Subgraph Rank: PageRank for Subgraph-Centric Distributed Graph Processing

Nitin Chandra Badam
Dept. of Computer Science and Engineering,
IIT Guwahati
chandra.nitin@iitg.ernet.in

Yogesh Simmhan
Supercomputer Education and Research Centre,
Indian Institute of Science, Bangalore
simmhan@serc.iisc.in

ABSTRACT

The growth of Big Data has seen the increasing prevalence of interconnected graph datasets that reflect the variety and complexity of emerging data sources. Recent distributed graph processing platforms offer vertex-centric and subgraph-centric abstractions to compose and execute graph analytics on commodity clusters and Clouds. Naïve translation of existing graph algorithms to these programming models can offer sub-optimal performance. We analyze the effectiveness of PageRank, a popular graph centrality measure, for a subgraph-centric programming model, and propose variations based on the existing BlockRank algorithm to improve the performance. We evaluate these algorithms on real-world graphs using the GoFFish platform on Amazon EC2 Cloud VMs, and demonstrate that the proposed Subgraph Rank algorithm outperforms the native PageRank and BlockRank algorithms, and is faster by $23 - 74\%$ for most graphs we evaluated, while achieving an equivalent PageRank quality.

1. INTRODUCTION

Data processing has seen a sea change in the recent decade. The term "Big Data" has been coined to reflect the potential of – and complexity of – managing, exploring and analyzing this massive influx, in order to offer knowledge and insights. Google's MapReduce [9] has proven seminal not just in providing a framework for processing large data volumes, but in allowing us to easily leverage distributed commodity resources and Clouds to achieve the same. As such, the characteristics of these large scale datasets have been evolving since the era of Google's massive web logs, whose text and tuple based processing motived MapReduce. We are now seeing the pervasiveness of *interconnected data* – linked data from the web [5], billions of social network users [36], online sensing networks from Internet of Things [27], genome networks [3], to name a few – that reflect the variety and complexity of emerging Big Data sources.

This need for large scale graph processing has motivated the development of *distributed graph platforms* such as Pregel [28], GraphLab [26], and Giraph++ [35]. Some of these also operate on special graph structures, such as PowerGraph [14] for power law graphs and GoFFish [34] for time-series graphs. Such graph platforms complement the extensive work on parallel graph processing [2] on high performance computing hardware by instead using commodity clusters and Clouds that are more broadly accessible. At the same time, these distributed platforms also, arguably, offer simpler programming abstractions than, say, MPI to compose graph applications and analytics. Vertex- [1, 28] and subgraph- [34, 35] centric abstractions, for example, allow users to compose the graph application from the perspective of a single vertex or subgraph and operate across iterative supersteps, much like MapReduce allows users to write their logic over individual key-value(s) pairs. The platforms handle scalable, data parallel execution of the graph applications once mapped to their programming model. Recent results have shown the *subgraph-centric programming models to out-perform the vertex-centric abstractions* [34, 35], and thus hold promise for wider adoption.

At the same time, the introduction of these novel graph programming abstractions means that existing shared memory or parallel graph algorithms may not be a direct fit on these platforms. And a naïve translation of existing algorithms to these new abstractions may offer sub-optimal performance. It is well known that algorithmic innovations at design-time, that effectively use the underlying abstractions, can significantly improve the application performance compared to relying exclusively on runtime optimizations provided by the platform. As a result, there is a need to examine where existing algorithms fit directly, need to be adapted or new algorithms are required, to make the best use of such platforms.

Graph centrality measures are a key analytic that is used in real-world networks, from understanding critical junctions in power grids [10] to the spread of ideas (or diseases) in social (or human) networks. *PageRank* [29] proposed by Google for web graphs is a special case of Eigenvalue Centrality [7], and is often used as a canonical algorithm for evaluating graph platforms. As graph structures and sizes have evolved over the past decade, research into PageRank has contributed more scalable algorithms that handle heterogeneous and distributed topologies. This paper continues in that spirit.

There has been extensive work on improving the PageRank algorithm to fit different platforms, including MapReduce [4, 18, 20]. As a result, it is useful to understand how

The 20th International Conference on Management of Data (COMAD), 17th-19th Dec 2014 at Hyderabad, India.

effectively the PageRank for a graph can be computed using such novel distributed graph processing frameworks – where existing PageRank algorithms work, and when new ones need to be developed.

To this end, we analyze how the PageRank algorithm and its parallel variants map to a subgraph-centric programming model, and their performance for real-world graphs, including non-web graphs. In particular, we use the *BlockRank* algorithm [22], that naturally appears to suit a subgraph-centric model, as an algorithmic starting point and propose variations to better leverage the subgraph-centric abstraction. In the process, we make the following specific contributions in this paper:

1. We map the BlockRank algorithm to a subgraph-centric programming abstraction, and analyze its deficiencies,

2. We propose variations of the BlockRank algorithm including *Subgraph Rank*, and hypothesize their behavior in a subgraph-centric model, and

3. We implement the PageRank, BlockRank variations and Subgraph Rank algorithms using the GoFFish sub -graph-centric distributed graph platform, and experimentally evaluate their quality and performance across diverse real-world graphs on Amazon EC2 Cloud Virtual Machines (VMs).

The rest of the paper is organized as follows: in § 2, we offer a background of PageRank, BlockRank and subgraph-centric graph programming abstractions; in § 3, we introduce variations to the BlockRank algorithm that can improve its performance, and also propose the Subgraph Rank algorithm as a suitable candidate for subgraph-centric programming abstractions; in § 4, we evaluate our proposed algorithms on three real-world graphs using the GoFFish platform running on Amazon's public Cloud; we discuss related work in § 5, and summarize our contributions and provide directions for future work in § 6.

2. BACKGROUND

We provide background material on PageRank and BlockRank algorithms from prior literature, and subgraph-centric programming abstractions that is our target distributed platform.

2.1 Subgraph-Centric Abstractions

Vertex-centric graph programming abstractions have been proposed by distributed platforms like Pregel [28] and GraphLab [26]. In Pregel, vertices of a graph are partitioned across multiple machines, and the computation is performed from the view of a single vertex in a pleasingly parallel manner. Coordination between distributed vertices takes place through synchronized message passing at superstep boundaries. The graph applications progress iteratively, one superstep at a time, interleaving computing and communication. Bulk Synchronous Parallel (BSP) [12] processing allows efficient bulk transfer of messages on slower, commodity networks, while also avoiding potential race conditions and circular dependencies of distributed task execution.

In a *subgraph-centric programming abstraction*, the computation is performed at the coarser granularity of a subgraph, with synchronized messages passed by subgraphs to its neighboring subgraphs at superstep boundaries. While platforms like Giraph++ [35] treat each graph partition as a (disconnected) subgraph, others like our own GoFFish platform [34] identify weakly connected components within each partition as units of subgraph execution. For consistency, we assume the latter definition of subgraphs as weakly connected components though our results translate to both definitions. Subgraph centric programming has been shown to be faster than vertex centric programming due to better use of shared memory algorithms on entire subgraphs, reduced message passing overheads, and fewer supersteps to convergence for graph applications.

