I/O-efficient Hierarchical Diameter Approximation. *

Deepak Ajwani[1]**, Ulrich Meyer[2], and David Veith[2]

[1] Centre for Unified Computing, University College Cork, Cork, Ireland
[2] Institut für Informatik, Goethe-Universität Frankfurt, Robert-Mayer-Str. 11–15, D-60325 Frankfurt am Main, Germany

Abstract. Computing diameters of huge graphs is a key challenge in complex network analysis. As long as the graphs fit into main memory, diameters can be efficiently approximated (and frequently even exactly determined) using heuristics that apply a limited number of BFS traversals. If the input graphs have to be kept and processed on external storage, even a single BFS run may cause an unacceptable amount of time-consuming I/O-operations.
Meyer [17] proposed the first parameterized diameter approximation algorithm with fewer I/Os than that required for exact BFS traversal. In this paper we derive hierarchical extensions of this randomized approach and experimentally compare their trade-offs between actually achieved running times and approximation ratios. We show that the hierarchical approach is frequently capable of producing surprisingly good diameter approximations in shorter time than BFS. We also provide theoretical and practical insights into worst-case input classes.

1 Introduction

Massive graph data stemming from social networks or the world wide web have implicitly become part of our daily life and also means big business. Consequently, a whole branch of computer science (and industry) deals with network analysis [8]. For connected undirected unweighted graphs $G(V, E)$ where $n = |V|$ and $m = |E|$, the distance $d(u, v)$ between two nodes $u, v \in V$ is the number of edges in the shortest path connecting u and v. The eccentricity of a node v is defined as $\mathrm{ecc}(v) = \max_u d(v, u)$. A fundamental step in the analysis of a massive graph is to compute its diameter $D := \max_{u,v} d(u, v) = \max_v \mathrm{ecc}(v)$.

We focus on the case when G is sparse ($m = O(n)$) but nevertheless too big to fit into the main memory of a single computing device. In this situation one can either distribute the data over many computers and apply parallel algorithms (e. g., see [6]) and/or store the data on secondary memory like hard disks or flash memory. In this paper we will concentrate on improved diameter

* Partially supported by the DFG grant ME 3250/1-3, and by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.
** Research of this author is supported by the EPS grant from IRCSET and IBM

approximation algorithms for the second (external memory) approach.

External memory model. The huge difference in data access times between main memory and disks is captured in the external memory (EM) model (aka I/O model) by Aggarwal and Vitter [1]. It assumes a two level memory hierarchy: The internal memory is fast, but has a limited size of M elements (nodes/edges). The external memory (of potentially unlimited size) can only be accessed using I/Os that move B contiguous elements between internal and external memory. The computation can only use data currently kept in internal memory. The cost of an algorithm is the number of I/Os it performs – the less, the better. The number of I/Os required to scan n contiguously stored elements in the external memory is $\text{scan}(n) = O(n/B)$ and the number of I/Os required for sorting n elements is $\text{sort}(n) = O(\frac{n}{B} \log_{M/B} n/B)$. For realistic values of B, M, and n, $\text{scan}(n) < \text{sort}(n) \ll n$. Frequently, the goal of designing external memory algorithms for sparse graphs is to reduce the I/O complexity from $\Omega(n)$ to not much more than $O(\text{sort}(n))$.

2 Related Work

The traditional approaches for computing the exact diameter rely on computationally expensive primitives such as solving the all pair shortest path problem (APSP) or fast matrix multiplication algorithms. Thus, many heuristics (e.g., [11, 15, 7, 10, 12]) have been proposed to approximate the diameter. However, these heuristics still rely on BFS and SSSP traversals, which are very expensive when the graph does not fit in the main memory.

In the external memory, network analysis algorithms have been developed for specific graph classes (e.g., [14] for naturally sparse graphs). For general undirected graphs, algorithms for the exact computation of the diameter [5, 9] require $\Theta(n \cdot \text{sort}(n))$ I/Os for sparse graphs ($m = O(n)$) and are thus impractical. We are interested in approaches that work for general undirected graphs and require even less I/Os than BFS, as even carefully tuned BFS implementations [4] require many hours on graphs with billions of edges. In this context, Meyer's parameterized approach [17] achieves efficient trade-offs between approximation quality and I/O-complexity in theory and can give good approximation bounds even when the desired I/O complexity is restricted to be less than that of external memory BFS algorithms [16].

