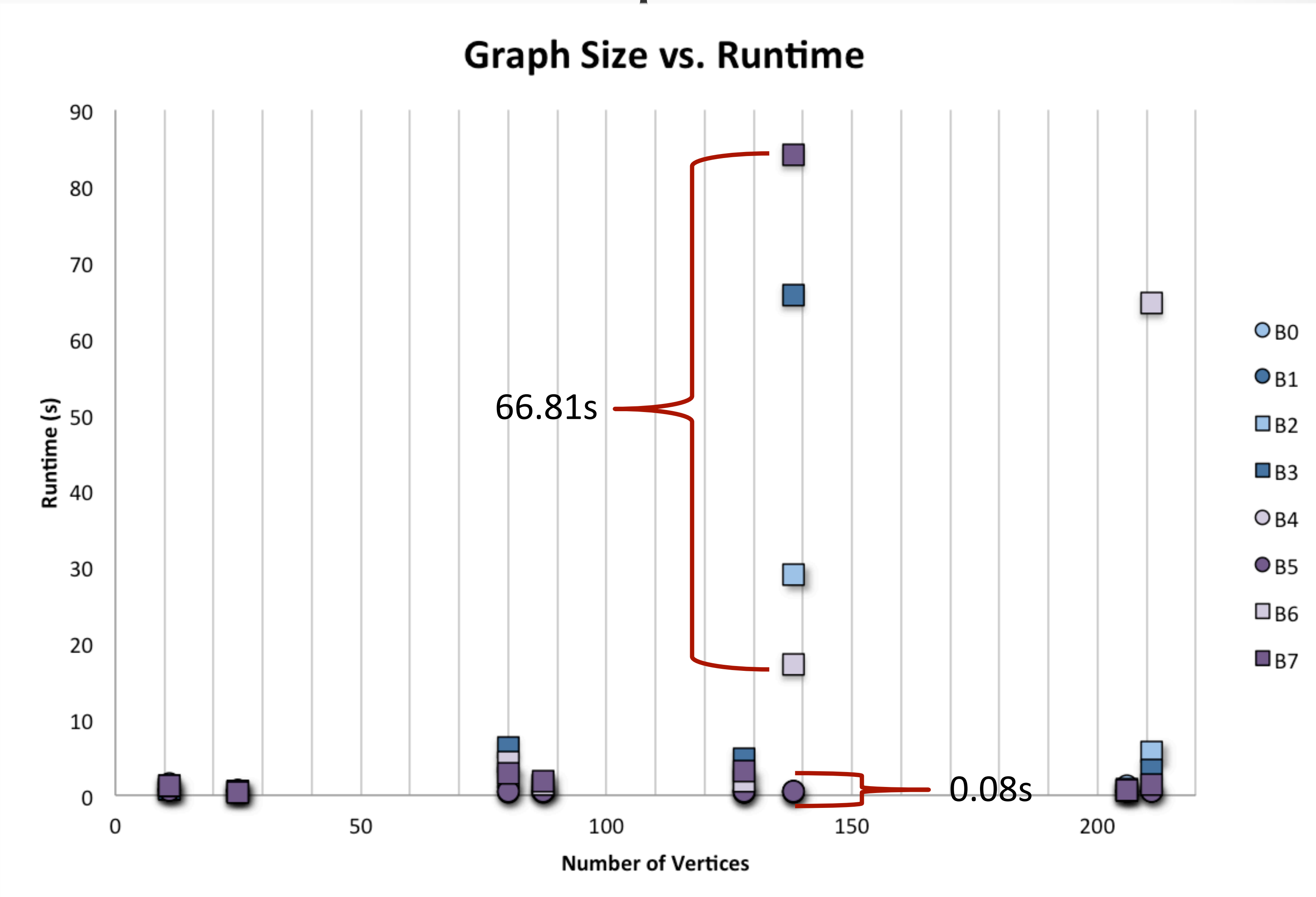
- k value found
- Runtime
- Nodes parsed until solution
  - cutoff = 100,000 nodes
  - after this, solution is not likely to get any better
- Last type of node parsed