Algorithm 1 Subgraph-Centric PageRank (SGPR)

```
 1: procedure COMPUTE(Subgraph $SG$,Message msg[])
 2:     if $superStep$ > MAX then                 ▷ Sanity check
 3:         VoteToHalt()
 4:     end if
 5:     if superStep == 1 then
 6:         for $v$ in $SG.vertices$ do
 7:             pr[$v$] = $\frac{1}{|G.vertices|}$             ▷ Initialize PR
 8:         end for
 9:         SendMessageToNeighbors(pr[])
10:     else
11:         sums[] = ComputePRSums(pr)
12:                 ▷ also includes local contributions
13:         for $v$ in $SG.vertices$ do                ▷ Update PR
14:             pr[$v$] = 0.85×sums[$v$]+0.15 × $\frac{1}{|G.vertices|}$
15:         end for
16:         if superStep == 30 then
17:             VoteToHalt()                 ▷ Halt after 30 iters
18:         else
19:             SendMessageToNeighbors(pr[])
20:         end if
21:     end if
22: end procedure
```

GoFFish, used to evaluate our algorithms in this paper, partitions graphs across multiple hosts or VMs, identifies subgraphs within each partition, and stores the subgraph and its attributes within its GoFS distributed storage. It is optimized for a write-once, read-many batch processing model. Subgraphs have local edges between their local vertices, and remote edges connecting to subgraphs in other partitions. Graphs are partitioned using METIS [23] to minimize the edge-cuts between partitions and balance the number of vertices per partition. Subgraph centric applications composed in GoFFish are executed on distributed hosts using the Floe Dataflow engine which implements the BSP execution model.

2.2 PageRank and BlockRank

PageRank is a centrality measure that indicates the relative importance of a vertex within a graph [29]. The PageRank of a webpage (vertex) in a web graph, with hyperlinks forming edges, is the probability that a web surfer who is performing a random walk on the web graph will end up at that page when following the links [7]. For a given webpage $v \in \mathbb{V}$ in a web graph $G = (\mathbb{V}, \mathbb{E})$ with the set of vertices $\mathbb{V}$ (webpages) and edges $\mathbb{E}$ (hyperlinks), its PageRank $\mathcal{P}(v)$ is given by the following iterative logic:

$$\mathcal{P}_0(v) = \frac{1}{n} \quad (1)$$

$$\mathcal{P}_{i+1}(v) = \alpha \times \left(\sum\nolimits_{\omega \in \mathcal{I}(v)} \frac{\mathcal{P}_i(\omega)}{|\mathcal{O}(\omega)|}\right) + (1-\alpha) \times \frac{1}{n} \quad (2)$$

$$\mathcal{P}(v) = \mathcal{P}_{i+1}(v) \mid \forall v \in \mathbb{V}, |\mathcal{P}_{i+1}(v) - \mathcal{P}_i(v)| < \epsilon \quad (3)$$

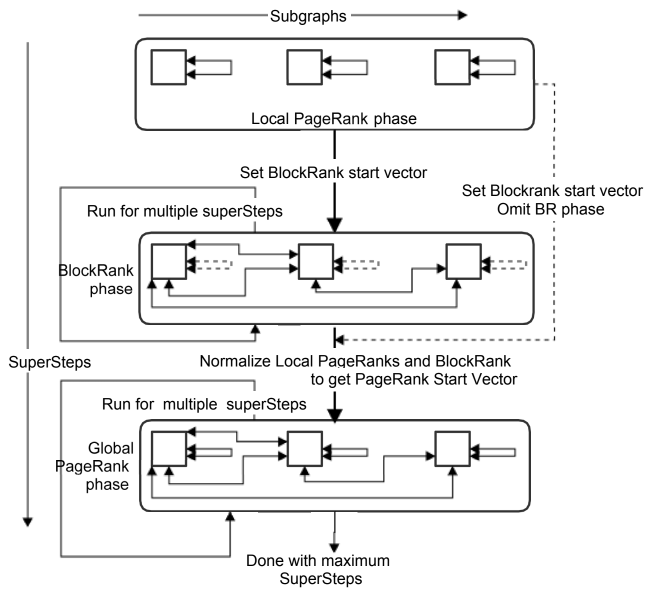
where $n = |\mathbb{V}|$ is the number of vertices in the web graph, $|\mathcal{O}(v)|$ is the out-degree of the vertex v and $\mathcal{I}(v)$ is the set of neighboring vertices that have incoming edges into v. α gives the probability with which the random walk will follow an outgoing link from a vertex, with $(1-\alpha)$ being the probability of taking a jump elsewhere. ϵ is a distance threshold beyond which successive iterations of PageRank should not change by, for the algorithm to terminate. Eqn. 1 initializes the PageRank to the probability of starting at any random vertex in the graph, Eqn. 2 is the iterative step which updates a vertex's PageRank values based on the weighted values reported by its neighbors, while Eqn. 3 is the halting condition where the PageRank has quiesced for all vertices across the graph.

PageRank has been a *de facto* graph algorithm for validating graph platforms, and the two recent subgraph-centric platforms, Giraph++ and GoFFish, have both mapped PageRank using a subgraph-centric programming model. Alg. 1 lists the pseudo-code for a subgraph-centric PageRank. After initialization the PageRank for all vertices in the subgraph in the first superstep, each subgraph sends the PageRank values for its vertices to their neighboring vertices present in remote subgraphs. In each subsequent superstep, the subgraph uses the PageRank values for neighboring vertices available from the received messages to update the PageRank value for its vertices, taking a damping factor into consideration. The algorithm either runs till a threshold value of convergence is reached, or for a fixed number of supersteps (30 shown in Alg. 1). As can be seen, the subgraph-centric algorithm still iterates through every vertex and applies the update in each superstep, mimicking the behavior of a vertex-centric algorithm. This makes this algorithm a naïve mapping to a subgraph-centric model.

Though PageRank was defined for web graphs and for link analysis, the measure is used in many other domains [13] like in Social Network, Citation Networks, and Road Network analysis. Hence, scaling PageRank on distributed platforms for diverse graphs has the potential to benefit multiple domains.

Different flavors of PageRank algorithms have been proposed, both to improve the quality of the ranks and to speed up convergence. These include Topic-sensitive PageRanks [18], Personalized PageRanks [20], and BlockRank [22]. The *BlockRank* algorithm is based on the idea that a general web graph has an inherent block structure, i.e., sets of webpages with a high concentration of interconnected hyperlinks, such as found between pages within a web domain. The intra-block edges tend toward a clique-like structure within a block, while the inter-block edges are sparse.

The generic BlockRank algorithm operates over three phases. In the *Local PageRank* phase, localized PageRank values are calculated for each vertex in a block by omitting all the remote links between the blocks. Next, in the *BlockRank* phase, a BlockRank value, which represents the relative importance of each block in the graph, is computed. For this, we consider each block as a meta-vertex in a meta-graph, and edges between blocks as meta-edges. A variation of the PageRank algorithm is performed on this meta-graph, and the PageRank for each block (meta-vertex) is its BlockRank. In the last *Global PageRank* phase, we estimate the initial PageRank values for each vertex by combining their Local PageRank with the BlockRank, and then perform the



Subgraphs
Local PageRank phase
Set BlockRank start vector
Set Blockrank start vector
Omit BR phase
Run for multiple superSteps
BlockRank phase
SuperSteps
Normalize Local PageRanks and BlockRank
to get PageRank Start Vector
Run for multiple superSteps
Global PageRank phase
Done with maximum SuperSteps

Figure 1: Flowchart of BlockRank algorithm phases. Boxes listed horizontally show different subgraphs, and boxes listed vertically span supersteps.

standard PageRank algorithm using these initial values.The authors [22] show that the BlockRank algorithm converges faster than traditional PageRank, and they offer theoretical performance bounds for block-structured graphs.

3. BLOCKRANK & SUBGRAPH RANK

In this section, we discuss BlockRank and its variations for a subgraph-centric model, and introduce the Subgraph Rank algorithm.

