Distance Estimation for Very Large Networks using MapReduce and Network Structure Indices

Hüseyin Oktay[1]
Dept. of Computer Science
University of Massachusetts
hoktay@cs.umass.edu

A. Soner Balkir[1]
Dept. of Computer Science
University of Chicago
soner@uchicago.edu

Ian Foster
Dept. of Computer Science,
University of Chicago
foster@anl.gov

David D. Jensen
Dept. of Computer Science
University of Massachusetts
jensen@cs.umass.edu

ABSTRACT

Distance calculation is key to many network mining applications such as centrality and clustering. As the size of available networks increases to millions of nodes and edges, distance calculation becomes a bottleneck for such applications. One way to overcome such bottlenecks is to use the *MapReduce* parallel processing framework, though increasing resources linearly does not scale well for many network mining applications. Hence, instead of calculating the exact distance, efficiently and accurately estimating distance may scale better for very large networks. In this paper, we propose a *network structure index*(NSI) on the MapReduce framework by extending the basic *breadth-first search* algorithm to accurately estimate shortest distance using MapReduce. We demonstrate the accuracy of our method for distance estimation on synthetic and real networks. We use distance estimation along with progressive sampling to achieve two specific applications for very large networks: *closeness centrality* and *betweenness centrality*. We first evaluate our distance estimation method on relatively small networks, then we report our observations about the most central nodes of a Twitter network with more than 40 million nodes.

1. INTRODUCTION

Networks with millions of nodes and edges are increasingly observable. Social networks such as Facebook, Twitter, and LinkedIn have millions of users; billions of web sites are linked by hyperlinks; call networks constructed by who-calls-who information with millions of users are now observable; biological networks representing the protein-protein interaction and numerous other networks are now available. Many researchers in different fields are interested in calculating network properties and statistics for such very large networks.

One key common operation in calculating such statistics and properties is shortest path calculation between nodes, and as the size of the network increases to millions of nodes and edges, such calculation becomes intractable. In this work we combine two existing ideas to overcome this challenge: (1) using the MapReduce (MR) distributed programming framework[2], and (2) efficiently and accurately estimating shortest paths between nodes by indexing the network structure[13].

MR is a widely used, distributed programming framework[2], and network structure indices (NSIs)[13] can provide efficient and accurate distance estimation, which is required for many network mining applications[10]. In this paper, we develop an NSI on the MR parallel processing framework using Hadoop to efficiently and accurately estimate shortest paths between nodes in very large networks. We also use distance estimation with NSIs to estimate closeness centrality and betweenness centrality. We show the accuracy of NSIs on relatively small synthetic and real networks for which we can also calculate the exact values. We also report results from a Twitter network with more than 40 million nodes.

[1]These authors contributed equally to this work.

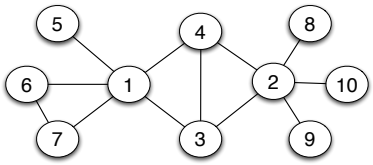


Figure 1: Example toy network

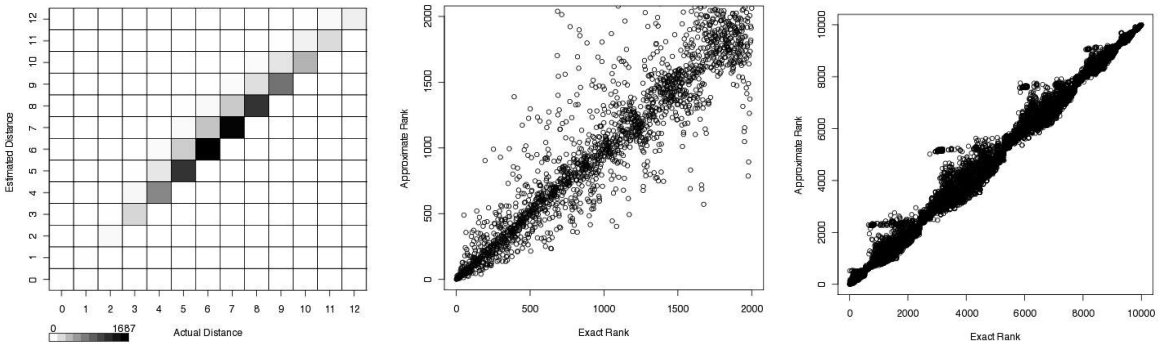
Table 1: Input file for the example network

