1 Introduction

This article will be looking at different methods at approximating graph metrics, mainly by graph reductions, and analyzing the results the approximations give. There are two approximation methods that this article will go through. The first is using approximation algorithms; these are algorithms that do not calculate the exact value of a metric, but a similar value that can be used to estimate the exact value. The second method is to compute the graph metric on a *reduced* graph. In this article, a reduced graph of a graph G is a graph with the same number of vertices as G, but with less edges. The goal of these approximation methods is to decrease the run time of graph algorithms, while still getting good results in the graph metrics.

2 Approximation Algorithms

This section will cover the approximation algorithms used in the experiments to compute graph metrics.

2.1 Approximate Diameter

The diameter of a graph is the maximal distance between a pair of vertices. That is, the shortest path between any two vertices is at least as small as the graph's diameter. The algorithm that will be analyzed in this article for approximating graph diameter will be based on iterations of "2-BFS".

Each iteration performs breadth-first search twice. The first BFS is performed starting at a random vertex v, to find a vertex u that is of maximal distance from v. The distance from u to v, d_v is recorded. Then, the second BFS is performed starting at u. The maximal distance from u to another vertex, d_u is recorded. (this is not necessarily the distance from u to v). Then an iteration of "2-BFS" will return $max(d_u, d_v)$. The approximate diameter algorithm runs "2-BFS" 1000 times, and returns the maximum value of the 1000 "2-BFS" iterations.

These values all give a lower bound to the actual diameter of a graph. However, an upper bound can be given by multiplying the values given by 2. Then a nice upper bound can be given by multiplying the lowest value by 2, and lower bound given by the highest result.

2.2 Graph Reductions

There are four main graph reductions used in the experiments below. All the reductions keep the same number of vertices, but try to lower the number of edges in the graph. Some of the graph metrics we want to measure characterize vertices in the graph, so there should be no loss of vertices.

The first reduction algorithm iterates through each vertex, and keeps the edges to the highest k_1 degree neighbours.

The second reduction iterates through each vertex v, and prioritizes its high degree neighbours, but also attempts to avoid any triangles formed by the neighbours. On each vertex iteration, each neighbour is initially given a priority 1. The algorithm will choose to keep the highest degree neighbour with priority 1. Whenever an edge vu is kept, the algorithm goes through each pair of remaining neighbours and checks if the pair forms a triangle with u. If so, then the priority of the pair of neighbours increase by 1. The algorithm continues by adding the next highest degree neighbour that still has priority 1. If there are no more neighbours with priority 1, then neighbours of priority 2 will be added, and so on until k_2 edges are kept. This reduction is detailed in Algorithm 1, and is named Triangle Top K, or T Top K.

The third reduction combines the edges of several spanning trees to form a reduced graph. In the experiments below, the spanning trees are rooted at the highest k_3 degree vertices of G.

The last reduction keeps 1 out of k_4 neighbours for each vertex, if the degree of the vertex is above the median value of all vertex degrees.

Algorithm 1 Graph Reduction Avoiding Triangles (T Top k)

Input: Original Graph G, Empty Graph H, integer k

for all vertices v of G **do**
  Add v to H
  for all neighbours u of v **do**
    Priority[u] $\leftarrow 1$
  end for
  EdgesAdded $\leftarrow 0$
  Priority $\leftarrow 1$
  while EdgesAdded != k **do**
    Find highest degree neighbour u where Priority[u] = Priority
    Add edge uv to H
    EdgesAdded = EdgesAdded +1
    for all neighbour pairs (w, x) of v **do**
      if (u, w, x) is a triangle in G **then**
        Priority[u] = Priority[u] +1
        Priority[w] = Priority[w] +1
        Priority[x] = Priority[x] +1
      end if
    end for
    Priority $\leftarrow$ Priority +1
  end while
end for

Output: Reduced Graph H

3 Analysis

Experiments were performed on graphs from Stanford's Large Network Collection. Running time was not measured, but the change in the number edges was measured, which is a good indication on how much faster some algorithms will perform on the reduced versions. For each graph, the first two reductions were running four times, with k_1, $k_2 = 2, 4, 8, 16$. This third reduction was run twice, with $k_3 = 3, 5$. And the last reduction was run once, with $k_4 = 4$. For completeness, the experiments were done with undirected and directed graphs; however, the purpose for these experiments are for undirected graphs.