Addressing the Question

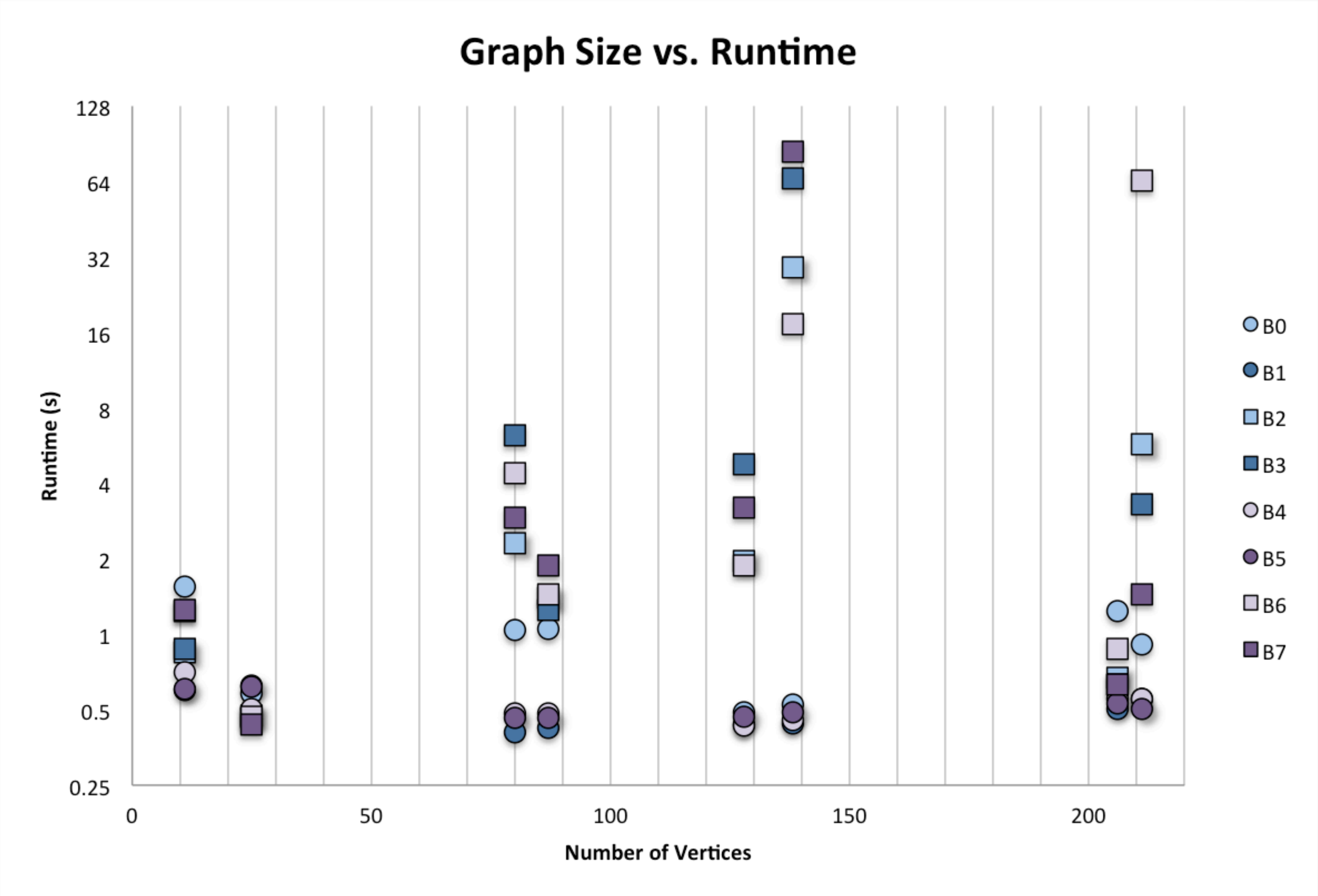
Branch and Bound Algorithms			
Algorithm	Priority Queue Implementation	Coloring Strategy	Edge Picking
BBC0	last created first	largest first	first encountered
BBC1	last created first	largest first	greedy
BBC2	last created first	independent set	first encountered
BBC3	last created first	independent set	greedy
BBC4	lowest lower bound first	largest first	first encountered
BBC5	lowest lower bound first	largest first	greedy
BBC6	lowest lower bound first	independent set	first encountered
BBC7	lowest lower bound first	independent set	greedy