3.1 Native BlockRank Algorithm (BRNA)

One of the challenges of running a naïve PageRank algorithm using a subgraph-centric model (SGPR) is that its execution behavior mimics that of a vertex-centric version [34]. In every superstep, the PageRank value of a vertex is updated and passed as messages to its neighbors. There by, the subgraph as a whole is unable to make progress without each localized vertex making identical progress. Exposing subgraph-level computation would mitigate this downside, and effectively leverage the abstraction.

The BlockRank algorithm leverages the block structure of the web to speed up the convergence of PageRank. A graph is said to have a block-structure if, when considering it as an adjacency matrix, there are blocks within the matrix that have high intra-vertex edge connectivity and the inter-block edge connectivity is sparse. This algorithm is theoretically proven to perform better than PageRank for block-structured graphs. Since blocks loosely correspond to the notion of subgraphs, it is worthwhile mapping BlockRank to a subgraph-centric model to potentially do better than naïve PageRank.

Fig. 1 shows the phases involved in performing BlockRank using a subgraph-centric model. The corresponding pseudocode is listed in Alg. 2. There are three key phases in BlockRank, as mentioned before: (1) Local PageRank (LPR) computation, (2) BlockRank computation, and (3) Global

PageRank (GPR) computation. Each of these phases are performed using a subgraph-centric abstraction, with computation distributed across subgraphs (horizontal boxes in Fig. 1) and proceeding iteratively over one or more supersteps (vertical boxes).

The LPR phase runs in the first superstep and here, each subgraph operates independently as its own graph, ignoring remote edges to other subgraphs, and calculates the PageRank for each of its vertices (pr[v]) using the standard iterative PageRank algorithm, in-memory. This is shown in Alg. 2, lines 6–17. It then proceeds to the BlockRank phase where each subgraph (block) is treated as a vertex in a meta-graph, and the BlockRank for each subgraph is calculated, starting with an initial value that equals $\frac{1}{|Subgraphs|}$ and iterating over supersteps to exchange BlockRank values between subgraphs after each superstep (Alg. 2, lines 20–32). This phase runs for a fixed number of supersteps – we set this to 10 supersteps since that achieves a reasonable BlockRank convergence for meta-graphs with up to 2700 subgraphs (meta-vertices) that we observe. The BlockRank values (br) calculated at the end of this phase are used as weighing factors to initialize the PageRank for each vertex of the graph (Alg. 2, line 30), referred to as "distributing" the BlockRank. Following this BlockRank distribution, the GPR phase starts with these initialized PageRank values and runs over multiple supersteps until convergence.

While we can map the BlockRank to a subgraph-centric model, we perceive shortcomings to this algorithm (substantiated in the evaluation, § 4) in practice:

- General graphs may not have a strong block-structure to them, and as such the BlockRank algorithm may not perform well for such graphs.
- The block structure only loosely maps to a subgraph. Since the block sizes of graphs can vary widely, partitioning these graphs can end up splitting blocks across multiple subgraphs, thereby impacting the benefits of block-level computations.

Hence, we propose a more robust solutions to calculating PageRank that goes beyond the native BlockRank algorithm.

Algorithm 2 Subgraph-centric Native BlockRank (BRNA)

```
1: procedure COMPUTE(Subgraph $SG$,Message msg[])
2:     if $superStep > $ MAX then                          ▶ Sanity check
3:         VoteToHalt()
4:     end if
5:     if $superStep == 1$ then                       ▶ Local PageRank
6:         for $v$ in $SG.vertices$ do
7:             pr[$v$] $= \frac{1}{|SG.vertices|}$                ▶ Initialize PR
8:         end for
9:         do
10:            sums[] = ComputePRSums(pr)
11:            $L1Norm = 0$
12:            for v in $SG.vertices$ do                  ▶ Update PR
13:                $prev$ = pr[$v$]
14:                pr[$v$] $= 0.85 \times$ sums[$v$]$+0.15 \times \frac{1}{|SG.vertices|}$
15:                $L1norm = L1norm +$ Abs($prev$ - pr[$v$])
16:            end for
17:        while $L1Norm > \epsilon$                    ▶ Test convergence
18:        $isBRActive$ = TRUE
19:    else if $isBRActive$ then                      ▶ BlockRank
20:        if IsFirstBRSuperStep() then
21:            $br = \frac{1}{|Subgraphs|}$                  ▶ Initialize BR
22:        else            ▶ Update BR
23:            $sum$ = ComputeBRSum(msg[])
24:            $br = 0.85 \times sum + 0.15 \times \frac{1}{|Subgraphs|}$
25:        end if
26:        SendMessageNeighbors($br$)
27:        if IsMaxBRSuperSteps() then
28:            isBRActive = FALSE
29:            for v in $SG.vertices$ do
30:                pr[$v$] = $br \times$ pr[$v$]              ▶ Distribute BR
31:            end for
32:        end if
33:    else        ▶ Global PageRank
34:        DoPageRank($SG$, pr[], msg[])
35:                ▶ VoteToHalt() on convergence
36:    end if
37: end procedure
```

3.2 Dimensions for Varying BlockRank

We can vary the native BlockRank algorithm along several dimensions to explore a variant that is more suitable to subgraph-centric models and for general graphs. Some of these change dimensions are discussed first, followed by the variations of BlockRank algorithms that include one or more of these. In this section, consider the graph $G = (\mathbb{V}, \mathbb{E})$ with the set of vertices $\mathbb{V}$ and edges $\mathbb{E}$ is partitioned into a set of subgraphs (blocks) $S_i = (V_i, E_i) \in \mathbb{S}$, where $V_i \subset \mathbb{V}$ and $E_i \subset \mathbb{E}$.

3.2.1 BlockRank Initialization Vector

SGC-inverse: The BlockRank phase has to start with an initial BlockRank value for each subgraph (block) in the first iteration. This choice is significant and affects the BlockRank phase. A natural choice is a uniform distribution, similar to PageRank, across all the blocks given by $\frac{1}{|\mathbb{S}|}$, where $|\mathbb{S}|$ is the count of subgraphs in the graph (hence, subgraph count or SGC). So every subgraph gets the same initial BlockRank value irrespective of its structure.

SG-by-G: Here the initial value of BlockRank is weighed by the number of vertices contributed by the subgraph to the entire graph, and given as $\frac{|V_i|}{|\mathbb{V}|}$ for the subgraph S_i, where $|V_i|$ is the number of vertices in S_i and $|\mathbb{V}|$ is the number of vertices in the entire graph. Intuitively, this is introduces fairness in the allocation that accounts for the varying sizes of different blocks.

3.2.2 BlockRank Distribution

Native: In the PageRank algorithm, a fraction $\frac{1}{n}$ of the PageRank values $\mathcal{P}(v)$ for a vertex v is passed at the end of each iteration to each of its n neighboring out vertices, uniformly. This fraction $\frac{1}{n}$ is called the transition probability distributed from vertex v to each its neighbors. Similarly, in the BlockRank phase for the BRNA algorithm, the BlockRank of each subgraph (block) is distributed to its neighboring subgraphs after each BlockRank iteration. This distribution logic is more complex as we consider both *self-edges* from a subgraph to itself, and the *local PageRank* calculated in the LPR phase for vertices in that subgraph in calculating the *block*-transition probability.