(a) Initial representation, root is 1

NodeId	Root	Distance	Degree	Neighbors
1	1	0	5	3 5 4 7 6
0	1	inf	1	2
2	1	inf	5	9 0 3 4 8
3	1	inf	3	1 2 4
4	1	inf	3	1 3 2
5	1	inf	1	1
6	1	inf	2	1 7
7	1	inf	2	1 6
8	1	inf	1	2
9	1	inf	1	2

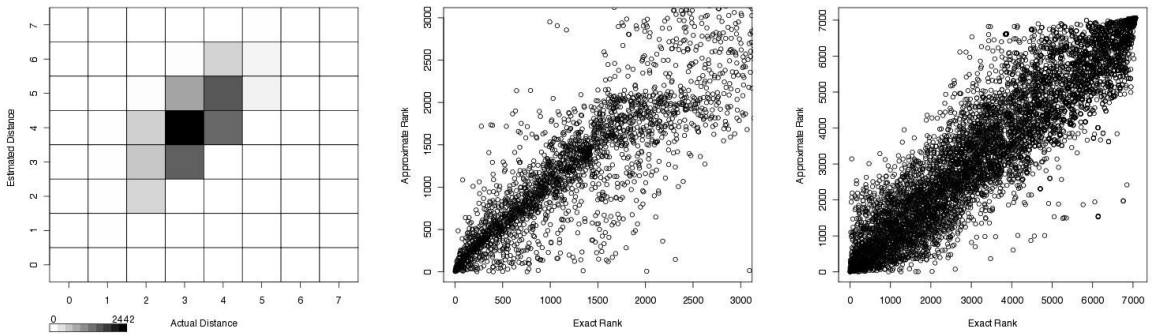
(b) Representation after convergence

NodeId	Root	Distance	Degree	Neighbors
1	1	0	5	3 5 4 7 6
0	1	3	1	2
2	1	2	5	9 0 3 4 8
3	1	1	3	1 2 4
4	1	1	3	1 3 2
5	1	1	1	1
6	1	1	2	1 7
7	1	1	2	1 6
8	1	3	1	2
9	1	3	1	2



(a) Distance estimation (b) Betweenness centrality (c) Closeness centrality

Figure 2: Forest fire network 10,000 nodes, 19,832 edges



(a) Distance estimation (b) Betweenness centrality (c) Closeness centrality

Figure 3: Wikipedia Vote network, 7,066 nodes, 100,736 edges

2. OUR IMPLEMENTATION OF NSIS

Our implementation of NSIs on MR is an extension of the *breadth-first search* (BFS) algorithm. Thus, we first review the BFS implementation on MR and then give details on how to extend it to realize NSIs. We assume for simplicity that networks are undirected and unweighted; however, our methods are easily applicable to weighted and directed networks. We also point out the required modifications to extend these same ideas to weighted and directed networks.

Breadth-First Search in MapReduce

In MR we choose to represent networks as *adjacency lists* because each line corresponds to a node and all local information about a node is summarized in one line. The format of the input file is:

<NodeId> <Root> <Distance> <Degree> <Neighbors>

where *NodeId* is the label of the particular node, *Root* is the root of the breadth-first search tree that is about to be calculated, *Distance* is shortest-path distance from node to root, and *Degree* and *Neighbors* are, respectively, the number of neighbors and a list of those neighboring nodes. Note that in the input file, the distance for each node is set to *"infinity"* for all nodes except the root node, for which distance is set to 0. Table 1a is an example input file for the network in figure 1 where the root is 1. If an NSI has more than one tree, we merge the files for each tree. In BFS for MR, the map function gets a line from the input file, and if the *Distance* is less than infinity, then for each neighbor, the mapper emits key-value pairs, where key is the label of the neighboring node and value is the *Distance* +1. Also, each mapper emits the input line as is, in order not to lose the degree and neighbors information for nodes.

