Predictive Partitioning for Efficient BFS Traversal in Social Networks

Damien Fay

Abstract In this paper we show how graph structure can be used to significantly reduce the computational bottleneck of the Breadth First Search algorithm (the foundation of many graph traversal techniques) for social networks. In particular, we address parallel implementations where the bottleneck is the number of messages between processors emitted at the peak iteration. First, we derive an expression for the expected degree distribution of vertices in the frontier of the algorithm which is shown to be highly skewed. Subsequently, we derive an expression for the expected message along an edge in a particular iteration. This skew suggests a weighted, iteration based, partition would be advantageous. Empirical simulations show that such partitions can reduce the message overhead in the order of 20 % *for graphs with common social network structural properties*. These results have implications for graph processing in multiprocessor and distributed computing environments.

Keywords BFS · Graph structure · Social network properties

1 Introduction

Breadth First Search (BFS) is a fundamental graph algorithm which is applied constantly to huge social network graphs in distributed and parallel systems consuming large amounts of energy and resources. BFS is central to several more complicated graph algorithms such as identifying connected components, testing for bipartiteness, belief propagation, finding community structures in social networks and computing the max flow-min cut for a graph [1]. As such it has drawn much attention from the parallel processing community as a benchmark algorithm with several competing variants focused on efficient implementation [1–14]. However, despite its importance *known structural properties* of social networks have not been leveraged to improve the algorithms efficiency. The aim of this paper is to *prove the concept* that a simple adjustment of the partitioning vector based on common graph structure can

D. Fay (✉)
Department of Computing, Bournemouth University, Poole, UK
e-mail: dfay@bournemouth.ac.uk

H. Cherifi et al. (eds.), *Complex Networks VII*, Studies in Computational Intelligence 644, DOI 10.1007/978-3-319-30569-1_2

greatly improve the efficiency at the algorithms bottleneck. We also show (theoretically and empirically) that it is the graph properties that result in this improvement, and *absence* of these properties can lead to little or no improvement. The graph theoretic analysis of the BFS frontier in this paper is novel and should be of interest to researchers in the parallel and distributed graph traversal communities.

The setting here envisages that BFS is performed repeatedly on an unweighted, undirected graph from random root vertices. In addition, we assume basic statistics about the graph can be collected after each run or alternatively offline. It is assumed the graph is traversed in parallel by several processors thus requiring a-priori a partition of the graph vertices across each processor. In this setting the basic computation step of BFS is dominated by the communications costs (messages) between processors after each iteration (as noted in [6] amongst others). The messages emitted after the peak iteration further dominate the communication costs amounting to $\sim$70 % of the total (Sect. 4), thus this is the *bottleneck* of the whole algorithm.

With the exception of a few papers (Sect. 2) most approaches ignore information about the structure of the graph focusing instead on CPU-GPU architecture specifics. We show that the incident degree distribution per iteration is highly skewed away from a power law distribution. Thus the number of edges crossing a partition is a biased estimate of the messages between partitions at the peak iteration. Further we propose a new weighted graph construction which reflects the expected number of messages per edge. Finally, we show empirically that using a weighted partitioning algorithm that the subsequent reduction in messages emitted across partitions can be reduced in some individual cases by $\sim$50 %, for some graphs on average by $\sim$20 %.

The paper is laid out as follows. Section 2 discusses related work, Sect. 3 gives the background behind the BFS algorithm, partitioning and develops the theory showing that the degree distributions are highly skewed. Section 4 presents empirical results and finally in Sect. 5 we mainly focus on future work and discussing the consequences of the findings.

2 Related Work

Implementing BFS in parallel is a well established approach which generally consists of three stages: graph pre-ordering, graph partitioning and parallel architecture specific implementation. This research is most pertinent to graph partitioning however there are several aspects of architecture specifics of interest.

Graph partitioning seeks to reduce the number of messages sent between partitions during processing which can be achieved in several ways. The most obvious mechanism is to use a 1-D partition; each vertex and associated edges are sent to an individual processor [1, 4, 9, 15]. An excellent overview of 1-D graph partitioning methods can be found in [13] with techniques designed specifically for scale-free networks exist such as [16]. Although [16] considers partitioning for social network graphs they do not do so in the context of BFS, indeed the two approaches are complimentary as here we provide a weighted social network graph for partitioning.

Shang and Kitsuregawa [4] consider partitioning edges across processors (as opposed to vertices). The edges may be uniformly distributed by either the source or the target vertex. They propose that when the degree of the target vertex exceeds a pre-defined threshold the algorithm performs best by switching to a target vertex partitioning, while Hong et al. [17] note that for low degree vertices partitioning should be based on vertex but for large degree vertices the partitioning should be based on edge. In contrast a 2-D partition [2, 8, 10, 11] distributes the edges of a vertex across several processors. The 2-D approach is based on the observation that an exploration from a set of vertices is equivalent to the product of the adjacency matrix and a vector of the vertices touched. Thus they partition the adjacency matrix into two dimensions (blocks along the rows and columns) and then collect the row products in one set of messages and the unique column entries in another. Thus the messages produced are between particular processors and not *all to all* as in the 1-D case. It would appear from the literature that the 2D partitioning approach results in more efficient BFS traversals but we do not consider this approach in this research (see future work, Sect. 5).