For a vertex $v \in \mathbb{V}$, its local PageRank is given by $\mathcal{LP}(v)$. The BlockRank at iteration 0 for a subgraph S_i is given by $\mathcal{BR}_0(S_i) = \frac{1}{|\mathbb{S}|}$. Now for every vertex $v \in V_i$ for the subgraph (block) S_i, we use $\mathcal{LP}(v)$ to calculate the block-transition probability, $\beta_k^{S_i \to S_j}$, from one subgraph S_i to S_j at the k^{th} iteration.

$$\forall S_i = (V_i, E_i) \in \mathbb{S}, \text{we have } \sum\nolimits_{v \in V_i} \mathcal{LP}(v) = 1 \qquad (4)$$

$\forall S_i, S_j \in \mathbb{S}$, we have

$$\beta_k^{S_i \to S_j} = \sum_{v \in S_i, w \in S_j, \langle v,w \rangle \in E_i \& E_j} \frac{\mathcal{LP}(v)}{|\mathcal{O}(v)|} \qquad (5)$$

$$\mathcal{BR}_{k+1}(S_j) = \alpha \times \left(\sum\nolimits_{S_i \in \mathbb{S}} \beta_k^{S_i \to S_j} \times \mathcal{BR}_k(S_i) \right) + (1-\alpha) \times \frac{1}{|\mathbb{S}|} \qquad (6)$$

where $|\mathbb{S}|$ is the number of subgraphs (blocks) in the graph, $\langle v, w \rangle \in E_i \& E_j$ refers to edges incident from vertices in S_i to S_j, and $|\mathcal{O}(v)|$ and α follow same conversion as for PageRank. Eqn. 4 asserts that the sum of local PageRank values inside a subgraph is 1. Eqn. 5 calculates the block-transition probability from subgraph S_i to S_j if there is an edge from one to the other. It also considers self-edges ($S_i \to S_i$) and local PageRanks; a fraction $\frac{\mathcal{LP}(v)}{|\mathcal{O}(v)|}$ of the BlockRank will be passed to S_j. Eqn. 6 applies this block-transition probability to get the BlockRank of the subgraph for the next iteration.

So this gives consideration to self edges from the subgraph to itself ($\beta_k^{S_i \to S_i}$), and also includes the local PageRank so that important local vertices within the subgraph pass on a higher fraction of the BlockRank values to its neighbors.

PageRank-like: In this adaptation, self-edges within a subgraph and local PageRank values for vertices in the subgraph are ignored in the block-transition probability. Only edges that go from one subgraph to another are taken into consideration. So we construct a meta-graph with the subgraphs as the meta-vertices and edges between subgraphs as the meta-edges and run PageRank on the meta-Graph. In the equation below, $\mathcal{I}()$ and $\mathcal{O}()$ refer to the in-coming and out-going meta-vertices (subgraphs) connected to a subgraph.

$$\mathcal{BR}_{k+1}(S_j) = \alpha \times \left(\sum\nolimits_{S_i \in \mathcal{I}(S_j)} \frac{\mathcal{BR}_k(S_i)}{|\mathcal{O}(S_i)|} \right) + (1-\alpha) \times \frac{1}{|\mathbb{S}|} \qquad (7)$$

3.3 Variations of BlockRank Algorithm

3.3.1 Native BlockRank (BRNA)

BRNA is a direct implementation of the BlockRank to a subgraph-centric model, as discussed in § 3.1 where each subgraph is treated as a block. Since while partitioning the graphs across distributed machines [23] we balance the number of vertices and minimizing the edge-cuts across partitions, a subgraph partially behaves like a block. This uses native BlockRank distribution logic and SGC-inverse initialization Vector.

3.3.2 BlockRank with PageRank-like Distribution Logic (BRDL)

Here the BRNA algorithm is used with a PageRank-like BlockRank distribution Logic instead of the native logic. Consequently, since the BlockRank phase effectively runs a PageRank algorithm on the meta-graph, it is more intuitive. BRDL completes the LPR phase in the first superstep and then proceeds to the BlockRank phase. We start with SGC-inverse initialization vector and follow PageRank-like BlockRank distribution for 10 supersteps, before switching to a global PageRank phase.

3.3.3 BlockRank with SG-by-G Initialization Vector (BRIV)

In BRIV, the BRNA algorithm changed to use SG-by-G initialization vector. Using SG-by-G ensures a fairer initial BlockRank value. One of the key rationales for using BlockRank is to compute a better start value for the global PageRank to allow rapid convergence. Vertices in smaller subgraphs have a higher local PageRank value compared to vertices in a larger subgraph since the sum of the local PageRank is 1 in both cases. Since the local PageRank is weighted with the BlockRank computed for that subgraph, using SG-by-G as the initial BlockRank allows vertices in larger subgraphs to regain their importance. As a result, we get a better estimate of the relative importance of the subgraphs, which is in the spirit of the original BlockRank paper. Note that SGC-inverse is a special case of SG-by-G where all subgraphs are of equal size.

3.3.4 BlockRank with PageRank-like Distribution Logic and SG-by-G Initialization Vector (BRDI)

Here, BRNA is modified to use SG-by-G as the initialization vector for the BlockRank Phase and uses PageRank-like BlockRank distribution logic. This couples the features of both BRDL and BRIV.

3.3.5 BlockRank without BlockRank Distribution (BRNO)

In this variation from BRNA, we skip running the BlockRank phase by setting the BlockRank value as $\frac{1}{|\mathbb{S}|}$ and passing it on to the initialization of the Global PageRank. We retain the initialization vector of SGC-inverse. As discussed before, it is observed that both large and small subgraphs end up converging to a uniform BlockRank value at the end of the BlockRank phase in the other variations. So this algorithm altogether avoids the BlockRank phase and sets the BlockRank directly to this uniform value, $br = \frac{1}{|\mathbb{S}|}$ (Alg. 2, line 30).

3.3.6 Subgraph Rank (SGRK)

The Subgraph Rank algorithm operates by changing the initialization vector for BlockRank in BRNO to SG-by-G, thereby intuitively captures the spirit of the original BlockRank algorithm for a general graph. The local PageRank phase is the same as for all BlockRank algorithms. In the BlockRank phase, we set the initial BlockRank for each subgraph as SG-by-G, and pass this value on directly to the global PageRank initialization without running any BlockRank supersteps. The pseudo-code for the Subgraph Rank algorithm is identical to the BRNA algorithm, Alg. 2, except that lines 20–26 to calculate BlockRank are replaced by the function: $br = \frac{|SG.vertices|}{|Graph.vertices|}$, i.e., the value of br

is the ratio of the number of vertices in the subgraph to the total number of vertices in the whole graph.

4. EXPERIMENTAL EVALUATION

We present an empirical evaluation of the baseline subgraph-centric PageRank and native BlockRank algorithms, and demonstrate the relative performance of the proposed BlockRank variants, including Subgraph Rank, compared to these baselines. The algorithms are implemented on our GoFFish subgraph-centric distributed graph platform, and run on Amazon EC2 Cloud VMs. We expect these results to generalize to other subgraph-centric platforms such as Giraph++, when run on any commodity cluster or Infrastructure as a Service (IaaS) Cloud. We discuss the graph datasets used in the evaluation, the experimental setup and metrics for success, before we present the results.

4.1 Graph Datasets

We choose three real-world graphs available from Stanford's SNAP graph repository for our evaluation. These are summarized in Table 1. The graphs selected have diverse topological characteristics, and span different application domains, to ensure that our experimental results on quality, performance and scalability can be generalized. The Amazon Product Network (**AMZN**)[1]is a small graph from the eCommerce space, has a higher vertex degree (2.76) and medium diameter (44). The California Road Network (**CARN**)[2]is a medium sized graph from the transportation domain with a large diameter (849) and smaller degree (1.4). The Wikipedia Talk Network (**WIKI**)[3]is a larger social network community graph with over 5 *million* edges and a small diameter (9). These graphs have also been used in literature for evaluating other graph platforms [16, 34]. All these graphs are deployed as undirected.