Parameterized approach. The parameterized approach decomposes the input undirected graph into many small-diameter clusters and then contracts each cluster to a single node forming a condensed graph. This condensed graph preserves the structure of the input graph and in particular, the diameter of the condensed graph approximates the diameter of the input graph. The condensed graph is typically much smaller than the input graph. If it fits internally, its diameter can be approximated using internal memory heuristics; otherwise a semi-external or fully external memory single source shortest path (SSSP) from an arbitrary vertex can be used to obtain a good approximation.

To compute small-diameter clusters, the parameterized algorithm first selects some vertices to be masters uniformly at random and then "grows" the clusters around the selected master vertices "in parallel". In a parallel cluster growing round, each cluster tries to capture all its unvisited neighbors, with ties being broken arbitrarily. Each such round is simulated in the external memory with a constant number of scanning and sorting steps.

For undirected, unweighted graphs with n nodes and $m = O(n)$ edges, an $O(\sqrt{k} \cdot \log n)$ approximation for the diameter can be obtained with high probability (whp) using $O(n \cdot \sqrt{\log k/(k \cdot B)} + k \cdot scan(n) + sort(n))$ I/Os by contracting the input graph to n/k vertices. Thus, if we strive for $O(n/B^{2/3})$ I/Os, we should contract the graph to $O(n/(B^{1/3} \cdot \log B))$ vertices and expect multiplicative errors of $O(B^{1/6} \cdot \sqrt{\log B} \cdot \log n)$ in the computed diameter whp.

In a recent workshop paper, Ajwani et al. [3] have shown that this parameterized approach produces much better approximation bounds on many different graph classes with varying diameters than its worst-case approximation guarantee would suggest. Also, if the number of vertices in the contracted graph is carefully chosen, the parameterized approach is significantly faster than the various internal memory heuristics for diameter approximation, even if the BFS routine in those heuristics is replaced by its external memory counterpart. This makes the diameter approximation of graphs with a few billion edges, a viable task in external memory on typical desktop machines.

3 Extending the Parameterized Approach

Hierarchical extension. The scalability of the parameterized approach is still limited. For very large graph sizes, either one needs to pick a very large value of k to contract the graph enough to be able to run semi-external SSSP and thus settle for a poor approximation ratio or one has to depend on the more I/O-intensive primitive of fully-external SSSP. Also, we are not aware of any efficient implementation for computing SSSP in fully-external memory. A potential approach to scale the parameterized approach further is to extend it by contracting the graph recursively. The difficulty in such an extension lies in the fact that after the first iteration, the graph becomes weighted and it is not clear how to I/O-efficiently contract a weighted graph while minimizing the approximation ratio of the resultant diameter. We consider different ways to select master vertices (and their numbers) for weighted graphs and study the approximation quality of the resultant diameter on various graph classes.

Bad graph class. Our first attempt for selecting the master vertices in the weighted graphs is to completely ignore the edge weights occurring from graph shrinking and simply select masters uniformly at random and with the same probability in all recursive levels. Already for two levels of graph contraction the multiplicative approximation error for the diameter might grow to $\Omega(k \cdot \sqrt{k} \cdot \log n)$: a factor of $O(\sqrt{k} \cdot \log n)$ based on the analysis of [17] for unweighted graphs, exacerbated by the fact that after the first shrinking, edges may represent a weight of $\Omega(k)$. Surprisingly, it will turn out in Section 4 that this weight-oblivious master selection rule is often capable to yield reasonable approximation bounds. Nevertheless, in Section 4.2, we also present a sophisticated artificial graph class where the computed diameter is provably a factor $\Omega(k^{4/3-\epsilon})$ away from the actual diameter using two recursive levels and the simple master selection. Note that after the second level, the number of clusters is reduced to $O(n/k^2)$. On the other hand, if we would have directly chosen master vertices with probability $1/k^2$ using the non-recursive parameterized approach, we could have achieved an approximation factor of $O(k)$. Thus, on this graph class, the diameter computed by the recursive extension of the parameterized approach is significantly worse than the one computed directly by the parameterized approach.