Skewed graph structure is a central topic in many papers [1, 2, 5, 12, 17]. The non-locality of neighbours in a graph, and the fact that some vertices can have degrees several factors larger than the average, leads to load imbalances across processors and random memory access patterns. Yuan et al. [12] examines the expected distance between two pairs of nodes being explored in a BFS and show that they can predict the vertex locality. This is perhaps the closest work to this research. In contrast our approach looks at the expected use of a vertex of a given degree in a particular iteration, though the two approaches are similar in spirit. Alternative approaches include implementing BFS from multiple sources [18]. However, to the best of our knowledge this research is the first to use the non-uniform frontier distribution to improve the parallel BFS algorithm.

3 Background

Given a graph $G(V, E)$ and a source vertex s, where V, E refer to the vertex and edge sets respectively the BFS algorithm returns a route from s to every reachable vertex in G. The BFS algorithm begins with a set $V_0 = \{s\}$ and explores the graph by identifying neighbours of s, denoted as the set V_0^+, where + denotes neighbour expansion. At the next iteration all vertices connected to V_0^+ minus those already visited are $V_1 = V_0^+ \setminus \{V_0\}$. We call the set of unique vertices in the τth iteration, V_τ, the *frontier* set. In general the frontier consists of

$$V_\tau = V_{\tau-1}^+ \setminus \{\bigcup_{i=0}^{\tau-1} V_i\} \qquad (1)$$

and the set of vertices already visited, $\{\bigcup_{i=0}^{\tau-1} V_i\}$, are said to be *touched*. The algorithm continues until $V_\tau = \{\emptyset\}$ and all vertices have been explored.

The BFS algorithm may be implemented on P parallel processors by partitioning V into P subsets $\mathscr{V}_1, \ldots, \mathscr{V}_P$, where $\mathscr{V}_i \bigcap \mathscr{V}_j = \{\emptyset\} \quad \forall i \neq j$, and $\bigcup_i \mathscr{V}_i = V$, such that each vertex is assigned a processor which performs the neighbour expansion of that vertex. This is the basic format of most parallel BFS (P-BFS) algorithm implementations. At the end of each iteration the processor owning each element in the next frontier must be notified that this vertex is now to be explored. We define a message $\mathscr{M}^\tau_{\mathscr{V}_i \to \mathscr{V}_j}(u, v)$ to be a notification from processor i to processor j that vertex u has identified vertex v to be a member of the next frontier set. If u and v reside in the same processor then there is no communication cost and thus the communications cost for P-BFS is here defined as the sum of all messages that cross a partition:

$$C^\tau = \sum_{u \in V_{\tau-1}, v \in V_\tau} \mathscr{M}^\tau_{\mathscr{V}_i \to \mathscr{V}_j}(u, v) \quad \forall i \neq j \tag{2}$$

The aim of a partitioning algorithm is to partition a graph into equal sets, $|\mathscr{V}_i| \approx |\mathscr{V}_j|$, such that a specific objective is achieved such as the number of edges that cross the partitions, the *edge-cut*, is minimized as:

$$\underset{\mathscr{V}_1, \ldots, \mathscr{V}_P}{\operatorname{argmin}} C = \sum_{u \in \mathscr{V}_i, v \in \mathscr{V}_j, \forall i \neq j} w_{u,v} \tag{3}$$

There are several methods for graph partitioning (a recent review of such methods may be found in [13]) and the one adopted here is the popular METIS [19] multi-level k-way algorithm. Like many partitioning algorithms METIS can operate on weighted graphs; the weights themselves are the core of our technique as now discussed.

The development here initially follows that of Kurant et al. [20] who derive expressions for the *observed* degree distribution of a graph sampled by BFS (i.e. a different problem). The configuration model [21] is a construct which allows construction of graphs with a desired degree distribution. N vertices are each assigned k *stubs* sampled uniformly from a desired degree distribution, p_k, i.e. $k \sim p_k$. The configuration model then pairs these stubs at random thus constructing edges and thus a graph with the desired degree distribution. The order in which these stubs are connected is irrelevant as the pairing is random. Thus we may assign to each stub an arbitrary time, $t \in [0, 1]$ and moving from $t = 0 \to 1$ connect the stubs as their randomly assigned time is passed. This converts a discrete graph generation process into a continuous time process and is a useful framework to derive expressions for the bias inherent in BFS sampling [20, 22]. Kurant et al. interweave the stub matching step with the exploration phase of BFS. Thus the stubs are connected only when the frontier is being explored and the unconnected stubs with the lowest time are chosen first. A vertex enters the frontier when all of its stubs have been paired and this happens with probability $(1 - t)^k$ therefore the expected fraction of vertices of degree k touched before time t is [20]:

$$f_k(t) = p_k(1 - (1 - t)^k) \tag{4}$$

where p_k is the probability a vertex has degree k (i.e. the degree distribution). The fraction of nodes of any degree visited before time t is [20]:

$$f(t) = 1 - \sum_k p_k(1 - (1 - t)^k) \tag{5}$$

Kurant investigated the bias of BFS samples but here we are concerned with the degree distribution of the frontier. In addition, we are only interested in particular times; those that correspond to the iterations. It is assumed that the number of vertices touched up to each iteration $n_\tau = \sum_{i=1}^{\tau} |V_i|$ is known.[1] Here we depart from Kurant's analysis [20]. Define times, t_τ:

$$t_\tau = f^{-1}(n_\tau / N) \tag{6}$$

where f^{-1} denotes the inverse of $f(t)$ as (5) cannot be inverted explicitly. This inverse consists of finding the minimum of a smooth function in one dimension and may be solved easily using gradient descent or any similar search algorithm. The frequency of degrees of type k in the τth frontier, n_k^τ, can be calculated iteratively by removing those seen in the previous frontiers as:

$$n_k^\tau = \frac{p_k(1 - (1 - t_\tau)^k)}{\sum_k p_k(1 - (1 - t_\tau)^k)} n_\tau - \sum_{i=1}^{\tau-1} p_k^i n_i \tag{7}$$

where $n_0 = 0$ and p_k^τ is the frontier degree distribution defined as:

$$p_k^\tau = \frac{n_k^\tau}{\sum_k n_k^\tau} \tag{8}$$

The probability that a vertex of degree k is used in the frontier is then the number of vertices of degree k in the frontier divided by the total number in the graph:

$$\pi_k^\tau = \frac{p_k^\tau n_\tau}{p_k N} \tag{9}$$

Figure 1 shows $f_k(t)$ for $p_k \propto k^{-2}$.[2] Up to iteration 3, 25 % of the degrees touched are of degree 1 which rises to ∼50 % by iteration 5. That is, BFS is biased (proportionately) towards higher degree vertices initially, moving towards lower degree vertices

[1] A good estimate of the number of vertices expected in each iteration of BFS can be obtained from a single graph traversals.

[2] Here we use the YouTube friendship graph as an example: the power law exponent = -2 and t_τ = $\{0.0006, 0.02, 0.19, 0.53, 0.81, 0.93, 0.97, 0.99, 1\}$, the results are similar for the other graphs we examined.

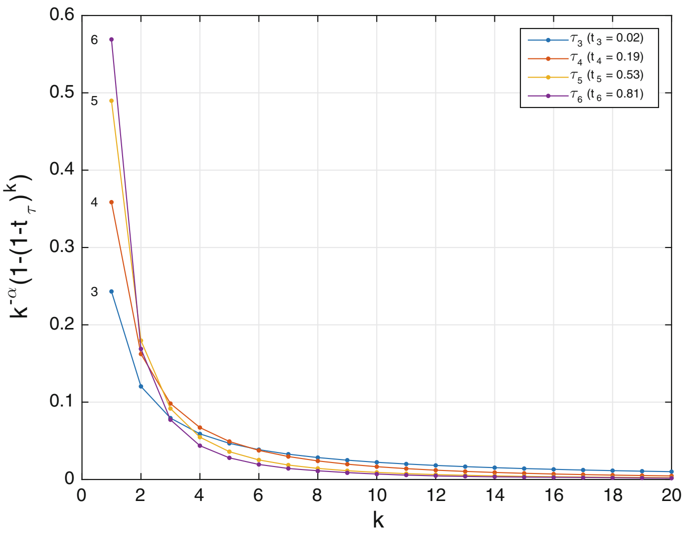

Fig. 1 Proportion of vertices of degree k seen before iteration τ ($\alpha = -2$)

at later iterations. Note that Fig. 1 shows the *accumulated* proportion as the algorithm progresses, however, it is the difference in these proportions that are touched at each iteration and this has a very different shape (Fig. 2).

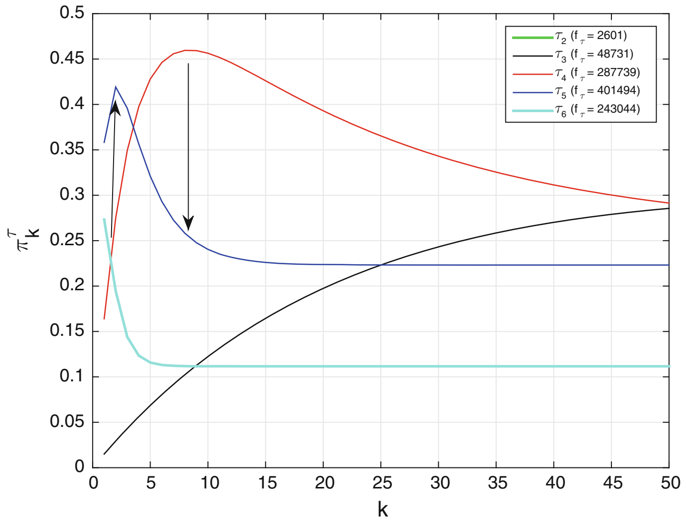

Fig. 2 Probability a vertex of degree k will be used in iteration τ (theoretical)

