Progress Log

Albert Kutsyy

27th January 2021

26th October 2020

- Set up git
- Started progress log

2nd November 2020

- Slower progress since the announcement of national lockdown
- AlGhamdi et al. and Matta et al. simply measure the total runtime, I will simply measure time per node, total time, memory usage, and number of graph reads. I will use `System.nanoTime`.
- For the time being, will represent graphs by an array of nodes and set of edges
- Began skeleton of framework
- Attempted to get access to high power computing on Windows, failed at CL VPN setup, will try on Linux
- Downloaded test datasets

3rd November 2020

- Worked on setting up access to CL resources

4th November 2020

- Successfully got access to the high power computing servers rio, nile, and yellow. **MILESTONE**

5th November 2020

- Got access working on Linux boot as well.
- First day of national lockdown.

4th November 2020

- Completed framework. **MILESTONE**

14th November 2020

- Completed `Brandes` for weighted and unweighted graphs. **MILESTONE**
- Tested on unweighted graphs ONLY. (by using known graphs

20th November 2020

- Tested `Brandes` against `JGraphT`. **MILESTONE**
- Found that I had bugs

21st November 2020

- Fixed bugs, tested on a medium (1k nodes) multigraph.
- Added multigraph support

30th November 2020

- Completed introduction
- Attempting to do Fibonacci Heap for Brandes weighted queue, but JGraphT has removed features
- Switching back to JGraphT 1.3.0 because it is the last documented version (we love documentation. . .)
- Cannot switch to older JGraphT because the way Graphs are imported has changed
- Instead using FibHeap from here:

  `https://keithschwarz.com/interesting/code/?dir=fibonacci-heap`
- Have bugs :(

1st December 2020

- Fixed bug! It was a 0-weight cycle in the input
- Got introduction approved!
- Will do background where I go into depth about previous work and the algorithms
- Copying from my own proposal is fine according to Tim
- Optimized Brandes, it is now faster than JGraphT

4th December 2020 - 11th December 2020

- Attempted to use a simpler representation of graphs - SetGraph, ran tests to compare to original (running on large graph), couldn't get it faster than ~10% slower, even with `trove` primitive hashmaps.

11th December 2020

- Brandes++ implementation can be found at `https://cs-people.bu.edu/edori/code.html#Betweenness_centrality`
- Holy shit that implementation is god awful wtf it's unreadable, I'm not gonna have trouble showing that my code is as fast as it, I would have trouble being slower than 100x faster
- My `Brandes` implementation runs 55500% faster...

14th December 2020

- Using Metis since it's consistently the fastest in Brandes++
- Using `http://glaros.dtc.umn.edu/gkhome/node/78`
- Using HEM since it's consistently the best tested by the above paper, and it is easy to revert to RM
- Merging weighted and unweighted graphs since the difference in memory usage is small and not worth the development time (and casting nonsense)

15th December 2020

- Using KGGGP since it was found to consistently perform best for partition
- `https://ieeexplore.ieee.org/document/7445341`

- Doing 5-way partition initially
- Using Local Greedy Approach since that's what appears to be used by METIS
- Using GGGP as described in "A FAST AND HIGH QUALITY MULTILEVEL SCHEME FORPARTITIONING IRREGULAR GRAPHS" 1998

16th December 2020

- Trying with ArrayGraph and `https://ocw.mit.edu/courses/electrical-engineering-and-c` `6-895-theory-of-parallel-systems-sma-5509-fall-2003/projects/kasheff.` `pdf`

19th December 2020

- The burnout got really intense, had to take a day off and it was very helpful and made me more productive today than I have been in a while.
- Got coarsening and partitioning done
- Using BKL(*,1) for refinement, as it was found to be the best and fastest
- Using boundary of 2% as in the paper.
- Need to implement a new data structure for finding neighbors and KL refinement
- This is sub-algorithm #7...

22nd December 2020

- Finished meletis!
- Tested on small graphs
- Tested that refinement reduces edge-cut
- Tested that for all P_i, union P_i is all nodes in G and for any P_i, P_j union is empty
- Tested partition sizes are roughly equal (not identical due to coarsening phase)
- Manually reviewed again and again that my code matches the specification
- URLs used:
  `http://glaros.dtc.umn.edu/gkhome/metis/metis/publications`
  "Multilevelk-way Partitioning Schemefor Irregular Graphs1" George Karypis and Vipin Kuma
  "MULTILEVEL GRAPH PARTITIONING SCHEMES" George Karypis and Vipin Kumar
  `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/`

6-895-theory-of-parallel-systems-sma-5509-fall-2003/projects/kasheff.pdf " FAST AND HIGH QUALITY MULTILEVEL SCHEME FORPARTITIONING IRREGULAR GRAPHS" GEORGE KARYPIS†ANDVIPIN KUMAR 1998
https://dl.acm.org/doi/10.5555/800263.809204
http://users.ece.northwestern.edu/~haizhou/357/lec2.pdf
https://github.com/sahibjotsingh/KL-Algorithm

- Replaced old Brandes implementation with a primitive based approach (using ArrayGraph) - testing improvements (previous wiki-vote was 7.2 seconds, now 3.95)

2nd January 2021

- Brandes++ assumes an undirected graph, need to add a djikstra's on each subgraph from each non-frontier to each frontier.
- Paper incorrectly says looking for s→v→f, code and formula indicate s→f→v
- Problem:
  - In the version of the paper published on researchgate on June 2014 here: https://www.researchgate.net/publication/263201045_A_Divide-and-Conquer_Algorithm_for_Betweenness_Centrality on page 8 under the Build_SK algorithm description it says "However, Brandes++ is not equipped to take shortest paths starting from s into consideration if $s \in V_i/F_i$ for some supernode G_i" and describes that they guarantee this is not the case by making s its own cluster.
  - The archivx version submitted 16 June 2014 and last edited 4 June 2015 available here: https://arxiv.org/abs/1406.4173 says on page 5 under the same section:

    "Note that since we need to know the shortest paths for every target node $s \in S$, we treat the nodes in S specially. More specifically, given the input partition P, we remove all targets from their respective parts and add them as singletons. Thus, we use the partition <Math>. Assuming that the number of target nodes is relatively small compared to the total number of nodes in the network, this does not have a significant effect on the running time of our algorithm." with no particular emphasis, immediately going to discussing how some operations could be parallelized.
  - The proceedings of the 2015 SIAM international conference on data mining says the same https://epubs.siam.org/doi/10.1137/1.9781611974010.49

8th January 2021

- So, turns out the paper is for the shortest paths between a subset of nodes, I've been struggling with that for a few days.
- Will implement and compare to an approach using a modified Brandes

- Their implementation takes $O(\sum |S||F_i| \sum |V_i/F_i|)$ which is slower than the paper describes
- Switched from combining ints to longs to just using a map of maps after finding that the latter is faster (intCombiningTest)

13th January 2021

- Testing on p2p unweighted undirected:
  Binary heap: 66.85s Fibonacci heap: 65.54s
- Nope I'm just stupid, that was with unweighted so didn't use the heaps

15th January 2021

- Implemented RankPairingHeap and it works!

27th January 2021

- Done with Geisberger!
- Also implemented GeisbergerLinear, GeisbergerEnumerative, and Brandes(2008)