Results: Benchmarks

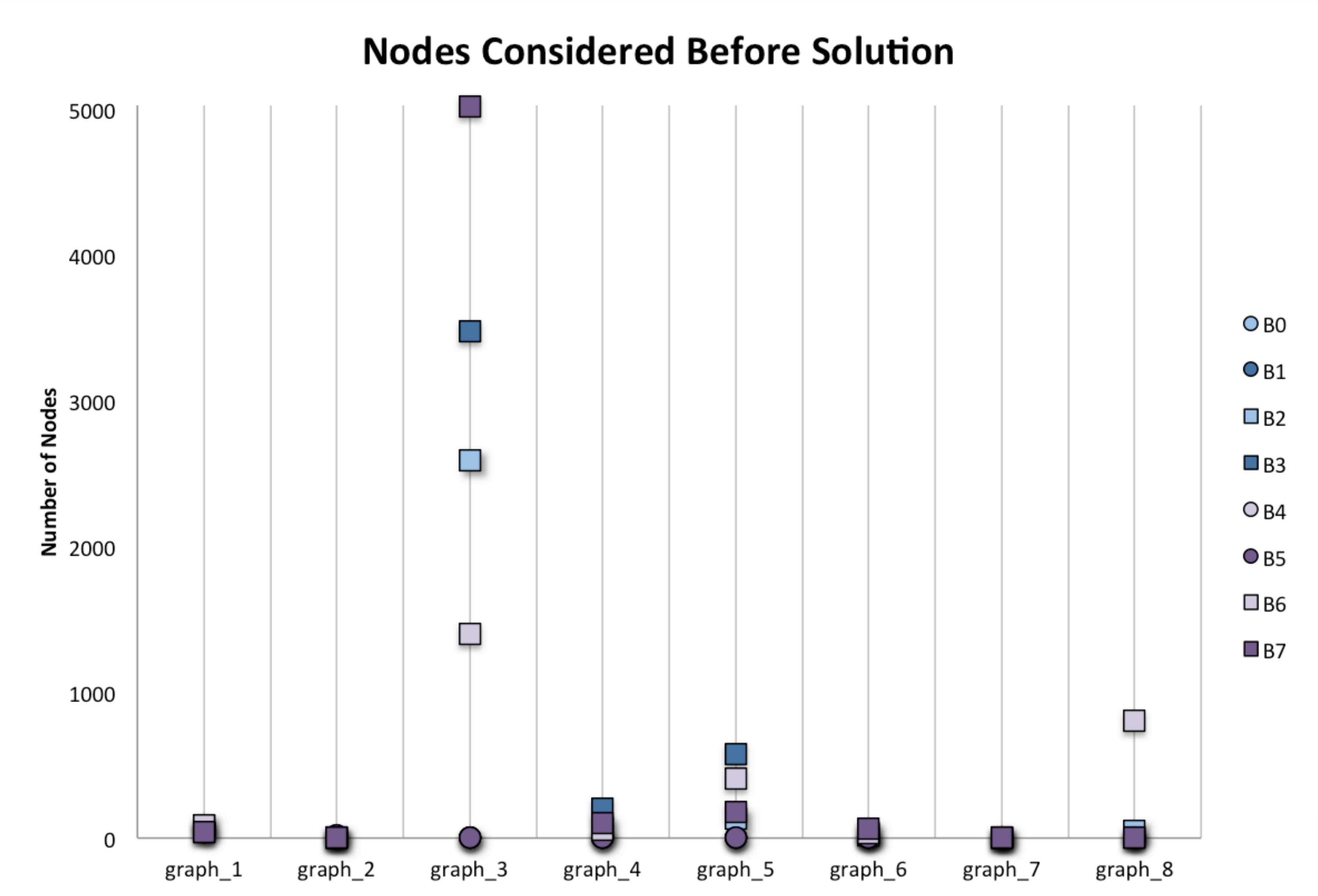
Results: Size of Graph vs Time to Solution