Figure 2 shows π_k^τ for the YouTube friendship graph (Sect. 4). The distribution of nodes used in iterations 2 and 3 is biased towards high degree nodes. In iteration 4 the bias centres on vertices of degree 10 with 40 % being touched but only 15 % of degree 1 nodes are touched. In iteration 5 the bias switches, ∼40 % of degree 1 vertices are touched but only ∼25 % of degree 10 vertices are touched. There is a similar switch between iteration 5 and 6. The interesting thing about this behaviour is that the degree distribution of vertices used is highly skewed and during the main iterations (4, 5, 6) those used in one iteration tend not to be used in the next and visa versa (as illustrated with arrows in Fig. 2). Thus at a specific iteration we have a prior probability over the vertices that will be used *and* a different prior over the vertices they are connected to in the next frontier, and these distributions are different from the initial power-law distribution, i.e. $\pi_k^\tau \neq \pi_k^{\tau+1} \not\propto p_k$. The transition from $\pi_k^\tau \rightarrow \pi_k^{\tau+1}$ involves connecting vertices with degree distribution π_k^τ to those with $\pi_k^{\tau+1}$. It would be tempting to assume that the probability of a node of degree k connects to a node of degree k' is just the product of π_k^τ and $\pi_k^{\tau+1}$, however the two events are not independent. Real-world graphs are generally assortative and as has been shown graph generators that take into account the correlation structure in the joint degree distribution $p_{k,k'}$ produce far better approximations to real-world graphs [21] and have very different properties from those that assume independence [23]. Here we assume that the joint degree distribution, $p_{k,k'}$, [21] gives a good approximation of the expected edges between the vertices in iteration τ and $\tau+1$, therefore we may define the probability of transitioning from a vertex with degree k to an edge with degree k' in iteration τ, $p_{k,k'}^\tau$ as:

$$p_{k,k'}^\tau = \pi_k^\tau p_{k,k'} \pi_k^{\prime\tau+1} \tag{10}$$

The probability of using a particular edge, $\{u, v\}$, in iteration τ is equal to the probability of passing from $u \rightarrow v$, or from $v \rightarrow u$ but not both, $u \leftrightarrow v$, as this would imply u and v have already been touched in iteration τ, therefore:

$$w_{k,k'}^\tau = p_{k,k'}^\tau + p_{k',k}^\tau - p_{k,k'}^\tau p_{k',k}^\tau \tag{11}$$

where $w_{k,k'}^\tau$ can be used to weight each edge in G where the weights represents the expected message along that edge in iteration τ. The total number of expected messages given a particular partition is then:

$$E[C^\tau] = \sum_{u \in V_\tau, v \in V_{\tau+1}} w_{k_u,k_v}^\tau \mathscr{I}_{\mathscr{V}_i \rightarrow \mathscr{V}_j}(u, v) \tag{12}$$

where $\mathscr{I}_{\mathscr{V}_i \rightarrow \mathscr{V}_j}(u, v)$ is an indicator variable s.t. $u \rightarrow v$ crosses a partition. To implement this approach requires estimates of; $p_k, p_{k,k'}, n_\tau$. Given these a weighted version of G, $W(V, E)$, may be constructed, and partitioned using a weighted partitioning algorithm (here we use the popular METIS algorithm).

4 Results

The simulations presented below consist of randomly choosing a source node, performing a BFS using the competing algorithms (described below), and recording the number of messages generated. Code and examples may be found on the project webpage.[3] The simulations are based on 500 randomly chosen root vertices. There are four competing algorithms which represent different levels of knowledge:

1. The original graph with no weighting is used as a baseline,
2. Using the results from 1. We calculate the actual messages counts and use these to give an empirical weighted matrix, W_{emp}. Note that in essence we are using the answer to derive the partition which is unrealistic. The aim here is to give an upper bound on the algorithms performance,
3. Using the $p^{\tau}_{k,k'}$ from all 500 iterations we combine and smooth these estimates to produce a single weighted graph called, W_{smooth}. The aim here is to give an estimate of performance without the approximation error inherent in Eq. 9, and
4. Using Eqs. (9,11,12) we form a single weighted graph, W_{avg}.

The joint degree distribution, $p_{k,k'}$, can present problems of storage and estimation especially when the maximum degree is high. However, as the graphs studied have a power law distribution, the number of vertices with a high degree falls rapidly. In this paper we calculate $p_{k,k'}$ where nodes with $k \geq 300$ are counted in a single bin. Therefore, $p_{k,k'}$ is formed of a, 300×300 grid. We choose the number of partitions to be 100 as this reflects the order of processors in a GPU (the number of processors varies greatly depending on the machine; the NVIDIA GeForce GTX280, for example, has 30 [9] while the NVidia Kepler architecture has 4,096 GPU's [10]).

The datasets used in this study are taken from the Konect graph repository.[4] We are specifically interested in social network graphs and so the RMAT graphs used in studies such as [1, 7] are not included though we do include a synthetically generated ER graph with a single large component. We also did not consider graphs with $N > 2$M for computational reasons. These graphs are listed in Table 1.

Figure 3 shows the empirical distribution of π^{τ}_{k} (based on a sample of 500 random root nodes) for the YouTube Graph versus the theoretical (Fig. 2). As can be seen for low degrees the approximation is excellent but deviates at higher degrees, especially during iteration 4. This occurs because high degree nodes in real networks cluster together in the network core (breaking the uniform assumption in the configuration model). That said, most nodes in power-law network are of low degree where the approximation is excellent and as will be seen the results are not effected adversely.

Figure 4 shows the average number of messages per iteration using the 4 algorithms above applied to the YouTube graph. As can be seen the three weighted graph versions perform better than the unweighted graph. The average number of messages (over all iterations) transmitted using W_{avg} is the lowest at 681 K while those for the unweighted graph are 790 K. The results differ on closer inspection however.

