Fast Approximate Distance Queries in Unweighted Graphs using Bounded Asynchrony

Adam Fidel, Francisco Coral Sabido, Colton Riedel,
Nancy M. Amato, and Lawrence Rauchwerger

Parasol Lab, Dept. of Computer Science and Engineering, Texas A&M University

Abstract. We introduce a new parallel algorithm for approximate breadth-first ordering of an unweighted graph by using bounded asynchrony to parametrically control both the performance and error of the algorithm. This work is based on the k-level asynchronous (KLA) paradigm that trades expensive global synchronizations in the level-synchronous model for local synchronizations in the asynchronous model, which may result in redundant work. Instead of correcting errors introduced by asynchrony and redoing work as in KLA, in this work we control the amount of work that is redone and thus the amount of error allowed, leading to higher performance at the expense of a loss of precision. Results of an implementation of this algorithm are presented on up to 32,768 cores, showing 2.27x improvement over the exact KLA algorithm with minimal error on several graph inputs.

Keywords: parallel graph algorithms, breadth-first search, distance query, approximate algorithms, asynchronous, distributed memory

1 Introduction

Processing large-scale graphs has increasingly become a critical component in a variety of fields, from scientific computing to social analytics. Due to the ever growing size of graphs of interest, distributed and parallel algorithms are typically employed to process graphs on a large scale.

Computing shortest paths in networks is a fundamental operation that is useful for multiple reasons and many graph algorithms are built on top of shortest paths. For example, computing centrality metrics and network diameter relies on distance queries. In addition to being a building block for other algorithms, shortest path queries can be used on their own to determine connectivity and distances between particular vertices of interest. For many large real-world graphs, computing exact shortest paths is prohibitively expensive and recent work [30, 27, 22, 28] explores efficient approximate algorithm for this problem. In unweighted graphs, an online distance query can be answered through the use of breadth-first search (BFS).

In this work, we introduce a novel approximate parallel breadth-first search algorithm based on the k-level asynchronous [15] (KLA) paradigm. The KLA paradigm is a generalization of level-synchronous processing [20] (based on the

bulk-synchronous parallel model [34]) and the asynchronous model [26], allowing for parametric control of the amount of asynchrony from full (asynchronous) to none (level-synchronous). In a level-synchronous execution of breadth-first search, converged values are correct at the end of a level, at the cost of expensive global synchronizations. On the other hand, a high amount of asynchrony in breadth-first search may lead to redundant work, as the lack of a global ordering could cause a vertex to receive many updates with smaller distances until the true breadth-first distance is discovered. Each update to the vertex's state will trigger a propagation of its new distance to its neighbors, potentially leading to all reachable vertices being reprocessed many times and negating the benefit of asynchronous processing.

Our novel algorithm alleviates the amount of redundant work performed by controlling how updates trigger propagation and allowing for vertices to contain some amount of error. By not sending the improved value to a vertex's neighbors, we limit the amount of redundant work that occurs during execution. We modify the KLA breadth-first search algorithm by conditionally propagating improved values received from a neighbor update.

The contributions of this work include:

- **Approximate k-level asynchronous breadth-first search algorithm.** We present a new algorithm for approximate breadth-first search that trades accuracy for performance in a KLA BFS. We find and prove an upper bound on the error as a function of degree of approximation.
- **Implementation that achieves scalable performance.** Our implementation in the STAPL Graph Library shows an improvement of up to 2.27x over the exact KLA algorithm with minimal error. Results show that our technique is able to scale up to 32,768 cores.

2 Approximate Breadth-First Search

Our algorithm is implemented in the k-level asynchronous paradigm. In KLA, algorithms are expressed using two operators. The vertex operator is a fine-grained function executed on a single vertex that updates the vertex's state and issues visitations to neighboring vertices. It may spawn visitations through the use of `Visit(u, op)` or `VisitAllNeighbors(v, op)`, where `u` is the ID of a single neighbor and `v` is the ID of the vertex being processed. These visitations are encapsulated in the neighbor operator, which updates a vertex based on values received from a single neighbor.

In the exact KLA breadth-first search, skipping the application of the neighbor operator could lead to an incorrect result, but reduces the performance overhead of redundant work that is often seen in highly asynchronous algorithms. We show that the amount of error can be bounded, while improving the performance of the distance query.

2.1 Algorithmic Description

In this section, we show how to express approximate breadth-first search using the KLA paradigm. The goal is to compute, for each vertex, the distance from the vertex and the root in the breadth-first search tree. We denote this distance as $d(v)$.

```
Function VertexOperator(v)
    if v.color = GREY then
        v.color = BLACK
        VisitAllNeighbors(v, NeighborOp, v.dist+1, v.id)
        return true
    else
        return false
    end
```

Algorithm 1: k-level asynchronous BFS vertex operator.

