Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 865643, 5 pages
http://dx.doi.org/10.1155/2013/865643

Research Article

Analysis of Average Shortest-Path Length of Scale-Free Network

Guoyong Mao[1,2] and Ning Zhang[3]

[1] *Department of Electronic Information and Electric Engineering, Changzhou Institute of Technology, Changzhou 213002, China*
[2] *German Research School for Simulation Science, 52062 Aachen, Germany*
[3] *Business School, University of Shanghai for Science and Technology, Shanghai 200093, China*

Correspondence should be addressed to Guoyong Mao; gymao@mail.shu.edu.cn

Received 4 April 2013; Revised 15 June 2013; Accepted 17 June 2013

Academic Editor: Michael Chen

Computing the average shortest-path length of a large scale-free network needs much memory space and computation time. Hence, parallel computing must be applied. In order to solve the load-balancing problem for coarse-grained parallelization, the relationship between the computing time of a single-source shortest-path length of node and the features of node is studied. We present a dynamic programming model using the average outdegree of neighboring nodes of different levels as the variable and the minimum time difference as the target. The coefficients are determined on time measurable networks. A native array and multimap representation of network are presented to reduce the memory consumption of the network such that large networks can still be loaded into the memory of each computing core. The simplified load-balancing model is applied on a network of tens of millions of nodes. Our experiment shows that this model can solve the load-imbalance problem of large scale-free network very well. Also, the characteristic of this model can meet the requirements of networks with ever-increasing complexity and scale.

1. Introduction

The research on complex networks is developing at a brisk pace, and significant achievements have been made in recent years [1–4]. Now it is attracting researchers from various areas, including mathematics, physics, biology, computer science, sociology, epidemiology, and others [5].

A scale-free network is a complex network or a connected graph with the property that the number of links originating from a given node exhibits a power law distribution [6]. Many networks are conjectured to be scale-free, including World Wide Web links, biological networks, and social networks.

Like other networks, specific structural features can characterize a scale-free network, among them are degree distribution, average shortest-path length (ASPL), clustering coefficient, and other aspects yet to be explored. Average shortest-path length is a concept in network topology that is defined as the average number of steps along the shortest paths for all possible pairs of network nodes. It is a measure of the efficiency of information or mass transport on a network. Some examples are the average number of clicks which will lead you from one website to another, or the number of people you will have to communicate through on average, to contact a complete stranger.

The average shortest-path length is defined as follows. Consider a network G with the set of vertices V. Let $\mathrm{dist}(v_1, v_2)$ denote the shortest distance between v_1 and v_2 ($v_1, v_2 \in V$). Assume that $\mathrm{dist}(v_1, v_2) = 0$ if $v_1 = v_2$ or v_2 cannot be reached from v_1, $\mathrm{has\_path}(v_1, v_2) = 0$ if $v_1 = v_2$ or if there is no path from v_1 to v_2, and $\mathrm{has\_path}(v_1, v_2) = 1$ if there is a path from v_1 to v_2; then the average shortest-path length ASPL_G is

$$\mathrm{ASPL}_G = \frac{\sum_{i,j}^{N} \mathrm{dist}\left(v_i, v_j\right)}{\sum_{i,j}^{N} \mathrm{has\_path}\left(v_i, v_j\right)}. \tag{1}$$

Here N denotes the number of nodes in G, $\sum_{i,j}^{N} \mathrm{dist}(v_i, v_j)$ is the value of all-pairs shortest-path length of graph G, and $\sum_{i,j}^{N} \mathrm{has\_path}(v_i, v_j)$ is the number of paths that exist in graph G. For a connected undirected graph, $\sum_{i,j}^{N} \mathrm{has\_path}(v_i, v_j) = N(N-1)$, because paths exist between any pair of nodes.