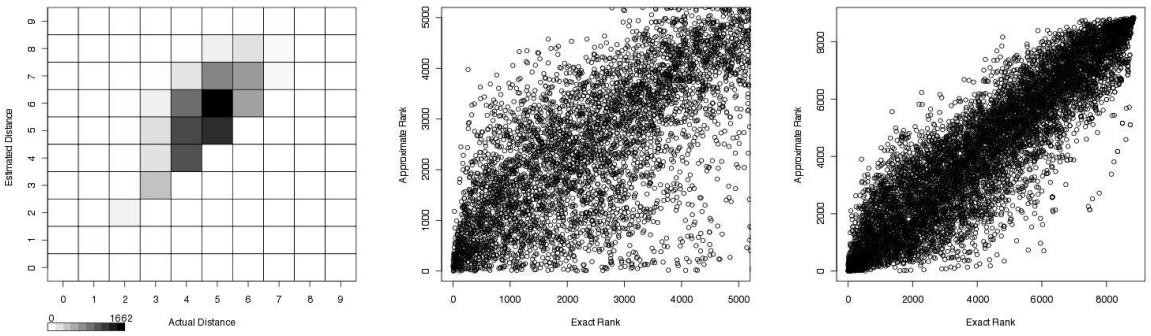
Since key is the NodeId, all values for a specific node are processed by one reducer. For each node, the reducer calculates the minimum distance among the values, and emits *NodeId, Root, Distance, Degree,* and *Neighbors* as output where Distance is the minimum distance. If the node is not reached (i.e., the *Distance* is infinity) then the distance remains infinity. BFS converges when there are no unreached nodes left (i.e., no *Distance* value with infinity). The output file after the BFS for the example graph in 1 is shown in table 1b. In the worst case, BFS needs to run d iterations, where d is the diameter of network.

NSIs in MapReduce

We extend BFS to keep track of the shortest path between nodes and the root of the tree. Basically, our new input file to the algorithm is in the following form:

<NodeId> <Root> <Distance> <Degree> <Neighbors> <Path>

The only modification to the BFS algorithm is to update the path when we emit new distances. We then use the



(a) Distance estimation (b) Betweenness centrality (c) Closeness centrality

Figure 4: Peer-to-Peer Gnutella File Sharing Network, 8,842 nodes, 63,674 edges

path information to find the lowest common ancestors and thus estimate more accurately the distance between nodes. We use the following formula to do distance estimation:

$$\mathrm{D}(n_1, n_2) = \mathrm{D}(n_1, lca) + \mathrm{D}(n_2, lca)$$

where n_1, n_2, and lca are nodes in the network and lca (the lowest common ancestor) is identified through path information we now have in the NSIs. For example, if we want to estimate the distance between (8,9) in our toy network in figure 1 using the BFS tree with root 1, we find that 2 is the lowest common ancestor by using the path information, and we estimate the distance as

$$\mathrm{D}(8,9) = \mathrm{D}(8,2) + \mathrm{D}(9,2) = 1+1 = 2.$$

Extending to Directed and Weighted Networks

Distance estimation can be performed for directed networks by considering only "out degree" and "out neighbors" in the initial representation of the network. For weighted networks, we add another column to the input file that has a list of weights corresponding to the weight of the edge to the corresponding neighbor. More concretely, the input file format for weighted version would be as follows:

<NodeId> <Root> <Distance> <Degree> <Neighbors> <Weights> <Path>

where *Weights* is a list of weights where w_i corresponds to the edge weight between *NodeId* and *ith* neighbor in *Neighbors*. While constructing NSIs, we emit *Distance*+w_i in the map function for each neighbor instead of *Distance*+1 in the unweighted case.