Initially, all vertices except the source have distance $d_k(v) = \infty$, no parent, and color set to black. The source vertex sets its distance to 0, itself as its parent and marks itself active by setting its color to grey. Algorithm 1 shows the vertex operator that is executed on all vertices in parallel. Each vertex determines if it is active by checking if its color is set to grey. If so, it issues visitations to all of its neighbors, sending its distance plus one. The traversal is completed if all invocations of the vertex operator return false in a superstep (i.e., none of the vertices are active).

Algorithm 2 presents the neighbor operator for the exact breadth-first search algorithm. The distance and parent are updated if the incoming distance is less than the vertex's current distance. In addition, the vertex sets its color to grey, marking it as active, and returns a flag indicating that it should be revisited. In the k-level async model, if the invocation of the neighbor operator returns true, the vertex operator will be reinvoked on that vertex only if its hop-count is still in bounds of the KLA superstep. That is, if $d(v) \bmod k = 0$, then the visitation is at the edge of the superstep and thus the vertex operator will not be invoked until the start of the next superstep.

In this work, we introduce a new neighbor operator in Algorithm 3 that allows for the correction of an error and repropagation of the corrected distance under certain conditions. We use tolerance $0 \leq \tau < 1$ to denote the amount of error a vertex will allow until it propagates a smaller distance. For a visit with current distance d and better distance d_{new}, we will propagate the new distance if $(d - d_{new})/d \geq \tau$. We now need to store two distances: one that represents the current smallest distance seen and the distance of the last propagation. The last propagated distance is required as a vertex may continually improve its own distance, but it will only repropagate if a neighbor visitation contains a distance that is τ-better than its last propagated distance. By following a vertex's parent property, the algorithm also provides a path from the every reachable vertex to

```
Function NeighborOperator(u, dist, parent)
    if u.dist > dist then
        u.dist ← dist
        u.parent ← parent
        u.color ← GREY
        return true
    else
        return false
    end
```

Algorithm 2: Original k-level asynchronous BFS neighbor operator.

```
Function ApproximateNeighborOperatorTolerance(u, dist, parent)
    if u.dist > dist then
        u.dist = dist
        first_time ← u.parent = none
        better ← (u.prop - dist)/u.prop ≥ τ
        if first_time ∨ better then
            u.parent ← parent
            u.prop ← dist
            u.color ← GREY
            return true
        end
    else
        return false
    end
```

Algorithm 3: Approximate k-level asynchronous BFS with tolerance neighbor operator.

the source, similar to the traditional version of breadth-first search. However, these vertices may report a larger distance than the length of the discovered path, due to updates that were not propagated.

The parameter τ controls the amount of tolerated error. Note that if $\tau < 1/|V|$, then there is no error in the result and the neighbor operator is equivalent to the exact version in Algorithm 2.

2.2 Error Bounds

As the approximate breadth-first search may introduce error, we quantify the error that may be caused due to asynchronous visitations. We denote the breadth-first distance of a vertex v at the end of a KLA traversal using $d_k(v)$, where k is the level of asynchrony. Similarly, $d_0(v)$ is the true breadth-first distance for vertex v. In this section, we will show that the error of the breadth-first distance is bounded by $d_k(v) \leq d_0(v)k$.

Lemma 1. *At the end of the first KLA superstep, all reached vertices have distance* $d_k(v) \leq k$.

Proof. Assume at the end of the first superstep, there exists a vertex v with distance $d_k(v) > k$. This means that v was reached on a path from the source that has $h > k$ hops. This is not possible, as the traversal will not allow a visitation that is more than k hops away. Therefore $d_k(v) \leq k$. □

Theorem 1. *At the end of the algorithm, all reachable vertices will have distance $d_k(v) \leq ks_v$, where s_v is the superstep in which v was discovered.*

Proof. Assume that after superstep s all reached vertices will have distance $d_k(v) \leq sk$. Lemma 1 shows this holds for $s = 1$. All active vertices will issue visitations to their neighbors, traveling up to at most k hops in superstep $s + 1$. Consider a previously unreached vertex u that will be discovered in superstep $s+1$ from some vertex w that was discovered in superstep s. Vertex w was on the boundary of superstep s and has distance at most sk from the source. Therefore, $d_k(u) \leq d_k(w) + k$ because u will be discovered from a path that is up to k hops from w.

$$\begin{aligned} d_k(u) &\leq d_k(w) + k \\ &\leq sk + k && \text{(inductive hypothesis)} \\ &\leq (s+1)k && \text{(simplification)} \end{aligned}$$

Through induction, $d_k(u) \leq sk$ for a vertex u discovered in superstep s. □

Lemma 2. *If there exists a path π from the source to a vertex v, then v must be discovered no later than superstep $|\pi|$.*

Proof. We will show the lemma holds by induction. If the length of path π is 1, vertex v shares an edge with the source. Then in the first superstep, the source will visit all edges and discover v.