Table 1: Graph datasets used and their properties

Graph	Vertices	Edges	Diameter	Vertex Degree
AMZN	334,863	925,872	44	2.76
CARN	1,965,206	2,766,607	849	1.40
WIKI	2,394,385	5,021,410	9	2.00

4.2 Execution Environment

All the existing and proposed algorithms are implemented and executed on the GoFFish graph analytics platform [4] [34]. Since the emphasis of this paper is on the improvements offered by the algorithm, implementing them all on GoFFish allows us to do a fair comparative evaluation. Alternatives such as Giraph++ [35] that offer a subgraph-centric model would also behave similarly. The subgraph-centric programming model is a natural superset of a vertex-centric programming model, and hence the naïve PageRank algorithm can also be implemented on platforms like Apache Giraph [34]. However, this does not extend to the other BlockRank algorithms that rely on the subgraph-centric model. The GoFFish platform and the algorithms evaluated are implemented in Java, and executed using JDK 1.7.

[1]http://snap.stanford.edu/data/com-Amazon.html
[2]http://snap.stanford.edu/data/roadNet-CA.html
[3]http://snap.stanford.edu/data/wiki-Talk.html
[4]http://github.com/usc-cloud/goffish

The experiments are run on Amazon Web Services (AWS) IaaS public cloud [5]. We use either `m3.large` or `m3.xlarge` Elastic Compute Cloud (EC2) Virtual Machines (VMs), as noted later. The specifications of the VM such as the number of virtual CPU cores and SSD-based local disk storage are given in Table 2. The guest OS on the VM is based on 64-bit Linux.

Table 2: AWS EC2 VM Specifications

VM Type	vCPU	Memory	Disk	Bandwidth[†]
`m3.large`	2	7.5 GiB	32 GB	moderate (∼700Mbps)
`m3.xlarge`	4	15 GiB	80 GB	high (∼1100Mbps)

[†]Indicative network bandwidth from http://blog.flux7.com/blogs/benchmarks/benchmarking-network-performance-of-m1-and-m3-instances-using-iperf-tool

All the VMs act as worker nodes for GoFFish, host the graph partitions and execute the subgraph-centric application. One these worker VM play an additional role of hosting the coordinator task, though it is a light weight process. GoFFish is multi-threaded, with each worker using twice as many threads as the number of CPU cores, and each thread working on one subgraph at a time.

4.3 Evaluation Metrics

We propose *quality* and *performance* measures to evaluate the success of the existing and proposed algorithms to calculate the PageRank of the three graphs. The quality measure calculates the proximity of the proposed algorithms' solutions to a *near-optimal PageRank* solution for the graph. We define this near-optimal PageRank for a graph as the PageRank value for its vertices arrived at the 100^{th} iteration (superstep) of running a naïve PageRank algorithm. We then calculate the L1 Norm at the k^{th} iteration of the PageRank values provided by the evaluated algorithms ($\mathcal{P}^\star_k$), against the PageRank values from this near-optimal solution ($\mathcal{P}_{100}$). This *Distance from Convergence* (DFC) for each vertex v in the graph at superstep k is given by:

$$\text{DFC}(k) = \sum\nolimits_{v \in \mathbb{V}} |\mathcal{P}^\star_k(v) - \mathcal{P}_{100}(v)| \tag{8}$$

This approximation helps avoid the oscillatory nature of the PageRank solution even as it incrementally narrows toward convergence as the supersteps increase. For e.g., Fig. 2 shows the incremental L1 Norm *between two successive supersteps* for the CARN graph using a naïve PageRank algorithm, and its oscillations past 55 supersteps. Due to this behavior, using the L1 Norm between supersteps, which estimates the incremental change in PageRank value (Eqn. 3), as a metric for convergence can lead to incorrect premature halting. This approach has been used elsewhere too [35].

For our evaluation, we consider representative threshold distance values for DFC: $0.1, 0.01, 0.001$, and 0.0001 ; these distances are in short denoted by $\Delta_{0.1}$, $\Delta_{0.01}$, $\Delta_{0.001}$ and $\Delta_{0.0001}$. Depending on the quality of the PageRank results

[5]http://aws.amazon.com/ec2

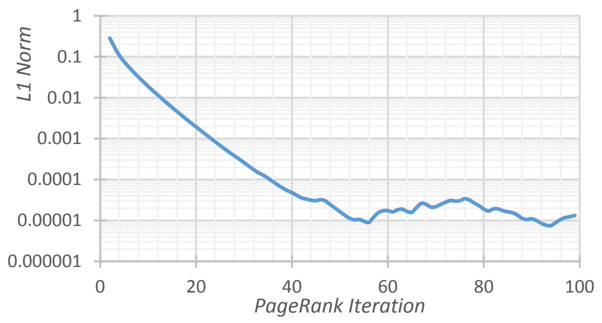


Figure 2: L1 Norm between PageRank values of successive supersteps for CARN using SGPR.

required for a graph, it suffices to iterate the algorithm till its PageRank reaches within one of these four Δ thresholds.

The performance measures we consider include the *number of supersteps* and the *wall clock time (Makespan)* the proposed algorithms take to reach within a particular Δ threshold from the near-optimal solution. We also consider the *scalability* of the algorithms given by the observed speedup as we increase or decrease the number of VMs on which the algorithms are run.

4.4 Baseline Experiments

Our prior work has shown that a naïve subgraph-centric implementation of the PageRank algorithm (**SGPR**) is only comparable to, but does not outperform, the vertex-centric PageRank algorithm [34]. The BlockRank algorithm [22], intuitively, is expected to improve the performance by leveraging the availability of the entire subgraph in local memory. We initially evaluate the effectiveness of this native BlockRank algorithm (**BRNA**), and compare it with the SGPR algorithm. Unless otherwise noted, all experiments are run thrice and the average of their values plotted.

We run the SGPR and BRNA algorithms on the AMZN and CARN graphs partitioned across 6 `m3.large` VMs, and run them to 100 supersteps. At every superstep, we record the DFC for the graph, and plot it in Fig. 3.

We see that the naïve Subgraph-centric PageRank, SGPR, that we wish to out-perform continues to converge in fewer supersteps than the native BlockRank algorithm, BRNA. BRNA takes 319 *secs* and 113 *secs* to run for 100 supersteps for AMZN and CARN, respectively, compared 321 *secs* and 86 *secs* for SGPR. While BRNA seems incrementally slower, the reality is worse when we consider the superstep and time at which BRNA reaches the $\Delta_{0.0001}$ threshold. For AMZN (Fig. 3a), BRNA reaches the threshold at superstep 70 (211 *secs*) and SGPR reaches it at superstep 31 (97 *secs*), while for CARN (Fig. 3b), BRNA reaches $\Delta_{0.0001}$ at superstep 74 (91 *secs*) and SGPR reaches it at superstep 32 (43 *secs*). So *BRNA is almost twice as slow as SGPR* in reaching convergence.

This is counter-intuitive, but can be explained. First, BRNA is reliant on a strict block-like structure of the graphs seen in web graphs, with subgraphs having a high edge density that borders on cliques [22]. The graph datasets from real-world networks that we consider do not have such an extreme block structure, and consequently the algorithm performs worse. Second, while GoFFish's partitioning algorithm (METIS) tries to reduce edge cuts between partitions and then identifies subgraph within each partition, the quality of partitioning may result in small subgraphs with just $100's$ of vertices, or split a large block across two subgraphs in different partitions. As a result, we see that BRNA is not usable as is for the general class of graphs, and for a subgraph-centric paradigm. This empirically motivates the need for better PageRank algorithms that can leverage the subgraph-centric programming abstraction that is available.