For each graph and reduced graph, in addition to diameter and number of connected components, one thousand reachable random pairs of vertices were picked from each original graph. Then, for the reduced graphs, the distance of each pair was measured again, the difference in distance was recorded, if the pair was still reachable. The Page Rank was computed for all vertices in each graph and reduced graph. Since there are a lot of numbers to process, the top 0.15% (or top 15 if ¡ 1000 vertices) page rank vertices were examined in each reduced graph, and the number of those vertices in the top 0.15% and top 1% page rank in the original were recorded. The results will be summarized below, and a full Excel spreadsheet is available with all results.

In the analysis ahead, there will be tables for each category. The category is listed on the top right of each table. Bolded graph names indicate that the graph metric was well approximated for all reductions. Otherwise, some values of the table of the graph will be bolded, indicating the graph metric was well approximated for the given reduction and graph. A "good" approximation is defined below under each category.

The graphs used for analysis were all provided using Stanford's SNAP datasets, found at `http://snap.stanford.edu/data`. The datasets used for undirected graphs are labeled ego-Facebook (fb), ca-CondMat (cm), ca-AstroPH(ap), email-Enron (enron), com-Youtube (yt), as-733 (auto). The datasets used for directed graph are labeled soc-Epinions (dir ep), wiki-Vote (dir wiki).

3.1 Analysis by Graph Metric

Edges - This is an important metric since this is will decrease computing time on a number of algorithms. Of course, this is the goal of these approximations, to obtain close results while lowering the time to compute them. The T Top K reductions have a lower amount of edges than each of its counterpart Top K reductions. For the four iterations for the Facebook graph, the reductions have 9%, 17%, 32%, and 53% of the original edges, going from $k_1 = 2, 4, 8, 16$. The number of edges in the 1 of 4 reduction was typically inbetween the top 4 and the top 8 reduction. The tree reductions contained very few edges, with 3 roots having typically less than the top 2 reduction, and 5 roots with similar number of edges as the top 2 reduction. As you will see, the spanning tree reduction has great performance in the other metrics relative to the number edges it has.

Diameter - Across all undirected reduced graphs, with the exception of youtube's reduced graphs, the approximation algorithm was always within 15% of the diameter of the original graph. The approximation typically had more error when only two or four edges were kept for each vertex in the top k reductions. All in all, the approximate diameter measured in the reduced graphs were very close to the original graph's diameter in all reduced graph iterations for undirected graphs.

For directed graphs, the results are not very good. The approximate diameter fluctuates heavily. As an example, the wikivote graph from Stanford's SNAP measured an approximate diameter of 35 on the reduced graph that kept the top 8 neighbours, and an approximate diameter of 11 on the reduced graph that kept the top 4 neighbours.

A. Diameter	fb	cm	ap	**enron**	yt	**auto**	dir ep	dir wiki
Original	8	15	14	13	24	9	13	8
Top 2	**8**	**16**	**14**	13	16	10	9	**7**
Top 4	**8**	**15**	**13**	12	15	9	**12**	11
Top 8	**8**	**14**	**12**	13	15	9	61	35
Top 16	**8**	**13**	**12**	12	16	9	31	15
T Top 2	10	**15**	**13**	14	17	10	**11**	12
T Top 4	10	**15**	**13**	12	16	1	**13**	27
T Top 8	**8**	**15**	**13**	13	15	10	45	15
T Top 16	**8**	**14**	**13**	12	15	10	19	10
Tree 3	**8**	**15**	**13**	14	16	10	28	16
Tree 5	**8**	**15**	**14**	13	15	10	22	29
Keep 1/4	**9**	20	17	12	16	10	**13**	12

A "good" approximation is defined to be within 15% of the original graph's diameter. (Listed in the Original row)

Page Rank - Page rank results varied depending on the original graph. For some graphs, around half of the reduced versions' top 0.15% page rank vertices were in the top 0.15% of the original. However, most graphs' had 90% of all reduced versions' 0.15% in the original top 0.15%. There is only slight improvement as k went up in all cases, until $k_1 = 8 or 16$, as those results are always at least 80% in the original top 0.15%.

In each original graph, the high degree vertices, in terms of ranking, will continue to be the high ranking vertices in the reduced graph as well. Since Page Rank is computed largely based on the degree of a vertex, it is intuitive that the reduced graphs give accurate rankings to the high page rank vertices.