Suppose the lemma holds for any path with length i. Let π be a path with length $|\pi| = i+1$. Then the i^{th} vertex along the path, v_i, will have been discovered in or before the i^{th} superstep. Now, by Algorithm 1, the vertex v_i will traverse all of its outgoing edges in or before the $(i+1)^{th}$ superstep and discover the $(i+1)^{th}$ vertex along the path π. This proves the lemma holds for any path π of length $i + 1$. Therefore, the lemma holds for any path π by induction.

Lemma 3. *If there exists a path from the source to a vertex v, then v will be discovered at the latest in superstep $d_0(v)$.*

Proof. If a vertex has distance $d_0(v)$, then the shortest path π^* to v has length $|\pi^*| = d_0(v)$. By Lemma 2, this path must be discovered at the latest in superstep $d_0(v)$. □

Theorem 2. *At the end of the algorithm, all reachable vertices will have distance $d_k(v) \leq d_0(v)k$.*

Proof. By Theorem 1, $d_k(v) \leq s_v k$. We know through Lemma 3 that v will be visited by superstep $d_0(v)k$. Combining these, the approximation of the true breadth-first distance is off by at most a multiplicative factor of k: $d_0(v)k$. □

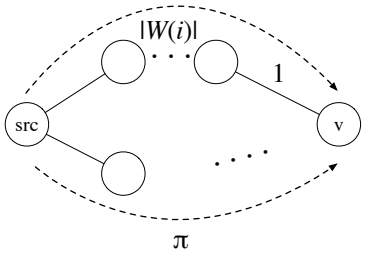


Fig. 1. Example graph showing two different paths from the source to a vertex v.

2.3 Bounds with Tolerance

When using the tolerance heuristic, a vertex with distance d will only propagate a new distance d_{new} if the following is true:

$$\frac{d - d_{new}}{d} \geq \tau \tag{1}$$

In the exact k-level asynchronous algorithm, all vertices that are distance $d_0(v)$ away from the source will be visited in superstep $\frac{d_0(v)}{k}$. However, since we allow some bounded error, it is possible for a vertex to be visited in the $\frac{d_k(v)}{k}$ superstep, which may be later than its original visitation. In addition, all edges that are traversed through visitations will be visited in the same superstep in which the visit was issued. However, not all visitations trigger a propagation of a better distance to the vertex's neighbors.

We will denote the discovered distance of a vertex using the tolerance heuristic as $d^\tau(v)$. In this section, we will prove that by using this heuristic, if a vertex v is reached at the end of the first superstep, then $d^\tau(v) \leq \frac{\sum_{j=0}^{d_0(v)-1}(1-\tau)^j}{(1-\tau)^{d_0(v)}}$.

Lemma 4. *All vertices with a true distance of 1 will propagate a distance that is at most* $\frac{1}{1-\tau}$.

Proof. Because the distance from the source to v is 1, the shortest path $\pi^* =< (src, v) >$ will be processed eventually in the traversal. Consider that vertex v is discovered along a path π from the source and marks itself as distance $|\pi|$. Once the path π^* is processed, v will not propagate its distance if $\frac{|\pi|-1}{|\pi|} < \tau$. Simplifying, the length of the path is $|\pi| < \frac{1}{1-\tau}$. Therefore, v will propagate a distance that is at most $\frac{1}{1-\tau}$, otherwise a repropagation will be triggered.

□

Theorem 3. *At the end of the first superstep, all reachable vertices will propagate a distance at most* $\frac{\sum_{j=0}^{d_0(v)-1}(1-\tau)^j}{(1-\tau)^{d_0(v)}}$.

Proof. Let $W(i) = \frac{\sum_{j=0}^{i-1}(1-\tau)^j}{(1-\tau)^i}$ denote the length of the longest path that will be tolerated by a vertex of true distance i without triggering a propagation. Lemma 4 shows that this holds for vertices with true distance 1. Assume that this property holds for vertices of distance i.