Adaptive reduction in graph size. Our first approach described above reduces the number of vertices by the same (expected) multiplicative factor in each recursive level. Since in the later recursive levels, the size of the graph is smaller, it might be beneficial to perform more aggressive reduction by performing extra scanning rounds to minimize the number of recursive levels required to reduce the size of the graph enough for it to fit in the main memory. Ideally, the reduction factor should depend not only on the graph size and the available internal memory, but also on the graph type. This is because the condensed graphs from different classes have very different sizes. For instance, if we contract a linear chain with 2^{28} vertices and $2^{28}-1$ edges to 2^{20} vertices, we get $2^{20}-1$ edges (the size reducing by almost 256). On the other hand, we found that when we reduced a $\sqrt{n}$-level random graph (described in Section 4) of 2^{28} vertices and 2^{30} edges to 2^{20} vertices, there were still more than 2^{28} edges in the graph (the size reducing by a factor less than 4).

Our eventual goal is to design a software for approximating the diameter of large graphs without assuming any a priori information about the graph class. Thus our scheme needs to learn the graph class on-the-fly and use it to adapt the number of master vertices and consider weights, too.

Selecting the master vertices. To improve the approximation quality without increasing the I/O complexity and adapt the number of master vertices, we experimented with different probability distribution functions for selecting the master vertices. We found that selecting the i^{th} vertex to be a master vertex in k^{th} round with a probability p_i as defined below, provides a good diameter approximation for a bad graph class of the two-level recursive approach. Let W_i be the set of weights of edges incident to a vertex i and let max_i be the maximum value in the set W_i. Let n_k (m_k) be the number of vertices (edges) in the graph before the k^{th} round of contraction, $sum\_max = \sum_{i=1}^{n_k} max_i$, $max\_max = \max_{i=1}^{n_k} max_i$ and $min\_max = \min_{i=1}^{n_k} max_i$. Then the probability p_i is equal to $\alpha(\cdot)\cdot(((max_i/sum\_max)\cdot n_k)-min\_max)/(max\_max-min\_max)$. Here, α is an adaptability function that infers the rate of graph contraction based on the past contraction history (values of $m_0,\ldots,m_{k-1},m_k;n_0,\ldots,n_{k-1},n_k$) and then adapts the number of master vertices n_{k+1} in the k^{th} recursive level based on the history and the expected graph size m_{k+1} from this level. We restrict the graph size in the final level to fit in the main memory.

Modeling the graph contraction. We considered various inference functions for modeling the graph contraction. Our first model assumed a direct proportionality between the number of vertices and edges, i.e., $n_{k+1} = n_k \cdot m_{k+1}/m_k$. Later, we integrated more complex linear and non-linear models such as $m_{k+1} = A \cdot n_{k+1} + B$ and $m_{k+1} = A \cdot n_{k+1} + B \cdot \sqrt{n_{k+1}}$ in our adaptability function, learning the values of parameters A and B based on the (limited) past contraction history of the graph.

Tie breaking. In the parallel cluster growing approach, ties are broken arbitrarily among the clusters vying for the same vertex. To reduce the weighted diameter of the clusters in the second recursive level, ties can be broken in favor of the cluster that has the shortest distance from the vertex to its master.

Move master vertices. We use another trick to further reduce the weighted diameter of the clusters. Once the clustering is done using parallel cluster growing, we move the master vertices closer to the weighted center of their corresponding clusters and re-compute the distances between the clusters based on the new master vertices. Since the clusters fit in the internal memory, it requires only one scanning step to load all clusters in the main memory and compute the distances of all vertices from the master vertex of their cluster.