Note that the near-optimal solution is calculated using SGPR's PageRank value at superstep 100, and one would expect that the DFC for SGPR at superstep 100 to be 0 in Fig. 3. However, this is not the case since every run of the algorithm does not deterministically produce identical PageRank values. As a result, the DFC proximity measure is a more appropriate metric. Even there, *having a DFC value smaller than* 0.0001 *may not be meaningful*. So we adopt $\Delta_{0.0001}$ as our upper bound of PageRank quality.

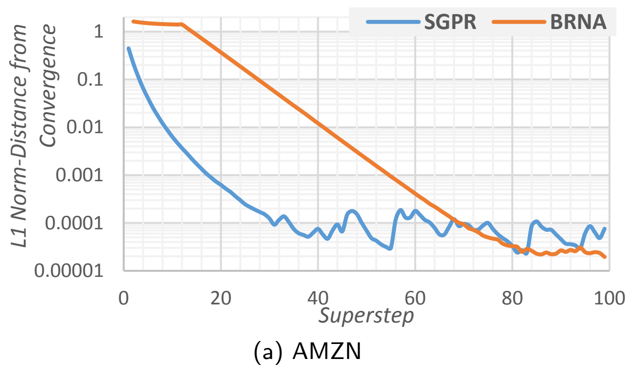
4.5 Quality Analysis of BlockRank Variations

We evaluate the proposed five variations to the BlockRank algorithms (**BRDL, BRIV, BRDI, BRNO**) from § 3, including the Subgraph Rank (**SGRK**) algorithm, and compare them against the SGPR and BRNA baseline algorithms. Fig. 4 shows the DFC for these seven algorithms for the AMZN and the CARN graphs when run on 6 `m3.large` worker VMs to 100 supersteps. We can make several observations from these plots.

The DFC for BlockRank variations are affected by the three phases of their algorithm: LPR, BlockRank and GPR. The artifact of LPR phase is seen in the DFC value at the initial superstep, of BlockRank phase (if present) over the next 10 supersteps, and the GPR phase for subsequent supersteps. Overall, we can see that SGRK consistently outperforms all the other algorithms, while the BlockRank variants, BRNA, BRDL, BRIV, BRDI, and BRNO underperform the baseline SGPR. However, there are subtle but consistent patterns unique to each phase, and each algorithmic variation that reflects the different initialization vectors and BlockRank distribution logic that are attempted. We discuss these next to help explain how SGRK is a successful refinement of the other BlockRank variations.

The BlockRank algorithms that use the native initialization vector of SGC-inverse – BRNA, BRDL and BRNO – end up with a larger distance from convergence at the first superstep, with a DFC value > 1.0. However, the algorithms that use the proposed alternative SG-by-G initialization vector end up with a lower (better) DFC value at their initial superstep (e.g. < 0.1 for AMZN and < 0.001 for CARN). This confirms our hypothesis that using a normalized BlockRank initialization value that considers the number of vertices in the subgraph is fairer and offers a better initial estimate of the eventual PageRank value.

In the initial supersteps, when the BlockRank phase is taking place, we see the distinctive behavior of the algorithms. BRNA and BRIV both use the native BlockRank distribution logic that not only includes remote edges in its computation of BlockRank but also the internal edges within a subgraph and the local PageRank for their vertices (Eqn. 6. At the end of the BlockRank phase, the DFC for both these two algorithms end up at near identical points – gradually decreasing from a high DFC for BRNA, and sharply increasing from a lower DFC for BRIV to ~ 1.0 for AMZN. Even when using the alternative PageRank-like distribution logic



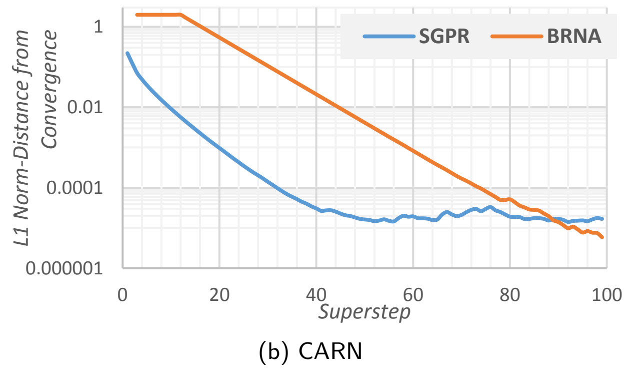
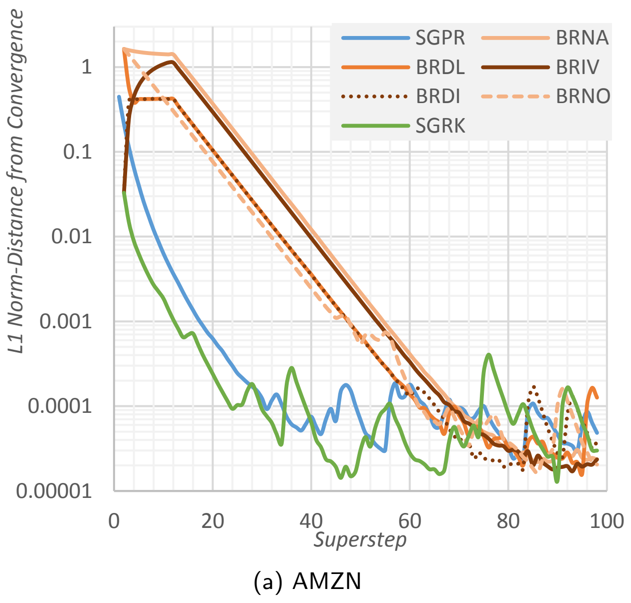


Figure 3: DFC at every superstep, using SGPR and BRNA algorithms, for AMZN and CARN graphs.



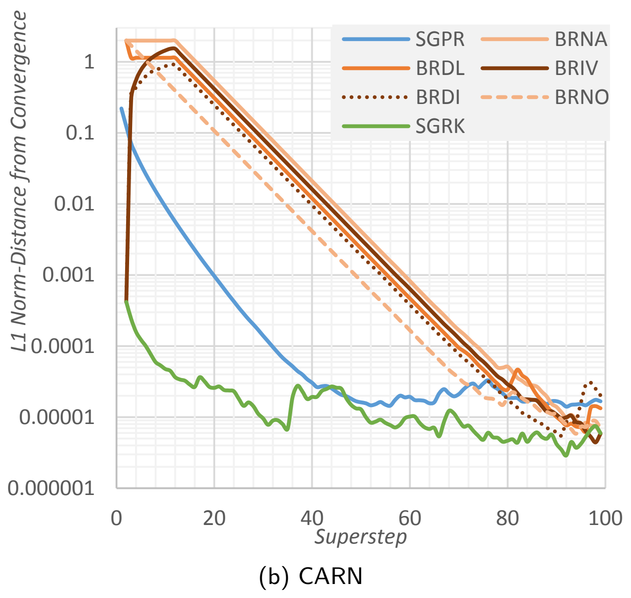


Figure 4: DFC at every superstep, using SGPR, the five BlockRank variations – BRNA, BRDL, BRIV, BRDI and BRNO – and SGRK algorithms, for AMZN and CARN graphs.

for BRDL and BRDI, we see a similar trend whereby the BlockRank phase causes the DFC values to converge to a similar point of ~ 0.4 for AMZN, though this value is closer to convergence than using the native distribution. In effect, the BlockRank phase, using either of the distributions, effectively converges to a specific value that does not offer any benefits. We observe that this constant value is close to $\frac{1}{|\mathbb{S}|}$ for the native BlockRank distribution.

Note that during the BlockRank phase, we assume that the PageRank value for each vertex in a subgraph at each superstep is the product of its local PageRank from the LPR phase and the current BlockRank value for the subgraph at that superstep. This allows us to compute the DFC at each superstep. Also, since the BlockRank phase operates on the meta-graph that is distributed across p partitions, the maximum diameter of the meta-graph is p, and this leads to the BlockRank values converging rapidly within a few supersteps.