[3]https://sites.google.com/site/structuralgraphproperties/home.

[4]http://konect.uni-koblenz.de.

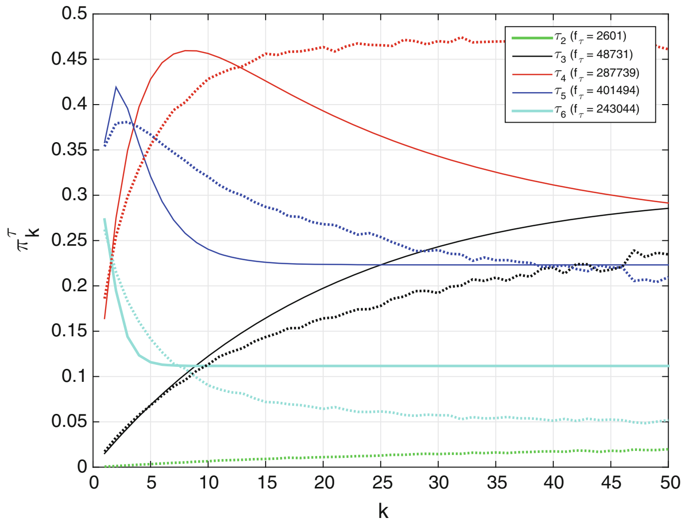

Fig. 3 π_k^τ theoretical (*solid*) versus empirical (*dashed*) (YouTube Graph)

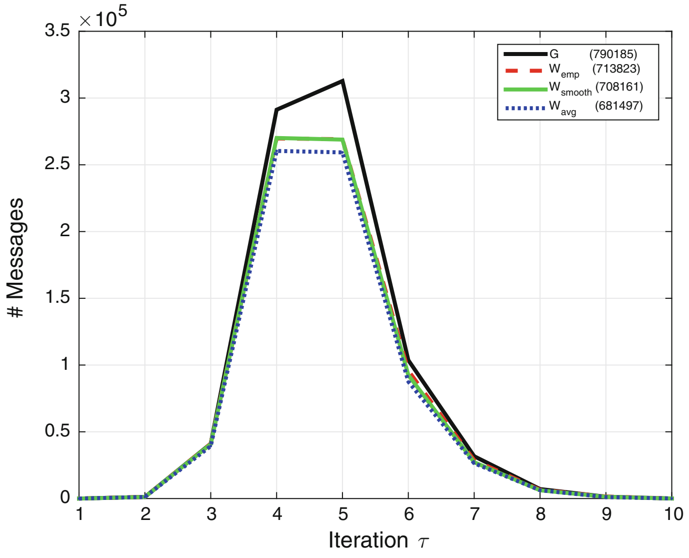

Fig. 4 Average number of messages per iteration (YouTube; totals in brackets)

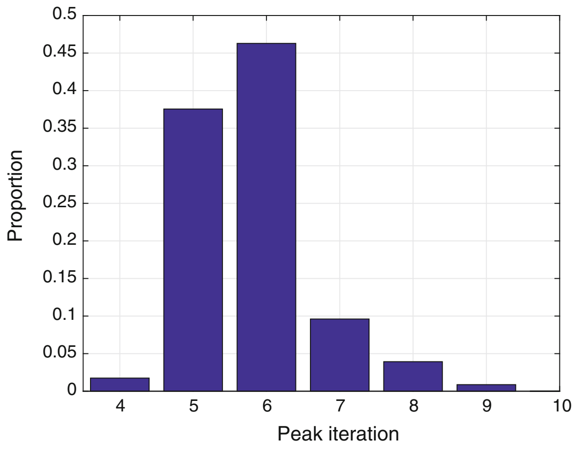

Fig. 5 Histogram of iteration at which the number of vertices in the frontier reached a peak

Figure 5 shows the histogram of the iteration at which the peak iteration occurred in each BFS run. For most source vertices the iteration at which the number of vertices in the frontier reaches a peak is 5 or 6.

Figure 6 shows the distribution of messages for a particular root node and as can be seen here the peak occurs at iteration 4 and the number of messages in the peak far

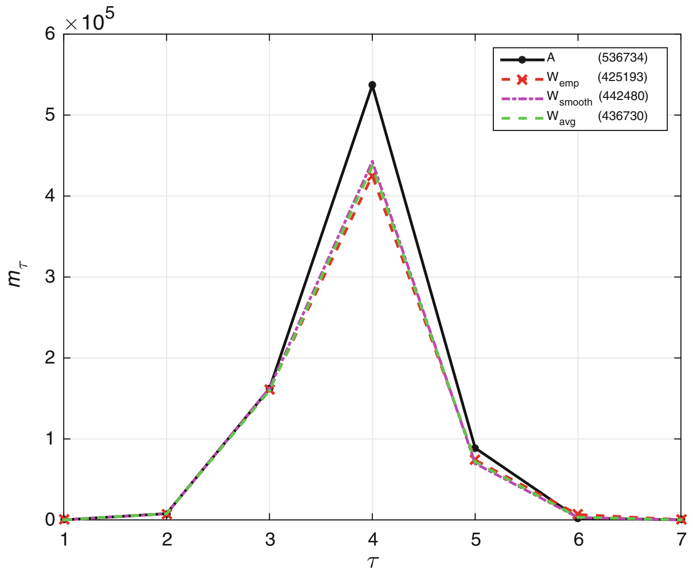

Fig. 6 Example showing number of messages per iteration for YouTube graph (root u=157, 298)