For unweighted directed networks, the time complexity is $O(N * (N + E))$ for computing the all-pairs shortest-path length using breadth-first-search algorithm [7], where E is the number of edges in G. For weighted directed network, the time complexity is $O(N^3)$ using Floyd algorithm [8]. If the network has millions of nodes or edges, the time needed in computing will be unbearable. Hence, parallel algorithm must be used to effectively reduce the computing time.

All-pairs shortest-path length is the sum of single-source shortest-path length (SSSPL) of each node in G. There are two types of parallel methods to compute all-pairs shortest-path length, the fine-grained parallelization method and the coarse-grained parallelization method. In fine-grained parallelization, the computing of the SSSPL for each node is accomplished using multiple cores to reduce the computing time [9–11]. Therefore, the time needed in getting the sum of the SSSPL of every node can be reduced as well. While in coarse-grained parallelization, suppose that there are N nodes in G and P cores available for computing; then one core is responsible for the computing of the SSSPL of N/P nodes. This kind of parallelization will not reduce the time needed in computing the SSSPL of one node, but ideally, the time needed in computing the all-pairs shortest-path length can be reduced to $1/P$ of that of a serial algorithm.

Compared with fine-grained parallelization, coarse-grained parallelization is much easier to implement on most parallel computers, because only a reduction operation is needed after all cores have finished their assignments of computing the SSSPL of their respective N/P nodes. In fine-grained parallelization, apart from a reduction operation in the end, a lot of scheduling and communication operations will be needed in computing the SSSPL of each node. These operations will decrease the performance of parallelization, as there are more and more computing nodes available to us, and the complexity and scale of networks are ever increasing. For any realistic scenario, the number of nodes in large scale-free networks will be significantly larger than the number of cores used by the parallel algorithm. Therefore, the coarse-grained method is also efficient because no computing resources are wasted.

Based on the observation mentioned previously, we will use coarse-grained parallelization to compute the all-pairs shortest-path length of large scale-free networks. It is a typical single-program-single-data (SPSD) parallelization; that is, program and data are the same in different cores; only the range of source nodes are different. Load imbalance might be the only problem, which is caused by the difference of time in computing the SSSPL of different nodes, as different cores are assigned different source nodes. Hence, in coarse-grained parallelization, the problem is how to schedule different source nodes to different cores such that the load is balanced across all cores.

Little work can be found on the load-balancing problem of this coarse-grained parallelization, due mainly to the fact that the research on scale-free networks was not popular until the end of last century. Even in the past decade, due to the large scale of network and restriction in computing power, the computing of the average shortest-path length was mainly achieved through approximation [12–15]. But nowadays, thanks to multicore and manycore technologies, the development in parallel computing has made it possible for accurate computing of such a problem.

We only concentrate on scale-free networks here, as in other networks, for example, the small-world network generated using WS and NW models [16] and the random network generated using the ER model [17], load imbalance is not a big problem for coarse-grained parallelization, because the computing time of the SSSPL for different nodes does not vary significantly.

The remainder of this paper is organized as follows. The influence of outdegree on computing time of the SSSPL is discussed in Section 2. The load-balancing model is discussed in Section 3. The evaluation of this model in large scale-free networks is discussed in Section 4, before discussing conclusion and future work in Section 5.

2. Analysis of the Outdegree on Computing Time

We only study the directed complex networks in this paper because most of the networks that naturally exist are directed. The SSSPL problem is to find the shortest-path length from a source node to all other nodes in the network or graph. It is easy to see that if a node has an outdegree of 0, the SSSPL will be 0 as well, because no path exists from this node to all other nodes in this network. Therefore, these nodes can be safely excluded from the job assignment without affecting the average shortest-path length. But apart from this, we can hardly find other relations between the feature of a node and its SSSPL. For example, if nodes with outdegree of 1 are selected, the corresponding computing time of the SSSPL is illustrated in Table 1. The network in Table 1 is taken from [1], and we name it BA network hereafter. It contains 325,729 nodes and 1,797,134 edges, among them 21,941 nodes have outdegree of 1.