In fact, BRNO, that skips having a BlockRank phase and directly uses a constant value as the BlockRank manages to make steady progress in converging during the 10 BlockRank supersteps it would otherwise have spent calculating BlockRank. The slope of the reduction in DFC is steady for the five BlockRank variants once they reach the global PageRank phase. Hence the initial PageRank at the start of the GPR phase impacts how quickly the algorithms converge.

SGRK, however, performs significantly better than the other BlockRank variations as well as SGPR. It leverages the best features of using a fair initialization vector like BRIV and BRDI, giving it an early advantage, and also avoids the pitfalls of the BlockRank phase, like BRNO. The impact of the initialization phase using the normalized SG-by-G weights is tangible. We see that the DFC at the start of the global PageRank in superstep 2 for SGRK is much smaller than the initial DFC for SGPR in superstep 1, with DFC values of 0.03291 vs. 0.21355 for AMZN (Fig. 4a), and 0.00042 vs. 0.22030 for CARN (Fig. 4b). This order(s) of magnitude improvement in the initial BlockRank phase is the key

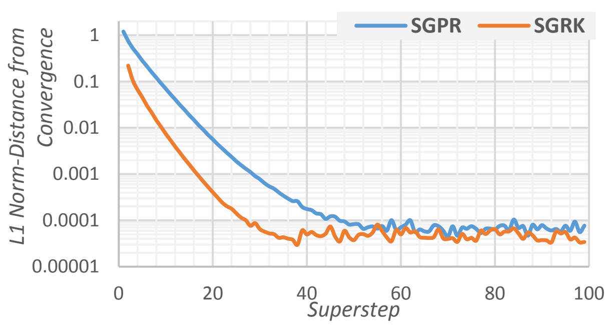


Figure 5: DFC at every superstep, using SGPR and SGRK algorithms, for WIKI graph.

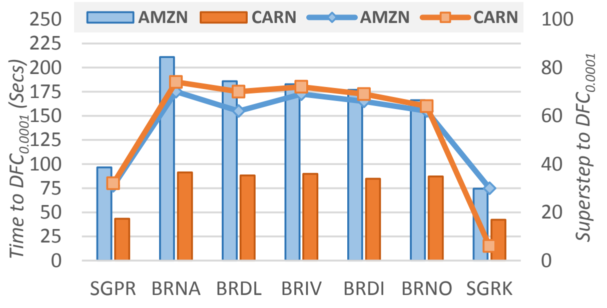


Figure 6: Time taken to reach $\Delta_{0.0001}$ (primary Y Axis, bar) and superstep reached at (secondary Y Axis, line), using the different algorithms, for AMZN and CARN graphs.

benefit that we expect from leveraging the subgraph-centric programming abstraction. The subsequent supersteps, once we are in the global PageRank phase, show more modest reductions in DFC that is comparable to SGPR.

This benefit extends to the WIKI graph too. Fig. 5 shows the DFC for the WIKI graph when running SGPR and SGRK on 6 `m3.xlarge` worker VMs to 100 supersteps – the larger VMs are necessary to ensure that the graph partitions and messages buffered at the end of each superstep fit in memory. We see that the SGRK has a DFC of 0.22010 at superstep 2 while SGPR has a DFC of 1.19161 when it is initialized, and SGRK also has a sharper convergence slope for subsequent supersteps than SGPR.

As such, SGRK is qualitatively better than the other BlockRank algorithms or the naïve PageRank algorithm. The local PageRank values which we compute in the first superstep are relatively close to the near-optimal PageRank values. This is thanks to the partitioning that is done during graph deployment to minimize edge cuts across partitions and balance the number of vertices in each partition. This has the effect of identifying block-structures that may exist, and manifesting them as subgraphs that are more generally applicable to even graphs without a strong block structure (like CARN). When normalizing the local PageRank values to bootstrap the global PageRank phase, we factor in the fraction of vertices held by the subgraph as it enables fairer allocation and is robust to for subgraphs of varying sizes

4.6 Performance Analysis of BlockRank Variations

Performance is another important metric which determines the success of an algorithm. The proposed algorithms have different time complexities for their different phases. For the LPR phase, the time complexity is $O(m \times (V_i^{MAX} + E_j^{MAX}))$, where each in-memory iteration m is linear in terms of vertex count and edge count of the largest subgraph. For the BlockRank phase, the time complexity is similar since it is like PageRank applied to the meta-graph formed from the subgraphs as vertices: $O(10 \times (|\mathbb{S}| + E_i^{remote}))$, where 10 is the constant number of supersteps the BlockRank phase runs for, $|\mathbb{S}|$ is the number of subgraphs (meta-vertices) in the graph and E_i^{remote} is the number of remote (meta) edges. For the GPR phase, the time complexity is $O(\frac{q}{p} \times (|\mathbb{V}| + |\mathbb{E}|))$ where $|\mathbb{V}|$ and $|\mathbb{E}|$ are the number of vertices and edges for the whole graph, p is the number of partitions, and q is the number supersteps taken to converge to a required Δ value. As we can see, the LPR and BlockRank phases have much smaller time complexity than the GPR phase, and hence the more progress that can be made in the former toward convergence, the better the performance.

We quantify the performance of these algorithms by plotting the time that they take to achieve $\Delta_{0.0001}$, and the superstep that they achieve this in. Fig. 6 shows these plots for the five BlockRank variations and the two baseline algorithms for AMZN and CARN. These were run on 6 `m3.large` worker VMs.

The BlockRank algorithms spend the first few supersteps in the initialization and LPR phases, after which they switch to the GPR supersteps that is similar to SGPR. For all algorithms, the time taken per superstep in the GPR phase is uniform and observed to have a linear time complexity. So we see from the figure that the number supersteps and the time taken are usually proportional (line and bar graphs). For e.g., we observe (not plotted) that for AMZN, both SGPR and SGRK take 3.2 *secs* per superstep, with less than a second spent in the initial supersteps for SGRK. On the other hand, for CARN, both SGPR and SGRK spend 0.66 *secs* per superstep while the initialization superstep takes about 29 *secs* for SGRK. This is because, CARN has about 4× more vertices and edges than AMZN, and the LPR phase in each subgraph takes longer to converge to an L1 Norm of 0.0001. This slower LPR phase also explains why SGRK takes about 41 *secs* to converge to $\Delta_{0.0001}$ despite reaching there in 6 supersteps. The WIKI graph takes about 123 *secs* per superstep for SGPR and SGRK, and a relatively modest 36 *secs* is spent in LPR for SGRK. Note that WIKI runs on an `m3.xlarge` machine which is twice as capable as `m3.large` that CARN runs on. Most of the time (97%) per superstep for WIKI is taken for passing PageRank update messages over the network between supersteps, compared to the time spent on computing the PageRank (3%).