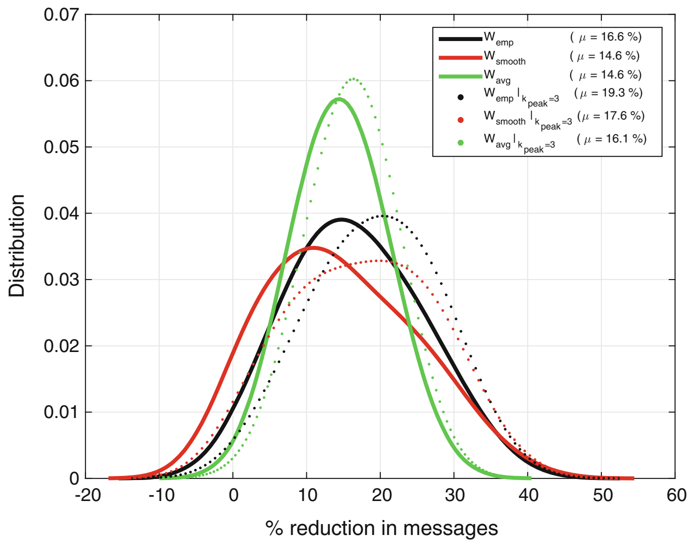

Fig. 7 Distribution of reduction for YouTube dataset (the distribution using those with peak ≥ 6 is shown using the *dotted line*, 500 samples)

exceeds those in the other iterations. Next we turn our attention to how the algorithm *performs* relative to the baseline. Figure 7 shows the *percentage improvement in messages* over the baseline algorithm. The savings are in the order of 15 % for this graph which is quite significant. In this particular case the three algorithms perform reasonably similarly but note that W_{avg} leads to the lowest improvement in messages at the peak but interestingly the highest improvement in the overall number of messages (Fig. 4).

Figure 8 shows the improvements observed with the Epinions graph. Here there is a distinct bi-modal distribution, with one distribution centred around 4 % and another centred ~35 %. For this graph about half the iterations peak at $\tau = 3$ and the remainder at $\tau = 4$. If we look at the improvement for those that peak at $\tau = 3$ alone then a clearer picture emerges. For these vertices the improvement is very small (the 4 % mode in the distribution). One possibility is that vertices which reach the peak at $\tau = 3$ lie in the core of the graph and have less hops to the periphery; thus the BFS algorithm has less time to achieve the random mixing assumed in Eq. 11 (Kurant similarly notes that the starting vertex can significantly effect their estimates [20]).

Next we examine a graph with no structure, an Erdos Renyi (ER) graph, where the joint degree distribution is uniform and the degree distribution is concentrated around the mean. As there is no structure in the graph we expect the algorithm to fail and this is exactly what is seen in Fig. 9.[5] The % (dis)improvement is a distinctive Gaussian distribution centred on zero. Moving onto a collection of graphs, Table 1 summarizes

[5]Alternatively one could insert a concentrated degree distribution for p_k in (4) and see that $\pi_k^{tau} = p_k$.

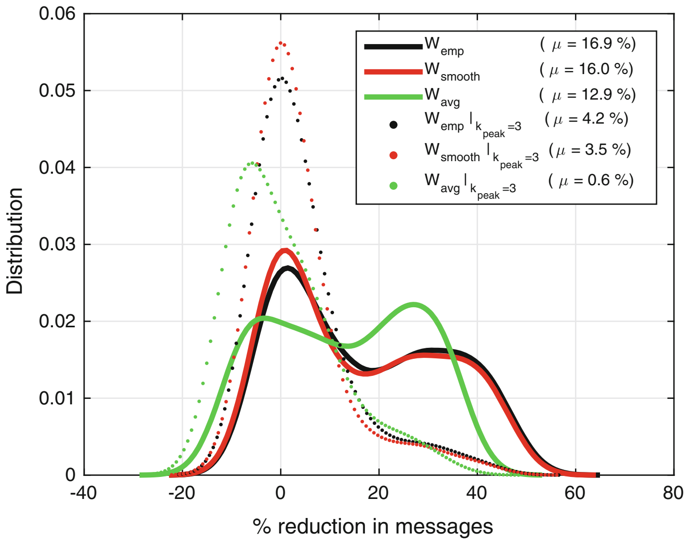

Fig. 8 Distribution of reduction for epinions dataset (the distribution using those with peak ≤ 3 is shown using the *dotted line*)