In Table 1, the first column is the ratio of computing time of these nodes to the maximum computing time of the node in the whole network. We can see that the time needed in computing the SSSPL of these nodes varies greatly. If we randomly assign these nodes to different processors, the load balancing will become a problem.

However, we know that if one node has large outdegree, which means that many nodes are connected to this node, the possibility of a long computing time of the SSSPL of this node will be high. For example, there are 1,882 nodes with outdegree larger than 900 in BA network, among them 1,760 nodes have computing time larger than $1e-2$; that is, more than 93.5% of the computing time of these nodes can be regarded as long. To express this more clearly, we give the detailed information about the relation between threshold of outdegree and computing time in BA network in Table 2.

The average outdegree of this network is about 10.85. From Table 2, we can see that bigger outdegree can in some sense reflect longer computing time. A similar situation can be observed for China education network in 2004 [18, 19], as listed in Table 3. We name it edu04 network hereafter, in which the average outdegree is 2.81.

However, there are still many nodes that have a long computing time with small outdegree. In Table 1, there are

Table 1: The relation between time span and number of nodes with outdegree 1.

Time span	Number of nodes	Number percentage
$(1e-1, 1)$	5989	27.29
$(1e-2, 1e-1)$	9667	44.05
$(1e-3, 1e-2)$	924	4.21
$(1e-4, 1e-3)$	791	3.61
$(1e-5, 1e-4)$	2359	10.76
$(1e-6, 1e-5)$	2211	10.08

Table 2: Threshold of outdegree and computing time of BA network.

Threshold of outdegree	Number of nodes	Number of nodes with computing time bigger than 0.01	Percentage
>10	28181	21652	76.83
>27	11720	10096	86.14
>47	5147	4666	90.65
>900	1882	1760	93.52

15,656 nodes having outdegree of 1, but their computing times is also long (the computing time is bigger than $1e-2$). We know that if one node has a small outdegree (e.g., 1), but this node is connected to a node with a large outdegree, then the possibility of a long computing time of the SSSPL of this node will be high. Also, if one node has a big outdegree, but all its neighbors have a small outdegree, then the possibility of a short computing time of the SSSPL of this node will also be high. Therefore, the computing time of the SSSPL of one node is related not only to its outdegree but also to the outdegree of its neighbors of the ith $(i = 1, 2, 3, \ldots)$ level. The neighbor of the first level is the direct neighbor of this node and the neighbor of the second level is the neighbor of the direct neighbor of this node, and so on.

The outdegree of the neighbors can be expressed in two terms, the sum of outdegree and the average outdegree Avg(i). Obviously, the latter can better express the connectivity of the neighboring nodes. For unweighted graph, the average outdegree Avg(i) of the neighbor of node i is

$$\operatorname{Avg}(i) = \frac{1}{|N_i|}\sum_{j\in N_i} k_j, \tag{2}$$

where N_i are the neighbors of node i and k_j is the outdegree of a node j that belongs to N_i. We will discuss the overall impact factor of each level of the average outdegree on the computing time of the SSSPL in the next section.

3. The Dynamic Programming Model of the Load-Imbalance Problem

Based on the analysis mentioned previously, we present the dynamic programming model of the load-balancing problem. Suppose that there are N nodes in graph G, D_{ij} is the average outdegree of the jth level neighbor of node i $(i = 1, 2, 3, \ldots, N,\ j = 0, 1, 2, 3, \ldots)$, D_{i0} is the outdegree of

Table 3: Threshold of outdegree and computing time of edu04 network.

Threshold of outdegree	Number of nodes	Number of nodes with computing time bigger than 0.01	Percentage
>3	20881	8225	39.39
>10	9472	4875	51.47
>50	1068	848	79.41
>100	268	232	86.56

node i, T_i is the computing time of the SSSPL of node i, P is the number of computing cores, C_j is the coefficients of D_{ij}, and $\sum C_j = 1$. We sort T_i using the value of $\sum C_j D_{ij}$; then the load-balancing problem is changed into computing C_j such that $\sum_k^N T_i$ $(k = 1, 2, 3, \ldots, P,\ i = i+P)$ are as equal as possible, where $\sum_k^n T_i$ is the total computing time in core k. The sorted nodes with id starting from k and step size P are assigned to core k, and N/P nodes are assigned to each core.