Once the contracted graph fits into the main memory, we can use any internal memory technique to approximate the diameter of the contracted graph. In this work, we use the double sweep lower bound [15] technique that first computes a single-source shortest path from an arbitrary source s and then returns the weighted eccentricity of a farthest node from s. Other techniques (e.g., [11]) can also be used for this purpose.

We study the relative merits of these ideas and show that by carefully tuning our implementation based on these ideas, we can approximate the diameter of graphs with many billions of edges in a few hours. Note that these graph sizes are significantly bigger than that of past experiments reported in the literature, even for external memory algorithms.

4 Experiments

Since external memory experiments on large graphs can take many hours and even days, a certain self-restraint in the number of such experiments is unavoidable. As such, we performed our initial experiments for analyzing the approximation quality of various extensions of the parameterized algorithm on a machine with 64 GB RAM, using an internal memory prototype. For the variant that gave the best approximation, we implemented it for optimized I/O performance in external memory. The external memory implementation relies on the STXXL library [13], exploiting the various features supported by STXXL such as pipelining and overlap of I/O and computation. The running time and I/O volume reported in the paper are based on external memory experiments.

Also, we restrict ourselves to extensions involving only two levels of recursion. We found that for the graph sizes that we considered, we could get acceptable running time and a good approximation ratio with two recursive levels.

4.1 Configuration

For experiments in internal memory we used a machine from the HPC cluster at Goethe University on graphs with 256 million vertices and about 1 billion edges. For our external memory experiments, we used an Intel dual core E6750 processor @ 2.66 GHz, 4 GB main memory (around 3.5 GB was available for the application) and four hard disks with 500 GB. Only the 250 GB from the outer tracks were used in a RAID-0.

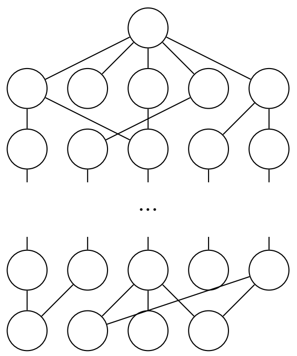
4.2 Graph Classes

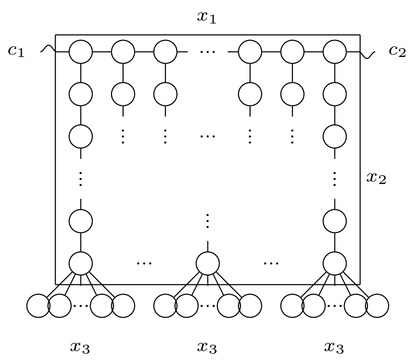
We chose four different graph classes: one real-world graph with logarithmic diameter, two synthetic graph classes with diameters $\Theta(\sqrt{n})$ and $\Theta(n)$ and a graph class that was designed to elicit poor performance from the simple extension of the parameterized approach. Recall that the simple extension (hereafter referred as Basic) chooses the master vertices at different recursive levels with the same probability.

The real-world graph sk-2005 has around 50 million vertices, about 1.8 billion edges and is based on a web-crawl. It was also used by Crescenzi et al. [11] and has a known diameter of 40. The synthetic x-level graphs are similar to the B-level random graphs in [2]. The graph consists of x levels, each having $\frac{n}{x}$ vertices (except the level 0 containing only one vertex). The edges are randomly distributed between consecutive levels, such that these x levels approximate the BFS levels if BFS were performed from the source vertex in level 0. The edges are evenly distributed between different levels. We selected $x = \sqrt{n}$ and $x = \Theta(n)$ to generate $\sqrt{n}$ and $\Theta(n)$-level graphs for our experiments. Figure 1a illustrates an example of such a graph.

While the basic recursive extension of the parameterized approach with uniform master probabilities already yields reasonable approximation ratios on many graph-classes including real-world data, it is possible to design artificial inputs that cause significant approximation errors. Before we sketch the construction of such a graph class (referred worse 2step), we consider worst-case inputs for the standard non-recursive approach with master probability $1/k$. Figure 1b displays a graph consisting of a main chain C_0 with x_1 nodes. Each of these x_1 nodes is connected to a side chain C_i of length x_2 that ends with a fan of size x_3. The diameter of this graph is $\Theta(x_1 + x_2)$.