3. APPLICATIONS OF NSIS

Betweenness Centrality

Betweenness centrality[4] is a widely used flow-based metric for identifying central nodes in networks. Formally, betweenness centrality for a node is defined as

$$B(u) = \sum_{v,w \in V} \frac{g_u(v,w)}{g(v,w)}, u \neq v \neq w \qquad (1)$$

where $g(v,w)$ is the number of lowest-cost (geodesic) paths connecting nodes v and w, and $g_u(v,w)$ is the number of these paths that pass through node u. Nodes that occur on more shortest paths between nodes have higher betweenness centrality. For example, in our small network in figure 1, node 2 occurs in most of the shortest paths between the nodes and hence has high betweenness centrality.

Since the original definition of betweenness centrality takes $\mathrm{O}(n^3)$ time, we employ two techniques to make computation tractable. First, as proposed by Maier et al. (2011), instead of calculating all shortest paths between all pairs, we estimate one shortest path between pairs and increase the score by 1 for nodes that constitute the path. Second, instead of calculating paths for all pairs, we use progressive sampling as proposed by Provost et al.[12] and calculate distance for a sample of pairs. We converge when the change in Spearman's ranking coefficient between the rankings of betweenness scores in consecutive iterations is less than a threshold (i.e., 0.01).

Closeness Centrality

Closeness centrality is another widely used metric based on diameter to identify central nodes in networks. Formally closeness centrality for a node is defined as

$$C(u) = \frac{1}{\sum_{v \in V} d(u,v)} \qquad (2)$$

where $d(u,v)$ is the distance between u and v. Nodes that have small average distance to all other nodes have higher closeness centrality scores. For example, node 3 has a high closeness centrality score in figure 1.

As Maier et al. (2011) proposed, we implement an approximate method to calculate the closeness centrality where, instead of calculating distance to all other nodes, we calculate the distance to a sample of nodes. We again use progressive sampling[12] and calculate the distance to a sample of nodes. As in betweenness centrality, the algorithm converges if the Spearman's ranking coefficient for the closeness centrality score between consecutive iterations is less than a threshold (e.g., 0.01).

4. RESULTS

We first report results from relatively smaller networks to evaluate our distance estimation method. We randomly sample 10,000 pairs, and we plot the frequency of estimated distance with respect to the actual distance. As Maier et

al. (2001) noted, relatively few trees are needed to provide acceptable performance on real world networks.

Figure 2a shows distance estimation for a synthetically generated forest fire network with 10,000 nodes and almost 20,000 edges[8] where we used 10 trees in our NSI. As the figure suggests, for many of the distance estimations, our method can find the actual distance, and for some pairs, our method over estimates. The average stretch of random 10,000 distance estimation calculations is 1.03. Similarly, in figure 3a, we report the same experiment for a wikipedia voting network from Stanford's network datasets[1] with more than 7,000 nodes and more than 100,000 edges where we use 10 trees in our NSI and the average stretch is 1.28. In figure 4a, we have the same experiment for a peer-to-peer file sharing network with more than 8,000 nodes and more than 63,000 edges again from Stanford's network library. We used 20 trees in our NSI, and the average stretch is 1.22. And finally in figure 5a, we report the same experiment on a protein-protein interaction network[2], almost 1,500 nodes and almost 2,000 edges where with 3 trees we get average stretch of 1.28. Our results suggest that with few trees, distance between pairs can be estimated with reasonable accuracy.

We also evaluated our centrality algorithms based on progressive sampling with relatively small networks. Figure 2b shows betweenness centrality rankings for the synthetic forest fire network. Our algorithm converges when the sample size is 50,000 pairs. Most centrality applications, require to identify the top k results, and in figure 2b we report that the approximate method finds the top 20% percent of nodes with high betweenness centrality with a 95% percent accuracy. Also, in figure 3b we report the betweenness centrality approximations by using 32,000 distance pairs for the wikipedia voting network for the top 3000 (roughly top 20%) instances, with 88% accuracy. In figure 5b, we show the top 20% nodes with high betweenness nodes in the protein-protein interaction network with 99% accuracy where we used 8,000 distance pairs to calculate. And finally, in figure 4b, we show the top 20% of the peer-to-peer network with 65% accuracy where we used 16,000 distance pairs. We suggest two ways to increase the accuracy:(1) Increase the number of trees in NSI, (2) Sample more distance pairs.