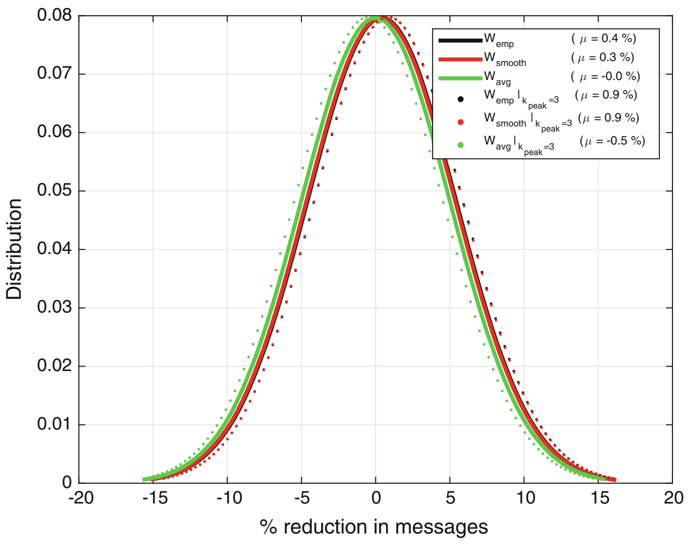

Fig. 9 Distribution of reduction for ER dataset (the distribution using those with peak ≥ 3 is shown using the *dotted line*)

Table 1 Summary of results for a collection of graphs (http://konect.uni-koblenz.de) values for ρ in brackets exclude core nodes, results averaged over 500 simulations

Name	Type	$\lvert V \rvert$	$\lvert E \rvert$	r	ρ_{emp} %	ρ_{smooth} %	ρ_{avg} %
YouTube friendship	Social	1,134,890	2,987,624	−0.03	16.57 (12.59)	14.61 (10.18)	14.59 (12.37)
Epinions	Social	75,879	508,837	−0.04	16.9 (21.4)	15.9 (20.3)	12.8 (17.2)
Gowalla	Social	196,591	950,3279	−0.02	11.79 (10.38)	9.52 (8.02)	7.2 (6.62)
DBLP	Coauthorship	1,314,050	18,986,618	0.10	8.75 (8.76)	8.11 (7.99)	6.74 (6.56)
Wikipedia En	Hyperlink	1,853,493	39,953,145	−0.05	7.75 (15.15)	6.69 (13.62)	4.80 (13.96)
Catster friendship	Social	149,700	5,449,275	−0.16	4.62 (3.09)	3.51 (2.36)	3.86 (2.61)
Google	Hyperlink	875,713	5,105,039	−0.05	−3.14(−3.75)	−0.83(−0.34)	−0.52(−0.89)
ER graph	Synthetic	100,000	1,151,281	0.00	0.41 (0.23)	0.34 (0.14)	−0.01 (0.16)

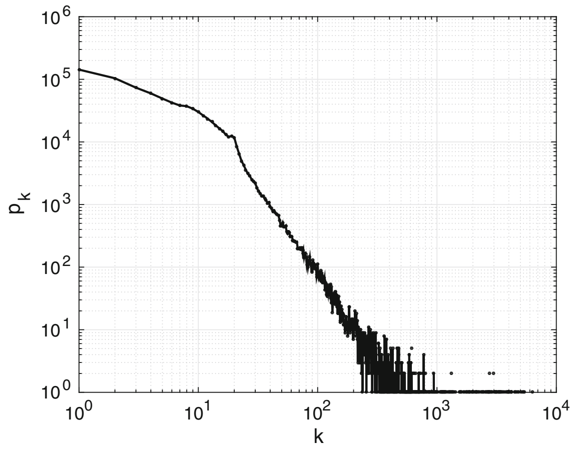

Fig. 10 Degree distribution for Google hyperlink graph (*loglog scale*)

our results. These results are quite mixed; for some graphs the reduction in messages can be very significant and in the order of $\sim$15 % while for others it can be quite low. For the epinions and YouTube graphs the improvement is 12.80 and 14.59 % on average which is not far from the upper bound of 16.90 and 16.57 %. For the Catster, Wikipedia, and DBLP graph the results are reasonable and in the region of 5 % (3.8, 4.8 and 6.7 %). The Google graph does not show any improvement as the degree distribution for this graph is not power law (Fig. 10). While the distinctive power law tail exists the distribution for low degree nodes is more uniformly distributed breaking the underlying assumption required for the algorithm to work.

For the Epinions graph the result for the non-core vertices increases to 17.20 % but for the YouTube graph it actually decreases to 12.37 %. For the DBLP graph, there is no difference. For Wikipedia the difference is quite significant with non-core vertices reporting a reduction in messages up from 4.80 to 13.96 %. The main conclusion here is that the position of a vertex in the graph certainly has an effect on the performance but it is unclear what the effect will actually be.

5 Conclusion

Social networks with power law characteristics are an important and common class of real-world graphs. This paper has clearly demonstrated the principle that their structure can be leveraged to improve the efficiency of BFS; in some cases significantly by up to 20 %. The computational overhead is minimal; the quantities required for the algorithm to work $\{p_k, p_{k,k'}, n_\tau\}$ can be easily estimated from an initial burn in period (several BFS runs). Future work will look at extending this approach to weighted, directed graphs, we also note that as vertices and edges are added to a real-world graph its degree distribution does not change rapidly and so there is scope for application in dynamic and streaming graph analysis. The skew present in π_k^τ is such that (the standard) unweighted edge partition is not optimal for any iteration. This is why in Fig. 4 we see that the total number of messages (not just at the peak) can also be significantly reduced.