Let v be a vertex with true distance $i+1$ discovered along some path π. By definition, v will not repropagate upon seeing a path π_{new} if the following holds:

$$\frac{|\pi| - |\pi_{new}|}{|\pi|} < \tau \tag{2}$$

The shortest path π_{new} that could be discovered without repropagating could have length $|\pi_{new}| = W(i)+1$. Any path longer than π_{new} would have triggered a repropagation along the path, by definition of $W(i)$. See Figure 1 for an example.

The vertex will not propagate the better distance if the threshold is not met:

$$\frac{|\pi| - (\frac{\sum_{j=0}^{i-1}(1-\tau)^j}{(1-\tau)^i} + 1)}{|\pi|} < \tau \tag{3}$$

Written in terms of $|\pi|$, this can be simplified:

$$\begin{aligned}
|\pi| &< \frac{\frac{\sum_{j=0}^{i-1}(1-\tau)^j}{(1-\tau)^i} + 1}{1-\tau} \\
&= \frac{\sum_{j=0}^{i-1}(1-\tau)^j}{(1-\tau)^{i+1}} + \frac{(1-\tau)^i}{(1-\tau)^{i+1}} \\
&= \frac{\sum_{j=0}^{i}(1-\tau)^j}{(1-\tau)^{i+1}} \\
&= W(i+1) && \text{(definition of } W(i)\text{)}
\end{aligned}$$

The bound therefore holds for vertices with true distance $i+1$ and thus all vertices by induction. □

As shown in Algorithm 3, a vertex always updates its distance upon seeing a better distance, without necessarily propagating it. This means that a vertex's discovered distance is at most its propagated distance. That is, all vertices discovered in the first superstep will have distance at most $d^\tau(v) \leq \frac{\sum_{j=0}^{d_0(v)-1}(1-\tau)^j}{(1-\tau)^{d_0(v)}}$.

Note that in the case of $\tau = 0$, $d^\tau(v) = \sum_{j=0}^{d_0(v)-1} 1/1 = d_0(v)$. Therefore, $\tau = 0$ is equivalent to the exact algorithm.

2.4 Combined Bounds

By the definition of KLA, the maximum distance that any vertex can be assigned in the first superstep is k. Therefore, for a vertex of true distance i, its discovered

distance can be at most k. $W(i)$ is the length of the longest path that can be tolerated by a vertex of true distance i without propagation. However, if this path is longer than k, then it will not be visited and thus the worst case distance will be less than $W(i)$. Now, solving $W(i) = k$ for i only considering $\tau > 0$ because, as shown above, there is no error for $\tau = 0$, we find:

$$\begin{aligned}
k = W(i) &= \frac{\sum_{j=0}^{i-1}(1-\tau)^j}{(1-\tau)^i} \\
&= \frac{\frac{1-(1-\tau)^i}{1-(1-\tau)}}{(1-\tau)^i} && \text{(Partial geometric sum, where } 1-\tau > 0\text{)} \\
k\tau &= \frac{1-(1-\tau)^i}{(1-\tau)^i} \\
k\tau + 1 &= \frac{1}{(1-\tau)^i} \\
i &= \log(\frac{1}{k\tau+1}) / \log(1-\tau)
\end{aligned}$$

If a vertex v has at most true distance i, then its discovered distance is bounded by $W(i)$. However, if the true distance is greater than $\log(\frac{1}{k\tau+1}) / \log(1-\tau)$, then the vertex's discovered distance can be no more than k, because the path that causes the bound of $W(i)$ is no longer reachable in k hops.

Therefore, if a vertex v is reached in the first superstep, the maximum distance $d_k^\tau(v)$ that v can have is:

$$d_k^\tau(v) \leq \begin{cases} d_0(v) & \tau = 0 \\ \frac{\sum_{j=0}^{d_0(v)-1}(1-\tau)^j}{(1-\tau)^{d_0(v)}} & d_0(v) \leq \log(\frac{1}{k\tau+1}) / \log(1-\tau) \\ k & otherwise \end{cases}$$

Figure 2 presents the trend of this function for various values of τ and a fixed value $k = 16$. We see that $W(i)$ can grow very rapidly, but is bounded by at most k. For $\tau = 0$, the approximated distance is the same as the exact distance.

Using the same technique as Theorem 2, we can show that error will accumulate across supersteps in an additive way. Therefore, the total distance that a vertex at the end of the algorithm will have is $d_k^\tau(v) \leq d_0(v)k$.