PR % in Top 0.15%	fb	cm	ap	**enron**	**yt**	**auto**	dir ep	dir wiki
Top 2	67%	63%	54%	89%	94%	93%	**88%**	80%
Top 4	60%	68%	53%	95%	97%	100%	**98%**	**87%**
Top 8	47%	**88%**	59%	96%	98%	100%	**100%**	**93%**
Top 16	60%	**99%**	79%	100%	99%	100%	**100%**	**100%**
T Top 2	73%	59%	53%	86%	94%	93%	**88%**	80%
T Top 4	73%	52%	49%	89%	97%	100%	**98%**	**87%**
T Top 8	80%	57%	47%	95%	99%	100%	**100%**	**93%**
T Top 16	73%	**95%**	62%	100%	99%	100%	**100%**	**100%**
Tree 3	73%	65%	35%	86%	94%	93%	4%	0%
Tree 5	73%	67%	40%	86%	94%	93%	4%	0%
Keep 1/4	**93%**	61%	58%	95%	94%	93%	**85%**	**87%**

An approximation is bolded if it is within 15% error.

Reachability - On undirected graphs, the reachability results had great results. On almost every reduced graph, all 1000 pairs were in the same component. The only time this not the case was for some reduced graphs that only kept 2 edges from each vertex.

However, for directed graphs, many pairs were often in different components in the reduced graphs. For the spanning tree reduced graphs, the reduced graph using 3 roots had $< 1\%$ of pairs in the same component, and the reduced graph using 5 roots had $\sim 10\%$ of pairs in the same component. T Top K reduced graphs showed better results than the Top K results, but still $< 10\%$ of pairs were in the same component for $k_1 = 2, 4, 8, k_2 = 2, 4$. With $k_2 = 8$, $\sim 50\%$ of pairs were in the same component. And there were good results for $k_1, k_2 = 16$.

Although there is some randomness to this measurement, I think these results can be useful to characterize real-world graphs. Vertices with large degree (or "hubs") will preserve most of their edges in the these graph reductions. I believe that since real-world graphs typically have several very large hubs, these hubs keep components similar in size in reduced graphs compared to the original. For directed graphs however, although smaller degree vertices keep their edges to large hubs, the edge can only be traversed one way, so that the number of outgoing edges from large degree vertices is not very large, although the number of incoming edges to the large degree vertices will be very high.

Reachability, /1000	**fb**	**cm**	**ap**	**enron**	**yt**	**auto**	dir ep	dir wiki
Top 2	966	999	998	1000	1000	1000	0	5
Top 4	1000	1000	1000	1000	1000	1000	1	18
Top 8	1000	1000	1000	1000	1000	1000	23	162
Top 16	1000	1000	1000	1000	1000	1000	442	750
T Top 2	1000	1000	1000	1000	1000	1000	0	8
T Top 4	1000	1000	1000	1000	1000	1000	8	187
T Top 8	1000	1000	1000	1000	1000	1000	228	653
T Top 16	1000	1000	1000	1000	1000	1000	548	841
Tree 3	1000	1000	1000	1000	1000	1000	3	2
Tree 5	1000	1000	1000	1000	1000	1000	5	81
Keep 1/4	1000	1000	1000	1000	1000	1000	569	861

An approximation is bolded if it is within 10% error.

Distance - The results on changes of distance were quite similar to reach-

ability. For undirected graphs, the change in distance between the vertex pairs were mostly + 0 or 1, with some distances + 2 if only the 2 edges were kept for each vertex. If at least 8 edges were kept for each vertex, a large majority of the pairs had their distances unchanged. The tree reduction gave similar results to the Top 2 and 4 neighbour reduction, and keeping 1 of 4 edges had similar results to the Top 4 neighbour reduction.

For directed graphs, we will only consider when 16 edges were kept for each vertex. The reachable pairs resulted in a difference in distance centered around +2, with mostly +1, 2, or 3, with some +0 or +4,5.

I believe the intuition for this is similar to reachability, with large hubs giving easy access to paths from one vertex to another, so that these short paths are left unchanged in real-world graphs. Of course, once the top 8 or 16 edges are kept, a lot of the edges are kept from the original graph, and it will be that "less important" edges, edges between low degree vertices are left out, which are typically not used in shortest paths.

Δ Dist 0 & 1, /1000	**fb**	cm	ap	**enron**	**yt**	**auto**	dir ep	dir wiki
Top 2	952	**852**	754	928	991	987	0	3
Top 4	999	**995**	**966**	991	999	999	0	15
Top 8	1000	**1000**	**999**	999	1000	1000	5	44
Top 16	1000	**1000**	**999**	1000	1000	1000	20	253
T Top 2	975	817	691	892	969	983	0	6
T Top 4	987	**956**	**887**	966	993	995	5	36
T Top 8	991	**982**	**949**	991	996	996	18	215
T Top 16	995	**997**	**968**	994	997	997	172	610
Tree 3	996	**839**	**767**	896	982	987	1	1
Tree 5	999	**916**	**855**	956	992	995	6	2
Keep 1/4	976	**975**	**996**	992	981	960	358	550

An approximation is bolded if it is withing 15% error.