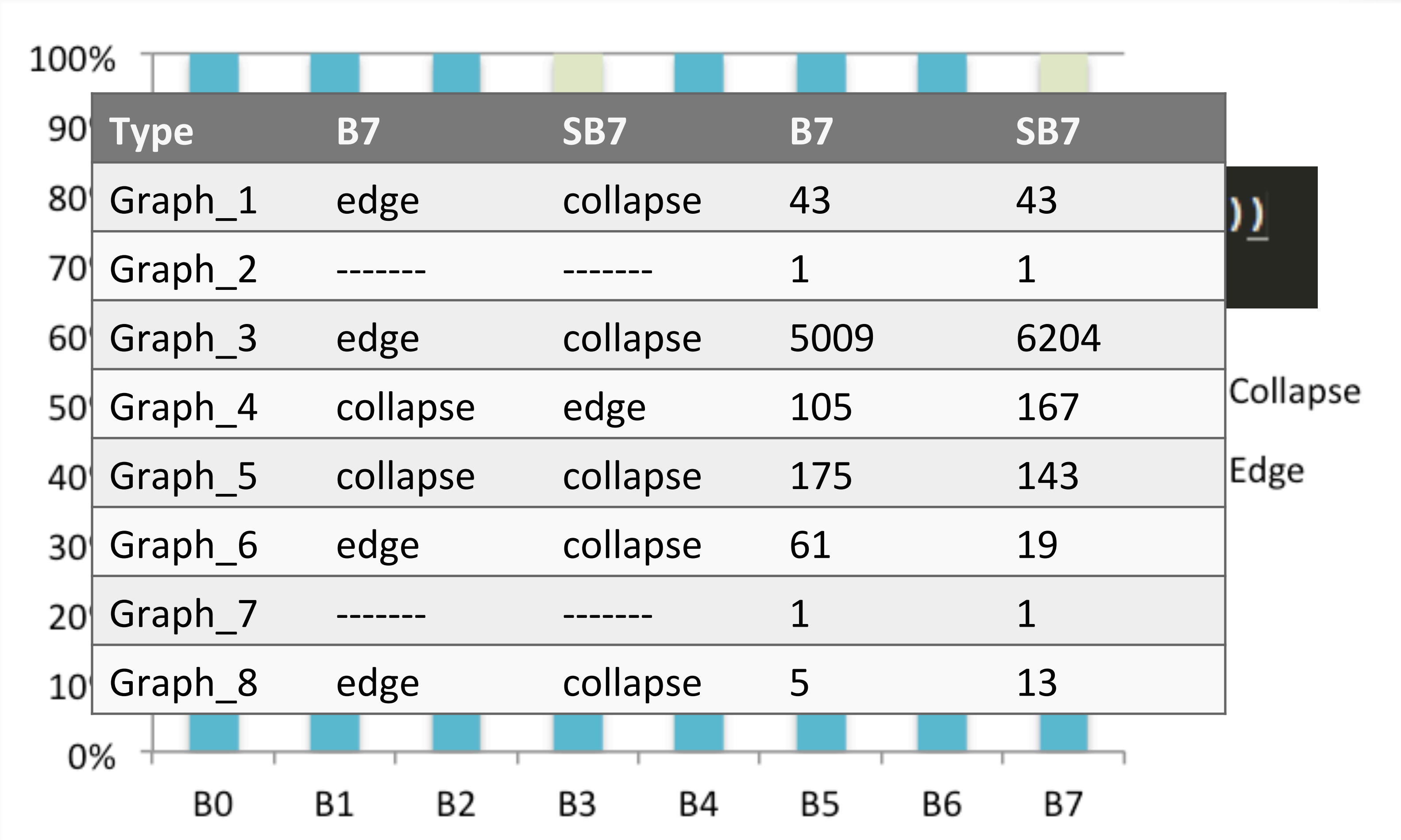
Results: Size of Graph vs Time to Solution



Results: Nodes Considered Before Solution



Results: Edge or Collapse?



Type	B7	SB7	B7	SB7
Graph_1	edge	collapse	43	43
Graph_2	-------	-------	1	1
Graph_3	edge	collapse	5009	6204
Graph_4	collapse	edge	105	167
Graph_5	collapse	collapse	175	143
Graph_6	edge	collapse	61	19
Graph_7	-------	-------	1	1
Graph_8	edge	collapse	5	13

Difficulties

- Difficult to find good bounds
  - non-trivial problems
- Small, hard-to-update lower bounds result in slow growth
  - growing too fast is also bad, k will not be minimized
- Greedy coloring and other adjustments greatly increase runtime for smaller graphs, but may pay off in the long run
- Python issues