Figure 2c shows closeness centrality rankings for the synthetic forest fire network. Our algorithm converges when the sample size is 1000 pairs. Also, we report that the approximate method finds the top 20% percent of nodes with high closeness centrality with a 92% percent accuracy by considering the distance to only 1000 random nodes. Also, in figure 3c, we report the closeness centrality approximations for the wikipedia voting network for the top 3000 (roughly top 20%) instances, where we have 76% accuracy by considering distance to 640 random nodes. In figure 5b, we report the top 20% of the nodes with high closeness centrality scores with 66% percent accuracy by considering distance to 40 random sample nodes. Finally in figure 4c, we show the top 20% of nodes with high closeness centrality for the peer-to-peer network with 71% accuracy by considering distance to only 160 random nodes. Again the suggestions above can further increase the accuracy. We conclude that approximation methods can estimate the top k results as well as centrality rankings with reasonable accuracy.

We also report results from a Twitter network with more than 40 million nodes where we used 100 trees for our NSI. Since we can not calculate the exact metrics for such a large network, we only report our observations. Table 2a lists the top 10 Twitter users with the highest betweenness centrality; table 2b lists the top 10 Twitter users with the highest closeness centrality.

5. RELATED WORK

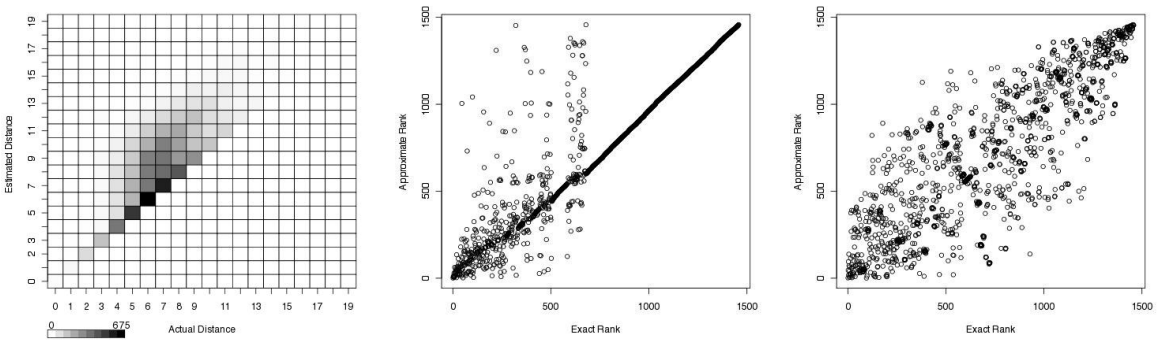
As the size of available networks becomes larger than the memory limits of available commodity computers, distributed frameworks for graph mining are becoming more widely used. One highly regarded framework for mining very large graphs is Hadoop[1], which is an open source implementation of MapReduce parallel processing paradigm[2]. Recently, Kang et al.[6, 7] developed graph mining algorithms for the Hadoop framework, such as finding the diameter, degree distribution and connected components of very large graphs via a generalization of matrix vector multiplication enabling performing basic graph mining tasks on graphs with billions of nodes and edges. Kang et al.[5] also developed an approximate method for different centrality definitions, a more computationally intensive as well as highly popular graph mining task. However, the centrality study modifies the original definitions of betweenness and closeness centrality for efficient computation since the original definitions of centrality, along with many other graph mining tasks, require calculating exact shortest paths between nodes, and such computation is intractable for very large graphs. Also, the newly introduced definition of closeness centrality only works for unweighted graphs. However, the centrality method we propose by utilizing NSIs on MapReduce is loyal to the original definitions of centrality, and applicable to weighted graphs. Also, our proposed distance estimation with NSIs can make computation more tractable than calculating exact distance, enabling more sophisticated graph mining tasks on the Hadoop framework, such as centrality and clustering.