As the techniques mentioned in the Related work (Sect. 2) are not graph structure dependent, it would be interesting to examine if the highly skewed BFS frontier statistics can be usefully incorporated. Future work will investigate a GPU implementation, collecting a taxonomy of graphs for which the technique gives significant improvement and integration with 2-D approaches for improved performance. Finally, further work is required to determine why the algorithm works better for some start vertices, if those vertices can be identified in advance, and in a computationally efficient manner. It is also possible that Eq. 4 could be made conditional on known information about the root vertex.

References

1. Merrill, D., Garland, M., Grimshaw, A.: Scalable GPU graph traversal. In: 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, vol. 47, no. 8, pp. 117–128, Feb 2012
2. Buluç, A., Madduri, K.: Graph partitioning for scalable distributed graph computations. Contemp. Math. **588**, 83 (2013)
3. Krzywdzinski, K., Kowalski, D.: On the complexity of distributed BFS in ad hoc networks with spontaneous wake-up. Discrete Math. Theor. Comput. Sci. **15**(3), 101–118 (2013)
4. Shang, H., Kitsuregawa, M.: Efficient breadth-first search on large graphs with skewed degree distributions. In: Joint 2013 EDBT/ICDT Conferences, EDBT'13 Proceedings, Genoa, Italy, March 18–22, 2013, pp. 311–322 (2013)
5. Chen, R., Shi, J., Chen, Y., Chen, H.: PowerLyra: differentiated graph computation and partitioning on skewed graphs. In: Proceedings of the Tenth European Conference on Computer Systems, EuroSys'15, pp. 1:1–1:15. ACM, New York, NY, USA (2015)
6. Fu, Z., Dasari, H., Berzins, M., Thompson, B.: Parallel breadth first search on GPU clusters. SCI Institute, University of Utah, SCI Technical report UUSCI-2014-002 (2014)
7. Wang, Y., Owens, J.D.: Large-scale graph processing algorithms on the GPU. UC Davis, Technical report (2013)
8. Buluç, A., Beamer, S., Madduri, K., Asanović, K., Patterson, D.: Distributed-memory breadth-first search on massive graphs. In: Bader, D. (ed.) Parallel Graph Algorithms. CRC Press, Taylor-Francis (2016). https://www.crcpress.com/Parallel-Graph-Algorithms/Bader/9781466573260
9. Luo, L., Wong, M., Hwu, W.-M.: An effective GPU implementation of breadth-first search. Proceedings of the 47th Design Automation Conference. DAC'10, pp. 52–55. ACM, New York, NY, USA (2010)
10. Bisson, M., Bernaschi, M., Mastrostefano, E.: Parallel distributed breadth first search on the Kepler architecture (2014). arXiv:1408.1605
11. Buluç, A., Madduri, K.: Parallel breadth-first search on distributed memory systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, pp. 65:1–65:12. ACM, New York, NY, USA (2011)
12. Yuan, L., Ding, C., Tefankovic, D., Zhang, Y.: Modeling the locality in graph traversals. In: ICPP. IEEE Computer Society, pp. 138–147 (2012)
13. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning (2013). arXiv:1311.3144

14. Akiba, T., Iwata, Y., Yoshida, Y.: Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD'13, pp. 349–360. ACM, New York, NY, USA (2013). http://doi.acm.org/10.1145/2463676.2465315
15. Idwan, S., Etaiwi, W.: Computing breadth first search in large graph using hmetis partitioning. Eur. J. Sci. Res. **29**(2), 215–221 (2009)
16. Abou-Rjeili, A., Karypis, G.: Multilevel algorithms for partitioning power-law graphs. Proceedings of the 20th International Conference on Parallel and Distributed Processing. IPDPS'06, pp. 124–124. IEEE Computer Society, Washington, DC, USA (2006)
17. Hong, S., Kim, S.K., Oguntebi, T., Olukotun, K.: Accelerating CUDA graph algorithms at maximum warp. SIGPLAN Not. **46**(8), 267–276 (2011)
18. Then, M., Kaufmann, M., Chirigati, F., Hoang-Vu, T.-A., Pham, K., Kemper, A., Neumann, T., Vo, H.T.: The more the merrier: Efficient multi-source graph traversal. In: Proceedings of VLDB Endow., vol. 8, no. 4, pp. 449–460 (2014). http://dx.doi.org/10.14778/2735496.2735507
19. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**, 96–129 (1998)
20. Kurant, M., Markopoulou, A., Thiran, P.: On the bias of BFS (breadth first search). In: 22nd International Teletraffic Congress (ITC), pp. 1–8 (2010)
21. Mahadevan, P., Hubble, C., Krioukov, D., Huffaker, B., Vahdat, A.: Orbis: rescaling degree correlations to generate annotated Internet topologies. SIGCOMM Comput. Commun. Rev. **37**(4), 325–336 (2007)
22. Achlioptas, D., Clauset, A., Kempe, D., Moore, C.: On the bias of traceroute sampling: or, power-law degree distributions in regular graphs. In: STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, May 2005, pp. 694–703
23. Fay, D., Haddadi, H., Uhlig, S., Kilmartin, L., Moore, A.W., Kunegis, J., Iliofotou, M.: Discriminating graphs through spectral projections. Comput. Netw. **55**(15), 3458–3468 (2011)