For $x_1 \cdot x_2 \leq k^{1-\epsilon}$ and constant $0 < \epsilon \ll 1$, there is at least a constant probability that master vertices of the non-recursive approach only appear at the very ends of the side chains (i.e., in the fans, outside of the marked box). Furthermore, if the value of x_3 is chosen sufficiently large ($\Omega(k \cdot \log n)$), each fan receives at least one master with high probability. Thus, with at least constant probability, for each side chain a cluster is grown from its fan towards the main


(a) Example of a x-level graph.



(b) Worst-case input for the parameterized approach.

Fig. 1: Various graph classes used in our experiments.

chain C_0 and all these clusters reach C_0 simultaneously and then stop growing. Therefore, the shrunken weighted graph features a path with $x_1 - 1$ edges of weight $\Theta(x_2)$ each, resulting in a diameter of $\Theta(x_1 \cdot x_2)$. Choosing $x_1 = k^{1/2}$ and $x_2 = k^{1/2-\epsilon}$, the expected multiplicative approximation is therefore $\Omega((x_1 \cdot x_2)/(x_1 + x_2)) = \Omega(k^{1/2-\epsilon})$, asymptotically matching the upper bound proved in [17]. Larger graphs with similar behavior can be obtained by chaining several copies of the graph structure discussed above along the zigzag lines (using simple paths of length $\Theta(x_1 + x_2)$ each).

For the recursive graph shrinking approach with $i \geq 2$ levels of shrinking we would like to construct an input class that after the first $i - 1$ levels resembles the worst-case graph for the non-recursive version and has accumulated huge edge weights in the side chains but not on the main chain. For ease of exposition we only sketch the 2-level case with master probability $1/k$ for both shrinking phases: (1) edges of weight $O(1)$ in the shrunken graph G' can be obtained with high probability from two high-degree fans that are connected by an edge in the input graph G. (2) simple isolated paths of length $\Theta(z \cdot k)$ in G will result in paths of total weight $\Theta(z \cdot k)$ distributed over an *expected* number of $\Theta(z)$ edges in G'. Appropriate fans in G' are obtained with high probability from double fan structures in G at the cost of a quadratic node overhead concerning the fans.

While the base chain for G' can easily be generated using (1), there is a technical problem with the side chain generation via (2): since the number of vertices in those side chains in G' are only bounded in expectation, their actual numbers will vary and therefore during the second shrinking phase some clusters would reach the main chain earlier than others. As a consequence, the negative impact on the approximation error caused by those clusters reaching the main chain late would be lost and the overall theoretical analysis will be significantly hampered. We deal with this problem by reducing the number of side chains so that each side chain has a certain buffer area on the main chain and therefore with high probability side chains do not interfere with each other, even if they feature different number of vertices. By Chernoff bounds, for side chains with $E[x_2] = k^{2/3-\epsilon}$ in G', buffer areas of $\Theta(k^{1/3})$ between two consecutive side chains suffice with high probability. Filling in the details, it turns out that the expected diameter of the resulting graph G'' after the second shrinking phase exceeds the diameter of the input graph G by a factor of $\Omega(k^{4/3-\epsilon})$.

We randomize the layout of the synthetic graphs on the disk to ensure that the disk layout does not reveal any additional information that is exploitable. However, we use the ordering provided with sk-2005 graph for fair comparison with results reported in the literature.

4.3 Results

In this section, we first demonstrate that our bad graph class does elicit poor approximation results from the basic extension. Then, we show that a combination of techniques mentioned in Section 3 improves upon the approximation quality significantly, even for the bad graph class. We analyze the reasons for this improvement, and finally show that altogether our external memory code results in a good approximation on a wide range of graph classes in a few hours, even on graphs with billions of vertices and many billions of edges.