3 Implementation

We implemented the approximate breadth-first traversal in the STAPL Graph Library (SGL) [14, 16, 15]. SGL is a generic parallel graph library that provides a high-level framework that abstracts the details of the underlying distributed environment. It consists of a parallel graph container (`pGraph`), a collection of

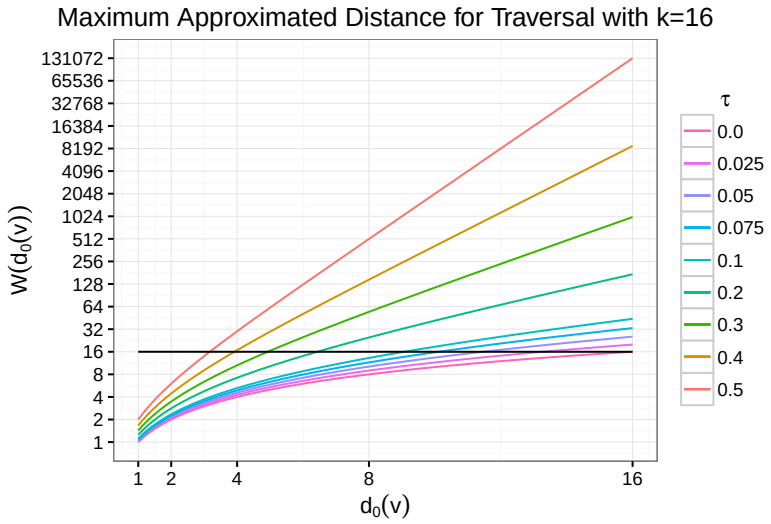


Fig. 2. Computed distance $d_k^\tau(v)$ vs actual distance $d_0(v)$ for multiple τ and fixed k.

parallel graph algorithms, and a graph paradigm that supports level-synchronous and asynchronous execution of algorithms.

The `pGraph` container is a distributed data store built using the `pContainer` framework (PCF) [31] provided by the Standard Template Adaptive Parallel Library (STAPL) [10]. It provides a shared-object view of graph elements across a distributed-memory machine. The STAPL Runtime System (STAPL-RTS) and its communication library ARMI (Adaptive Remote Method Invocation) use the remote method invocation (RMI) abstraction to allow asynchronous communication on shared objects while hiding the underlying communication layer (e.g MPI, OpenMP).

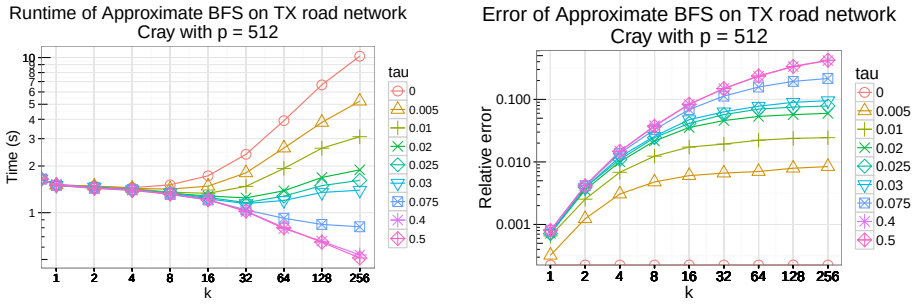
4 Experimental Evaluation

We evaluated our technique on two different systems.

Cray-XK7. This is a Cray XK7m-200 system which consists of twenty-four compute nodes with AMD Opteron 6272 Interlagos 16-core processors at 2.1 GHz. Twelve of the nodes are single socket with 32 GB of memory, and the remaining twelve are dual socket nodes with 64 GB of memory.

IBM-BG/Q. This is an IBM BG/Q system available at Lawrence Livermore National Laboratory. IBM-BG/Q has 24,576 nodes, each node with a 16-core IBM PowerPC A2 processor clocked at 1.6 GHz and 16 GB of memory. The compiler used was `gcc` 4.8.4.

The code was compiled with maximum optimization levels (`-DNDEBUG -O3`). Each experiment has been repeated 32 times and we present the mean execution time along with a 95% confidence interval using the t-distribution. We



(a) (b)

Fig. 3. Approximate BFS with tolerance heuristic on TX road network with 512 cores on Cray evaluating (a) runtime and (b) error.

also measure the relative error of a vertex's distance, where error is defined as $(d_k^\tau(v) - d_0(v))/d_0(v)$. We show the mean relative error across all vertices.