To estimate such low-cost paths between nodes, Rattigan et al.[13] proposed approximation methods by indexing the network structure by dividing the whole network into zones (i.e., sections) and annotating every node with these zones. They then used these annotations to estimate distances. Maier et al.[10] extended NSIs by constructing them through shortest path trees and generalized NSIs to handle weighted graphs with better time and space complexity. Rattigan et al.[14] also applied NSIs to different graph mining tasks including graph clustering and centrality. However, the main assumption of these indexing approaches is that the network should fit into the memory of a workstation, hence these methods can only analyze smaller networks, whereas we propose indexing network structures on the MapReduce framework, which is capable of analyzing networks beyond the memory limits. Thus our approach enables many network mining tasks, such as calculating closeness centrality and betweenness centrality for very large graphs.

Eppstein[3] as well as Peleg and Schaffer[11] also developed methods for approximating low-cost paths between nodes, though they applied their methods to routing algorithms in actual physical networks. However, to the best of our knowledge, such studies also assume that the networks are small enough to fit in the memory of a workstation, and hence they are not applicable to the very large graphs we

[1]http://snap.stanford.edu/data/

[2]http://www.barabasilab.com/rs-netdb.php



(a) Distance estimation (b) Betweenness centrality (c) Closeness centrality

Figure 5: Protein-Protein Interaction Network, 1,458 nodes, 1,993 edges

Table 2: Twitter network with more than 40 million nodes

(a) Betweenness centrality

Screen Name	Name
BarackObama	Barack Obama
aplusk	Ashton Kutcher
cnnbrk	CNN Breaking News
TheEllenShow	Ellen DeGeneres
britneyspears	Britney Spears
Oprah	Oprah Winfrey
MCHammer	Mc Hammer
twitter	Twitter
stephenfry	Stephen Fry
KimKardashian	Kim Kardashian

(b) Closeness centrality

Screen Name	Name
Hamywithfath	Hamy With Fat
ronpark	Ronpark
PrimeTime09	PrimeTime
MrP2theA	Pete
reb4peace	Rebecca Roper
karnner	Narkkie Allie
1cederman	Cederman
Jodie173	Jodie Fagg
sunoneal	Sunil Tolani
ScottHarbuck	Scott Harbuck

focus on in this study.

6. CONCLUSIONS

In this paper, we propose a distance estimation method for the MapReduce parallel processing framework by indexing the network structure. We show that low-cost path distance between pairs in very large networks can be estimated with a reasonable accuracy. We also propose progressive sampling-based centrality algorithms for very large networks that utilize our distance estimation method. We show that, with reasonable accuracy, estimation methods can find the central nodes in very large networks.

In the future work, we want to use our distance estimation methods to develop more sophisticated graph mining tasks as network clustering.

7. ACKNOWLEDGEMENTS

Discussions with Marc E. Maier and Ramesh K. Sitaraman contributed to the paper. Helpful comments and patient editing were made by Cynthia Loiselle. We would like to thank Yahoo! Inc. for the access to the M45 cluster.

8. REFERENCES

[1] Hadoop information. http://hadoop.apache.org/.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[3] D. Eppstein. Spanning trees and spanners. *Handbook of Computational Geometry*, pages 425–461, 2000.

[4] L. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1979.

[5] U. Kang, S. Papadimitriou, J. Sun, and H. Tong. Centralities in large networks: Algorithms and observations. In *SIAM International Conference on Data Mining (SDM) 2011, Mesa, Arizona, USA*, 2011.

[6] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *IEEE International Conference On Data Mining*, 2009.

[7] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: Mining peta-scale graphs. *Knowledge and Information Systems*, 2010.

[8] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 177–187. ACM, 2005.

[9] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[10] M. Maier, M. Rattigan, and D. Jensen. Indexing network structure with shortest-path trees. *ACM Transactions on Knowledge Discovery from Data*, 5(3), 2011.

[11] D. Peleg and A. Schaffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

[12] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 23–32. ACM, 1999.

[13] M. Rattigan, M. Maier, and D. Jensen. Using structure indices for efficient approximation of network properties. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 357–366. ACM, 2006.

[14] M. Rattigan, M. Maier, and D. Jensen. Graph clustering with network structure indices. In *Proceedings of the 24th International Conference on Machine Learning*, pages 783–790. ACM, 2007.