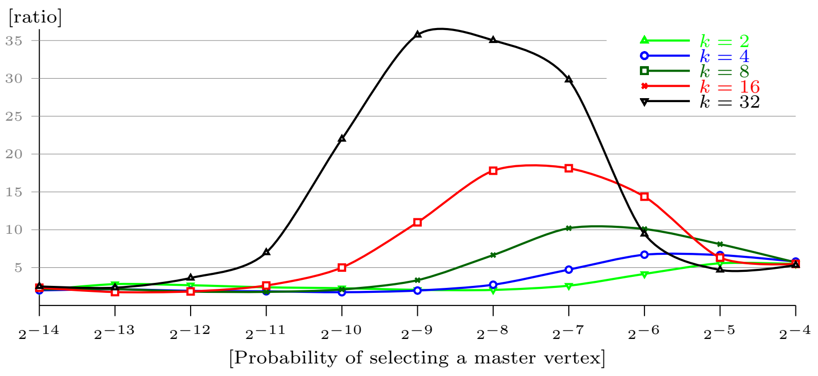


Fig. 2: Approximation ratio using the basic extension with varying probability of choosing the master vertex on five worse 2step graph instances generated with different parameters depending on k.

Empirical analysis of worse case graphs. We first ran the basic extension of the parameterized approach on worse 2step graph instances that were generated using parameter settings suitable for different values of k. As can be seen in Figure 2, for larger values of k and within a certain range, smaller master probabilities cause higher approximation ratios and as k grows the highest achievable ratios start growing even faster. However, due to the constants hidden in the construction the maxima appear for somewhat lower master probabilities than k suggests. Nevertheless, already for rather small values of k, we experience significant ratios (larger than k) demonstrating that this graph class can indeed elicit poor approximation ratio from the basic extension.

	Exact	Basic	Adaptive	Tie break	Move	All
sk-2005	40	185	196	173	182	168
ratio		4.625	4.900	4.325	4.550	4.200
$\sqrt{n}$-level	16385	16594	16416	16604	16597	16408
ratio		1.013	1.002	1.013	1.013	1.001
$\Theta(n)$-level	67108864	67212222	67131347	67212123	67174264	67131036
ratio		1.002	1.000	1.002	1.001	1.000
worse 2step	3867	138893	38643	137087	17321	13613
ratio		35.918	9.993	35.450	4.479	3.520

Table 1: Diameters on different graph classes with various extensions.

Approximation quality. Next, we consider the various techniques described in Section 3 to determine if they can improve the approximation ratio on different graph classes. These techniques include (i) *Adaptive*, where we use the probability distribution function described in Section 3; (ii) *Tie break*, where the ties in the parallel cluster growing rounds are broken such that among all clusters competing to get a vertex v, the cluster that has the shortest distance between its master and v gets the vertex v; (iii) *Move*, where the masters are re-selected after the clustering to reduce the weighted diameter of the clusters and (iv) *All*, where all of the above techniques are combined. Table 1 presents the diameter computed by these techniques together with the exact diameter for different graph classes. This table is computed for graphs with 2^{28} vertices where in both recursive levels master vertices are chosen with a probability of 2^{-8}. Thus, the various extensions contract the number of vertices in the input graph by a factor of around $2^{16} = 65,536$, making it possible to handle graphs that are significantly bigger than the main memory size. Despite such a large reduction in the number of vertices, we get a fairly small approximation ratio with the All approach in our experiments, thus proving the efficacy of our techniques.

For graphs with small diameter such as sk-2005, various additive errors dominate the approximation ratio. However, this can be easily rectified as once it is determined that the input graph has a small diameter, one can run BFS based heuristics to compute better approximations. For such graphs, BFS can be efficiently computed in external memory. In contrast, the hardest case for the external memory BFS (in terms of running time) is the $\sqrt{n}$-level graph. For this hard graph class, however, even the basic extension yields an approximation ratio of 1.013 and the All approach improves it to 1.001. The case for $\Theta(n)$-level graph is similar – the Basic approach already gives a fairly good approximation ratio and the All approach improves it further. For the interesting case of the worse 2step graph where the basic variant gives a very poor approximation ratio of 35.92, the All approach manages to improve it to 3.52 – *a factor of more than 10*. These results imply that the additional techniques provide considerable robustness to the approximation quality by significantly improving the ratio for the very bad case. Most importantly, this improvement comes at little or no additional cost to the I/O complexity and the overall runtime of the external memory implementation.