4.1 Breadth-First Search

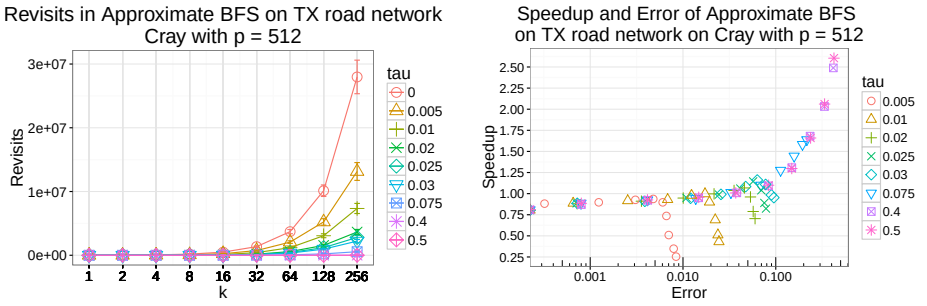
In this section, we evaluate our algorithm on various graphs in terms of execution time and relative error.

In Figure 3, we evaluate both the execution time and error on the Texas road network from the SNAP [2] collection on 512 cores on the Cray-XK7 platform. This graph has 1.3 million vertices and 1.9 million edges. As expected, a lower value of τ results in slower execution time as more repropagations occur with lower tolerance. In the extreme case of $\tau = 0$, every message that contains a better distance is propagated and thus it is the same as the exact version of the algorithm. Figure 4(a) shows the number of repropagations that occur as we vary the level of asynchrony and τ. As expected, higher values of k result in many more visitations, while higher τ triggers relatively less visitations. This behavior results in the corresponding time and error tradeoffs we observe in Figure 3.

Figure 4(b) shows speedup vs error on the Texas road network. Speedup is defined as the ratio of the exact algorithm's execution time with the fastest k and the approximate algorithm's execution time. If an application is willing to tolerate error in the result, we see that we are able to achieve 2.6x speedup for an execution with 42% error.

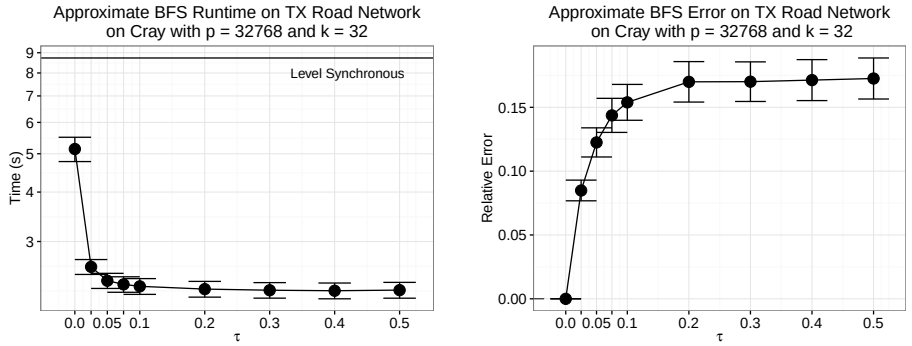
Figure 5 shows that we see similar benefit using the road network graph on the IBM-BG/Q platform for a fixed value of k. We see that the exact version of the KLA breadth-first search ($\tau = 0$) is slower than the level synchronous version, and the approximate version is faster than both. At 32,768 cores, the approximate version is 2.27x faster with around 17% mean error.

Random Neighborhood. We next evaluate the algorithm on a deformable graph that allows us to vary the diameter from very large (circular chain) to very small (random graph). This results in graphs with different diameters by allowing any given vertex to randomly select and connect only to its $\pm m$-closest



(a) (b)

Fig. 4. Approximate BFS with tolerance heuristic on TX road network with 512 cores on Cray-XK7 evaluating (a) number of repropagations that occur during traversal and (b) speedup over the fastest k.

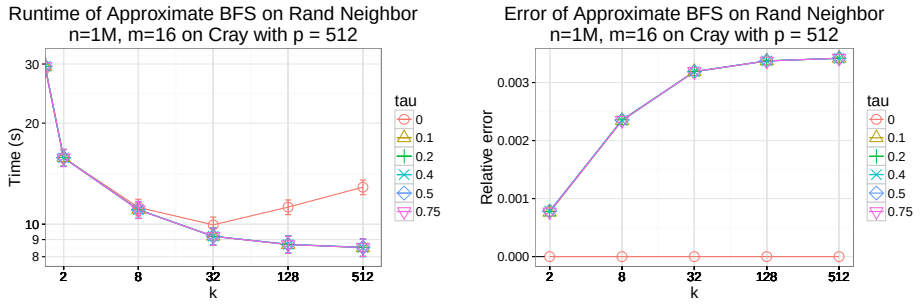


(a) (b)

Fig. 5. Strong scaling of approximate BFS on IBM-BG/Q platform evaluating sensitivity of (a) runtime and (b) error.

neighboring vertices. This is similar to the approach described by Watts and Strogatz [35] where the rewiring mechanism is limited in terms of distance.