3.1. Implementation of the Load-Balancing Model. The serial computing time of the SSSPL of each node and the average outdegree of different levels of each node are measured using NetworkX [20] and Python [21]. To make the goal of this model more clear, we use $\operatorname{Max} T$ to denote the maximum sum of computing time among P cores, $\operatorname{Min} T$ to denote the minimum sum of computing time among P cores, and time difference $= (\operatorname{Max} T - \operatorname{MinT})/\operatorname{Max} T$. Then, the goal of this model can be expressed as finding the coefficients such that time difference takes the minimum among all possible values.

We only compute the average outdegree till the second level, as it will take too long if the level goes deeper. Hence, the sum of $C_0 * D_{i0} + C_1 * D_{i1} + C_2 * D_{i2}$ is used as the index value to sort T_i, where $C_0 + C_1 + C_2 = 1$, $0.1 \le C_j \le 0.8$, and the variation of C_j is 0.1; there are 36 combinations for C_j $(j = 0, 1, 2)$. We use different numbers of cores and different coefficients to compute the time difference of the BA and edu04 networks. The results show that time difference will be the smallest when $C_0 = 0.6$, $C_1 = 0.3$, and $C_2 = 0.1$. Figure 1 illustrates the relation between time difference and number of cores of these two networks, where the x-axis is the number of cores and y-axis time difference. Different values of time difference of BA network with coefficients 0.6, 0.3, and 0.1 are described using solid circles. The values of edu04 network with the same coefficients are described using up triangles.

3.2. Analysis of the Model. From the analysis of these two networks, we get two results. First, when the average degree of a network (the number of edges divided by the number of nodes) is large, the time difference is relatively small. Second, when the number of cores is small, the difference is also very small. For example, the difference is 0 when 2 cores are used. The difference will be larger when more cores are used. The first characteristic of this model can meet the demands of the ever-increasing complexity of the network, because

Table 4: Time difference for different coefficients and number of cores.

	Coefficients		
Cores	0.6, 0.3, 0.1	0.7, 0.2, 0.1	0.5, 0.3, 0.2
8	1.55%	1.48%	1.65%
16	2.37%	2.49%	4.21%
32	4.12%	4.81%	6.34%

the network is becoming more and more complex, and the average degree is getting bigger and bigger. For example, the average degree of the edu04 network is only about 1.5. But in 2008, it has grown into a network with 2,508,811 nodes and 25,278,346 edges [22], and the average degree reaches nearly 10. For the second characteristic, we did experiments on networks of various sizes and found out that the real cause is the decrease of the number of nodes assigned to each core when more cores are used. In other words, for a fixed number of cores, the difference will become smaller when the network is getting larger. We know that the network is growing very fast. Therefore, this characteristic of the model can meet the requirements of the ever-increasing scale of networks.

We also use this load-balancing model on other scale-free networks of various sizes. These networks are generated using BA model with different parameters [1]. We find out that when the ratio of the number of nodes over the number of cores (N/P) is getting larger, the coefficients are not unique to get the acceptable time difference values, but in every scenario, the difference to the best is very small. For example, when 16 cores are used in a network with 500,000 nodes and 1,500,000 edges, N/P being 31,250, there are 2 combinations of coefficients to get similar minimum time difference values; when 8 cores are used in this network, N/P being 62,500, there will be 3 combinations of coefficients to get similar minimum values. The time difference values related to different numbers of cores and coefficients are listed in Table 4. We can see that the coefficients of 0.6, 0.3, and 0.1 might not be the best, but they can still be safely used when the ratio is getting bigger. Similar situations can be found in other networks with different parameters.