Distance distribution within clusters. Next, we analyze the reasons for the poor performance of basic extension on the worse 2step graph. After the first level of clustering, the resultant graph has a diameter of 10,067 with an approximation ratio of 2.6. It is in the (second level) clustering of this (weighted) contracted graph that the quality of the approximation deteriorates. Our techniques such as careful selection of master vertices closer to high weight edges, breaking the ties in favor of clusters with shorter distance to their masters, and moving the masters to weighted cluster centers reduce the weighted diameter of the second level clusters. The smaller diameter clusters, in turn, ensure that the resultant condensed graph better captures the diameter of the input graph.

The fact that vertices in the second level clusters are closer to their master vertices in the All scheme than in the basic extension is evident from Figure 3, where we plot the number of vertices at varying distance from their cluster masters. The number of vertices with distance at most 10 from the master vertex is 3638 for the basic approach and 5362 for the All approach, while the number of vertices with distance greater than 300 is 210 for the basic approach and 12 for the All approach.

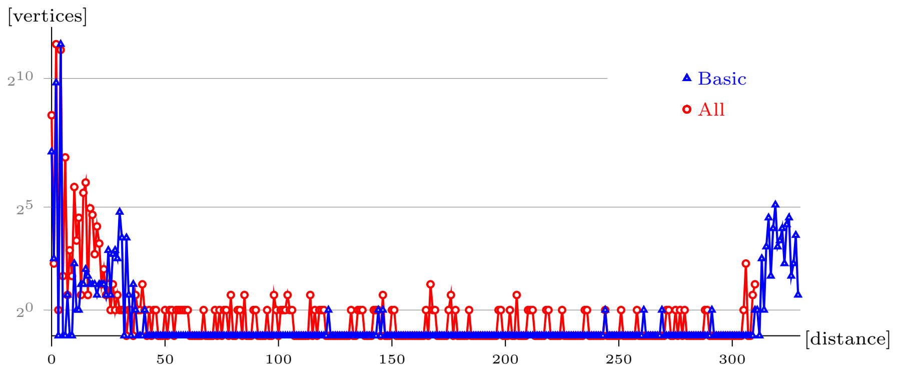


Fig. 3: Distance distribution on worst 2step graph after the second recursive level of the basic approach and all extensions

Running time and I/O complexity. Theoretically, our approach requires $O(n \cdot \sqrt{\log (k_1 \cdot k_2)/(k_1 \cdot k_2 \cdot B)} + (k_1 + k_2) \cdot scan(n) + sort(n))$ I/Os for contracting

	Size [GB]	Exact Diameter	Computed Diameter	Approx Ratio	Running Time [h]	I/O Time [h]	I/O Volume [TB]
sk-2005	27.0	40	133	3.325	0.7	0.1	0.3
$\sqrt{n}$-level	128.0	46342	46380	1.001	11.1	5.7	4.7
$\Theta(n)$-level	83.8	536870913	546560415	1.018	5.4	3.5	3.0
worse 2step	31.9	8111	25271	3.116	2.1	1.4	1.1

Table 2: Results of our external memory implementation.

the graph to n/k_1 nodes in the first iteration and then to $n/(k_1 \cdot k_2)$ in the second iteration. This is better than the BFS complexity when $k_1 + k_2 < \sqrt{B}$.

Empirically, Table 2 presents the result of using our external memory implementation to approximate the diameter on $\sqrt{n}$-level, $\Theta(n)$-level and worse 2step graphs with 2^{31} vertices and 2^{33} edges (except worse 2step which is a tree). On moderate to large diameter graphs such as these, the external memory BFS implementation based on parallel cluster growing requires *months* and even the BFS implementation based on chopping the Euler tour of a bidirectional spanning tree requires many hours [2] on graphs that are 8 times smaller than ours. Thus, the various diameter approximation techniques based on multiple calls to BFS routine are likely to take many *days*. On the other hand, our approach provides fairly good approximations of graph diameter in a few hours, that is *even less time than that required for one exact BFS call*. Alternatively, applying the single step approach [3] hardly yields better approximations in practice than the hierarchical method but results in significantly larger running times: on the $\Theta(n)$-level graph, e.g., 76 hours were needed to approximate the diameter with 2^{22} master vertices and still 21 hours with 2^{24} masters. Note that the running time of our external memory approach is still dominated by the time required for I/Os and the total volume of data moved between the two memory levels for approximating the diameter is still in the order of terabytes, thereby showing that the problem continues to remain I/O-bound, even with four parallel disks.