Figure 6 shows the performance and error of an execution of this algorithm on a random neighborhood graph on 512 cores on the Cray-XK7 platform. As shown, we see a benefit for using the approximate version for higher values of k. At a k of 512, the approximate algorithm has a 1.12x speedup over the fastest exact version but only has an error of 0.3%. Because this graph does not have as much opportunity for wasted work as the road network, the benefits of approximation are not as pronounced, but we still see an improvement in performance with negligible error.



(a) (b)

Fig. 6. Approximate BFS with tolerance heuristic on random neighborhood network ($n = 1,000,000$ and $m = 16$) with 512 cores on Cray-XK7.

5 Related Work

Graph Processing and Breadth-First Search. The vertex-centric programming model, popularized by Pregel [20] and its open-source equivalent Giraph [3], has become a standard in parallel graph processing. The so-called *think like a vertex* paradigm allows algorithm writers to express their computation in terms of vertex programs, which describe the operations to be executed on a single vertex and its incident edges. Whereas Pregel's model is push-based, GraphLab [19] offers a pull-based model based on the three operators gather-apply-scatter.

Many general purpose frameworks and runtimes [23, 21, 12] for graph processing have been proposed and are used in practice. Galois is an amorphous data parallel processing framework with support for many vertex-centric paradigms [24]. Grappa [23] is a distributed shared memory framework designed specifically for data-intensive applications. Graph-based domain-specific libraries [17] exist and have been shown to perform well in practice.

Many techniques have been proposed specifically to improve breadth-first search. Most notably, the Graph 500 benchmark [1] has sparked much research into improving [8, 9] breadth-first search on scale-free networks for distributed-memory architectures. A hybrid top-down bottom-up breadth-first search was presented in [6] that shows large improvement on scale-free networks.

Approximation. Decades of research exist for efficiently approximating graph features, including diameter [11], neighborhoods [25] and triangles [7]. In this work, we focus on single-source distance queries for unweighted graphs.

In [29], the authors propose automatic synthesis of approximate graph programs through several auto-approximation techniques. Our work is similar to the task skipping approach where inputs from neighbors are ignored under certain conditions. However, the authors primarily focus on single-core processing while we consider distributed-memory parallel algorithms.

There has been a large body of work to approximate the all-pairs shortest path problem for weighted graphs through the use of distance oracles [13, 32, 4] ([30] provides a comprehensive survey). An $O(\min(n^2, kmn^{1/k}))$-time algo-

rithm for computing a $2k-1$ approximation has been presented in [5]. In [27], a distributed-memory algorithm using local betweenness is presented. We focus our work on online queries of unweighted shortest paths from a single source.

Ullman and Yannakakis [33] show a high-probability PRAM algorithm for approximating a breadth-first search tree by performing multiple traversals from landmark vertices. This was extended to weighted graphs [18] on a concurrent-write PRAM using a hop-limited traversal, similar to the k-level async model. A recent work [22] introduces a $(1 + o(1))$-approximation for weighted graphs using multiple rounds of an exact BFS.

To the best of our knowledge, our approach is the first to incorporate asynchrony into the approximation and leverage the benefit of asynchronous processing for performance.

6 Conclusion

In this paper, we presented a novel parallel algorithm for approximating breadth-first distances in a graph. We provide bounds for the error of such an approach and show that experimentally, the observed errors are much lower than the theoretical bounds. Our implementation shows substantial benefit in some cases with only minor losses in precision of the exact answer.

References

1. The graph 500 list. `http://www.graph500.org`, 2011.
2. Stanford large network dataset collection. `http://snap.stanford.edu/data/index.html`, 2013.
3. C. Avery. Giraph: Large-scale graph processing infrastructure on hadoop. Hadoop Summit, 2011.
4. S. Baswana and T. Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM J. Comput.*, 39(7):2865–2896, May 2010.
5. S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected o(n2) time. *ACM Trans. Algorithms*, 2(4):557–577, Oct. 2006.
6. S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 12:1–12:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
7. L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, Oct. 2010.
8. A. Buluç, S. Beamer, K. Madduri, K. Asanović, and D. Patterson. Distributed-memory breadth-first search on massive graphs. In D. Bader, editor, *Parallel Graph Algorithms*. CRC Press, 2015.
9. A. Buluç and K. Madduri. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 65:1–65:12, New York, NY, USA, 2011. ACM.