For large-scale complex networks, it will take too long to compute the average neighbor outdegree of the second level. Hence, we simplify this model into just one level; that is, only outdegree and the average outdegree of the neighbor of the first level are considered. That is, the sum of $C_0 * D_{i0} + C_1 * D_{i1}$ is used as the index value to sort T_i, $C_0 + C_1 = 1$, $0.1 \leq C_j \leq 0.9$, the variation of C_j is also 0.1. The results show that the difference will be the least when $C_0 = 0.8$ and $C_1 = 0.2$. Different values of TimeDifference of BA network with these coefficients are described using down triangles in Figure 1, the values of edu04 network with the same coefficients are described using rectangles.

4. Evaluation

There are various algorithms for the computation of the ASPL, for example, NetworkX [21]. However, if the network is large, it will take a large amount of memory to represent the network. The network we used in the experiment has 2,508,811 nodes and 25,278,346 edges, and the memory needed to represent this network would be around 74 G if NetworkX were used, much bigger than the memory capacity of a workstation. It is obviously impossible to parallelize the serial algorithm because each core will get a copy of the whole network during the parallel execution process.

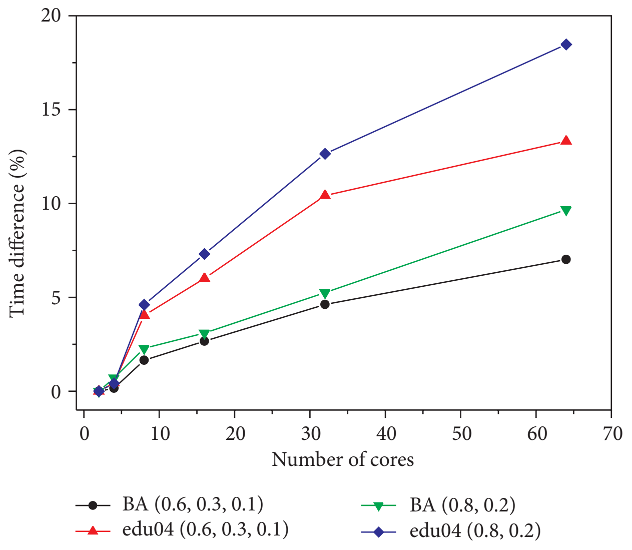

Figure 1: Relation between time difference and number of cores.

To solve this space-complexity problem, we used a native 2-dimensional array to represent the network, where the first row in the array is the start node of an edge and the second row is the target node. Also, as there are many search operations in the Breadth-First-Search algorithm, we transform the array into a C++ multimap using the start node as the key and all nodes connected to this node as the values of this key. With this data structure, the memory needed for loading the same network is reduced to only 1.48 G. Also, these structures can significantly speed up the keywords-based fast index, as all data in this map is sorted.

With the simplified load-balancing model, we compute the average shortest-path length of the network with 2,508,811 nodes on a cluster of 6 Dell PowerEdge workstations that have 72 cores; the operating system is CentOS 5.5 64 bit, and the compiler is Intel C++ 11.1. We use MPI to implement job assignment and to get the time used on each core, and the average computation time is 537,721 seconds or about 6 days and 5.5 hours. The time difference between the maximum time and the minimum time is only about 1.43%.

5. Conclusion and Future Work

The main contribution of this paper is the dynamic programming model that can be used to solve the load-balancing problem in coarse-grained parallel computing of average shortest-path length problem of large scale-free network. Though we only test the model with networks of up to tens of millions of edges, the feature of this model can make it scalable to networks with more complexity and larger scale.

In this paper, we presented our reason of using coarse-grained parallelization to compute the average shortest-path length. We analyzed the relationship between the computing time of the SSSPL of node and the features of nodes and presented a dynamic programming model to reduce the time difference among all computing cores. We used time measurable networks to get the coefficients of this model and applied this model to large scale-free network with satisfactory result.