5 Conclusion

Using our new hierarchical extensions of the parameterized diameter approximation approach we managed to process significantly larger external-memory graphs than before while keeping approximation ratio and computation time reasonable. Open problems concern directed and dynamic versions, and the question whether the *worse 2step* graph class construction yields an asymptotically tight lower bound on the approximation error for the two level basic extension.

Our framework can as well be initialized directly from a weighted graph, but our analysis of approximation quality holds only when the input graph is unweighted. To make our hierarchical extension cache-oblivious, one might consider recursively contracting the number of vertices by a factor independent of B, till the graph reduces to a constant size.

Acknowledgements

The authors would like to thank Andreas Beckmann for his help with STXXL and the hardware resources and the anonymous reviewers for their feedback.

References

1. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM, 31(9)*, pages 1116–1127, 1988.
2. D. Ajwani. *Traversing large graphs in realistic setting.* PhD thesis, Saarland University, 2008.
3. D. Ajwani, A. Beckmann, U. Meyer, and D. Veith. I/O-efficient approximation of graph diameters by parallel cluster growing – A first experimental study. In *10th Workshop on Parallel Systems and Algorithms (PASA)*, 2012.
4. D. Ajwani, U. Meyer, and V. Osipov. Improved external memory BFS implementation. In *Proc. 9th ALENEX*, pages 3–12, 2007.
5. L. Arge, U. Meyer, and L. Toma. External memory algorithms for diameter and all-pairs shortest-paths on sparse graphs. In *Proc. 31st ICALP*, volume 3142 of *LNCS*, pages 146–157. Springer, 2004.
6. D. A. Bader and K. Madduri. Snap, small-world network analysis and partitioning: An open-source parallel graph framework for the exploration of large-scale networks. In *Proc. 22nd IPDPS*, pages 1–12. IEEE, 2008.
7. K. Boitmanis, K. Freivalds, P. Ledins, and R. Opmanis. Fast and simple approximation of the diameter and radius of a graph. In *Proc. 5th WEA*, volume 4007 of *LNCS*, pages 98–108. Springer, 2006.
8. U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *LNCS*. Springer, 2005.
9. R. Chowdury and V. Ramachandran. External-memory exact and approximate all-pairs shortest-paths in undirected graphs. In *Proc. 16th SODA*, pages 735–744. ACM-SIAM, 2005.
10. D. G. Corneil, F. F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. *Discrete Applied Mathematics*, 113(2-3):143–166, 2001.
11. P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, and A. Marino. Finding the diameter in real-world graphs – experimentally turning a lower bound into an upper bound. In *Proc. 18th ESA*, volume 6346 of *LNCS*, pages 302–313. Springer, 2010.
12. P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino. On computing the diameter of real-world directed (weighted) graphs. In *Proc. 11th SEA*, 2012.
13. R. Dementiev and P. Sanders. Asynchronous parallel disk sorting. In *Proc. 15th SPAA*, pages 138–148. ACM, 2003.
14. M. T. Goodrich and P. Pszona. External memory network analysis algorithms for naturally sparse graphs. In *Proc. 19th ESA*, pages 664–676, 2011.
15. C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics*, 13:1.10–1.9, 2009.
16. K. Mehlhorn and U. Meyer. External-memory Breadth-First Search with sublinear I/O. In *Proc. 10th ESA*, volume 2461 of *LNCS*, pages 723–735. Springer, 2002.
17. U. Meyer. On trade-offs in external-memory diameter-approximation. In *Proc. 11th SWAT*, pages 426–436, 2008.