10. A. A. Buss, Harshvardhan, I. Papadopoulos, O. Pearce, T. G. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger. STAPL: standard template adaptive parallel library. In *Proceedings of of SYSTOR 2010: The 3rd Annual Haifa Experimental Systems Conference, Haifa, Israel, May 24-26, 2010*, pages 1–10, New York, NY, USA, 2010. ACM.
11. M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. Space and time efficient parallel graph decomposition, clustering, and diameter approximation. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 182–191, New York, NY, USA, 2015. ACM.
12. D. Gregor and A. Lumsdaine. The parallel BGL: A generic library for distributed graph computations. In *Parallel Object-Oriented Scientific Computing (POOSC)*, July 2005.
13. A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 499–508, New York, NY, USA, 2010. ACM.
14. Harshvardhan, A. Fidel, N. M. Amato, and L. Rauchwerger. The STAPL Parallel Graph Library. In *Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, pages 46–60. Springer Berlin Heidelberg, 2012.
15. Harshvardhan, A. Fidel, N. M. Amato, and L. Rauchwerger. KLA: A new algorithmic paradigm for parallel graph computations. In *Proc. Intern. Conf. Parallel Architecture and Compilation Techniques (PACT)*, PACT '14, pages 27–38, New York, NY, USA, 2014. ACM. Conference Best Paper Award.
16. Harshvardhan, A. Fidel, N. M. Amato, and L. Rauchwerger. An algorithmic approach to communication reduction in parallel graph algorithms. In *Proc. Intern. Conf. Parallel Architecture and Compilation Techniques (PACT)*, PACT '15, pages 201–212, San Francisco, CA, USA, 2015. IEEE. Finalist for Conference Best Paper Award.
17. S. Hong, H. Chafi, E. Sedlar, and K. Olukotun. Green-marl: a dsl for easy and efficient graph analysis. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 349–362, New York, NY, USA, 2012. ACM.
18. P. N. Klein and S. Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 25(2):205 – 220, 1997.
19. Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, Apr. 2012.
20. G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.
21. M. Méndez-Lojo, D. Nguyen, D. Prountzos, X. Sui, M. A. Hassaan, M. Kulkarni, M. Burtscher, and K. Pingali. Structure-driven optimizations for amorphous data-parallel programs. In *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '10, pages 3–14, New York, NY, USA, 2010. ACM.
22. D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 565–573, New York, NY, USA, 2014. ACM.

23. J. Nelson, B. Holt, B. Myers, P. Briggs, S. Kahan, L. Ceze, and M. Oskin. Grappa: A latency-tolerant runtime for large-scale irregular application. WRSC'14, April 2014.
24. D. Nguyen, A. Lenharth, and K. Pingali. A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 456–471, New York, NY, USA, 2013. ACM.
25. C. R. Palmer, P. B. Gibbons, and C. Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 81–90, New York, NY, USA, 2002. ACM.
26. R. A. Pearce, M. Gokhale, and N. M. Amato. Multithreaded asynchronous graph traversal for in-memory and semi-external memory. In *Conference on High Performance Computing Networking, Storage and Analysis, SC 2010, New Orleans, LA, USA, November 13-19, 2010*, pages 1–11, 2010.
27. Z. Qi, Y. Xiao, B. Shao, and H. Wang. Toward a distance oracle for billion-node graphs. *Proc. VLDB Endow.*, 7(1):61–72, Sept. 2013.
28. M. Qiao, H. Cheng, L. Chang, and J. X. Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):55–68, 2014.
29. Z. Shang and J. X. Yu. Auto-approximation of graph computing. *Proc. VLDB Endow.*, 7(14):1833–1844, Oct. 2014.
30. C. Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, Mar. 2014.
31. G. Tanase, A. A. Buss, A. Fidel, Harshvardhan, I. Papadopoulos, O. Pearce, T. G. Smith, N. Thomas, X. Xu, N. Mourad, J. Vu, M. Bianco, N. M. Amato, and L. Rauchwerger. The STAPL parallel container framework. In *Proceedings of the 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2011, San Antonio, TX, USA, February 12-16, 2011*, pages 235–246, 2011.
32. M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 183–192, New York, NY, USA, 2001. ACM.
33. J. Ullman and M. Yannakakis. High-probability parallel transitive closure algorithms. In *Proceedings of the Second Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '90, pages 200–209, New York, NY, USA, 1990. ACM.
34. L. Valiant. Bridging model for parallel computation. *Comm. ACM*, 33(8):103–111, 1990.
35. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. In *Nature*, pages 440–442, 1998.