In the future, we will evaluate this model with various kinds of networks to determine the relation between the features of the network (e.g., the average degree) and the coefficients. With this relation, we can distribute the nodes more evenly among cores to achieve better load balance and make this model more scalable to networks of different types.

Acknowledgments

The authors are grateful to colleagues in German Research School for Simulation Science for their constructive suggestions. The authors are also grateful to the reviewers for their valuable comments and suggestions to improve the presentation of this paper. This work is supported by the National Natural Science Foundation of China under Grant no. 70971089, Shanghai Leading Academic Discipline Project under Grant no. XTKX2012, and Jiangsu Overseas Research and Training Programs for University Prominent Young and Middle-Aged Teachers and Presidents.

References

[1] R. Albert, H. Jeong, and A. Barabási, "Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[2] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[3] A.-L. Barabási, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A*, vol. 272, no. 1, pp. 173–187, 1999.

[4] R. Albert and A. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 47–97, 2002.

[5] A. E. Motter and R. Albert, "Networks in motion," *Physics Today*, vol. 65, pp. 43–48, 2012.

[6] S. H. Strogatz and D. J. Watts, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[8] Floyd and W. Robert, "Algorithm 97: shortest path," *Communications of the ACM*, no. 5, p. 345, 1962.

[9] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of Dijkstra's shortest path algorithm volume," in *Mathematical Foundations of Computer Science (MFCS)*, vol. 1450 of *Lecture Notes in Computer Science*, pp. 722–731, 1998.

[10] J. L. Träff and C. D. Zaroliagis, "A simple parallel algorithm for the single-source shortest path problem on planar digraphs," *Journal of Parallel and Distributed Computing*, vol. 60, no. 9, pp. 1103–1124, 2000.

[11] K. Madduri, D. A. Bader, W. B. Jonathan, and J. R. Crobak, "An experimental study of a parallel shortest path algorithm for solving large-scale graph instances," in *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics (ALENEX '07)*, pp. 23–35, New Orleans, La, USA, January 2007.

[12] Q. Ye, B. Wu, and B. Wang, "Distance distribution and average shortest path length estimation in real-world networks," in *Proceedings of the 6th International Conference on Advanced Data Mining and Applications*, vol. 6440, pp. 322–333, 2010.

[13] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis, "Fast shortest path distance estimation in large networks," in *Proceedings of the 18th International Conference on Information and Knowledge Management (CIKM '09)*, pp. 867–876, November 2009.

[14] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum, "Fast and accurate estimation of shortest paths in large graphs," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM '10)*, Toronto, O'ntario, Canada, October 2010.

[15] l. Su, N. Zhang, and L. Ma, "A new approximation algorithm for finding the shortest path in large networks," *Complex System and Complex Science*, vol. 5, no. 2, pp. 51–54, 2008.

[16] X. F. Wang and G. Chen, "Complex networks: small-world, scale-free and beyond," *IEEE Circuits and Systems Magazine*, vol. 3, no. 1, pp. 6–20, 2003.

[17] P. Erdős and A. Rényi, "On the evolution of random graphs," *Mathematical Institute of the Hungarian Academy of Sciences*, no. 5, pp. 17–61, 1960.

[18] N. Zhang, "Demonstration of complex network: CERNET," *Journal of Systems Engineering*, vol. 21, no. 4, pp. 337–340, 2006 (Chinese).

[19] N. Zhang, "Analysis of china education network structure," *Computer Engineering*, vol. 33, no. 17, pp. 140–142, 2007 (Chinese).

[20] "NetWorkX1. 7, Official website for NetWorkX," http://networkx.lanl.gov/index.html.

[21] "Python2. 7, Official website of python," http://www.python.org/index.html.

[22] N. Zhang, L. Su, and L. Ma, "Comparative studies on the topological structure of China Education Network," *Journal of University of Shanghai for Science and Technology*, vol. 30, no. 3, pp. 297–299, 2008 (Chinese).