As we can see from Fig. 6, SGPR and SGRK reach $\Delta_{0.0001}$ earlier than the other BlockRank variations. BRNA is the slowest due to poorer BlockRank initialization vector and the time spent on the BlockRank phase. But this extra time taken during the initial supersteps does not offer a faster convergence. The other BlockRank variants are marginally better than the baseline BRNA. We also note that SGRK is consistently faster than SGPR for both AMZN and CARN, giving 23% and 2% speed improvements, respectively. Also, while CARN is a larger graph than AMZN, it converges

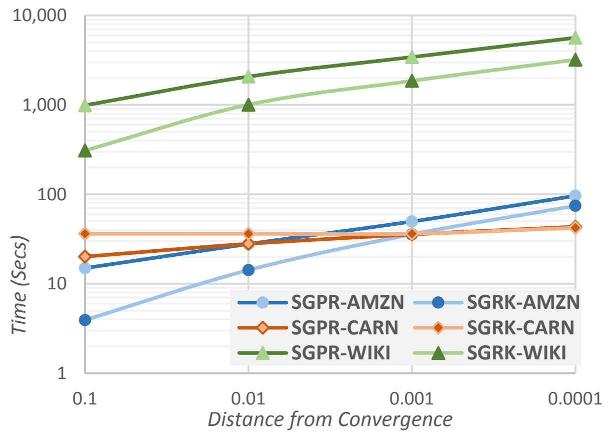


Figure 7: Time taken to reach $\Delta_{0.1-0.0001}$ using SGPR and SGRK for AMZN, CARN and WIKI graphs.

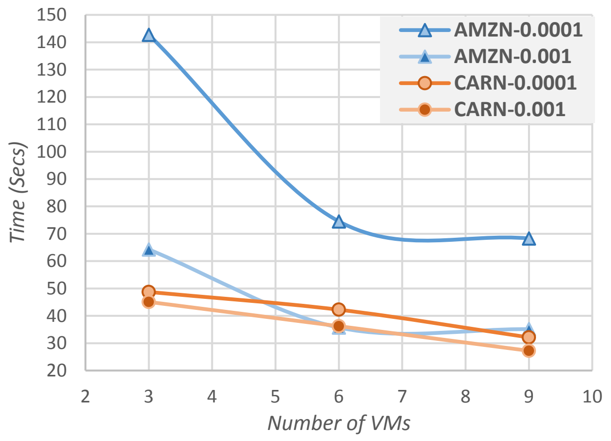


Figure 8: Time taken to reach $\Delta_{0.001}$ and $\Delta_{0.0001}$ using SGRK for AMZN and CARN graphs, as number of VMs increase from 3 to 9.

faster. This is because CARN has a much larger diameter than AMZN, which means subgraphs are sparsely connected, and hence local PageRank values dominate and network-overhead due to edge cuts across partitions is lower. Since the algorithms are often network bound than compute bound, this leads to a lower per-superstep time for CARN.

The performance benefits of SGRK over SGPR is even better, in many cases, when a lower PageRank quality is adequate. Fig. 7 shows the time taken to reach quality values of $\Delta_{0.1}$, $\Delta_{0.01}$, $\Delta_{0.001}$ and $\Delta_{0.0001}$ for AMZN, CARN and WIKI graphs for both these algorithms. These were run on 6 `m3.large` worker VMs for AMZN and CARN, and on as many `m3.xlarge` VMs for WIKI. SGRK for AMZN converges to $\Delta_{0.01}$ 49% faster than SGPR, while it converges to $\Delta_{0.0001}$ 23% faster than SGPR. Similarly, we see that the time for SGRK on WIKI is smaller than SGPR by $69\%, 52\%, 46\%$ and 43% for $\Delta_{0.1}$, $\Delta_{0.01}$, $\Delta_{0.001}$ and $\Delta_{0.0001}$, respectively. This means that SGRK gives comparable results as SGPR in half the time for WIKI, saving between $11 - 40$ *mins*. These indicate diminishing returns as we run more supersteps: the rapid gain in convergence distance within the first few supersteps of the Subgraph Rank is mitigated as more time is spent on the GPR supersteps.

SGRK for CARN does not outperform SGPR for larger values of DFC, above 0.0001. Since CARN has a large diameter of 849, the LPR phase took much longer to converge to an L1 Norm of 0.0001. So even though the GPR supersteps were fewer to reach a higher quality, and each GPR superstep is faster, the initial overhead of LPR cannot be surmounted. In fact, just the LPR phase lands the DFC value at < 0.001, and the incremental time in the GPR phase to reach $\Delta_{0.0001}$ is relatively negligible.

4.7 Scalability of Subgraph Rank

Lastly, we examine the scalability of these SGRK algorithm as the graph is partitioned to 3, 6 and 9 worker VMs for AMZN and CARN. We use the time taken to reach PageRank qualities of $\Delta_{0.001}$ and $\Delta_{0.0001}$ to estimate the scaling, and these are plotted in Fig. 8. We see that for AMZN, there is a linear speedup as the number of VMs increases from 3 to 6 to reach both $\Delta_{0.001}$ and $\Delta_{0.0001}$, with the time taken reducing from 142 *secs* to 74 *secs*. However, this benefit flattens out when moving from 6 to 9 VMs, with only an $\sim 8\%$ reduction in time. For CARN, the scaling is more gentle, giving a 25% and 66% speedup with a $2\times$ and $3\times$ more VMs. Increasing the number of VMs can increase the degree of parallelism but also cause more (and costlier, due to network I/O) supersteps to be required in the GPR phase. Having a fewer VMs can help identify larger blocks (subgraphs) per partition, and the LPR phase can potentially find an better quality initial value for the GPR that requires fewer steps to converge. However, running LPR on larger blocks (subgraphs) also increases the time complexity for it. As such, there may be a sweet spot of number of VMs for each graph that offers the best time to cost ratio.

5. RELATED WORK

There as been a large body of work on scalable graph processing platforms as well as algorithms for PageRank.

We can broadly classify large scale graph processing platforms as shared memory systems running on individual machines, parallel processing systems that use high-performance and accelerate hardware, and distributed computing systems that scale on commodity hardware. Shared memory graph analysis systems [33][6] offer memory efficient graph data structures to load and analyze networks, along with optimized graph kernel algorithms, but offer limited scalability. Some provide fast data structures such as indexes on top of large graphs to optimize the design of graph algorithms such a reachability queries [31]. Yet others provide domain-specific language (DSL) for easing the composition of graph analytics using richer abstractions like iterators, traversers and reducers, with a compiler that generates optimized OpenMP applications [19]. These shared memory systems however suffer from a hard memory limit.

Parallel graph processing frameworks such as the Parallel Boost Graph Library [15] provide graph APIs that are transparently mapped to MPI-based execution on HPC clusters, with some also mapping kernel algorithms like single source shortest path onto GPU accelerators [17]. Oth-

[6]http://snap.stanford.edu

ers offer a more traditional programming abstraction like BSP [12] but for massively multi-threaded systems like the Cray XMT [11]. However, access to such high end HPC systems is limited for a majority of users.

Distributed computing platforms such as commodity clusters and Clouds offer a more democratized access numerous frameworks for distributed graph processing [26,28,32,34,35] have emerged recently. These offer simple yet scalable programming abstractions such as vertex or subgraph centric models [28, 34, 35], provide a shared-memory view over distributed machines [32], or have specialized graph libraries suited for domains like machine learning [26]. Our work fits in this space, and specifically explores the use of novel abstractions such as subgraph-centric programming. However, the recency of these systems means that the suite of available distributed algorithms is limited [30] and consequently, their scalability for many graph algorithms is unknown. In this paper, we try to explore algorithmic adaptations of PageRank to effectively leverage these distributed graph programming abstractions.

Similar to GoFFish, Blogel [37] provides a block-centric graph parallel abstraction, which additionally uses a Graph Voronoi Diagram (GVD) partitioner to improve the scalability. They implement naïve PageRank and BlockRank without algorithmic optimizations, but handle the problem of PageRank loss due to sink vertices. In our algorithmic contributions, we efficiently compute the PageRank values rather than perform platform-level optimizations. Their results too show that BlockRank is much costlier than naïve PageRank, due the extra, unnecessary computations caused by poor Global PageRank initialization; this short-coming has been addressed in our work.

Significant research has gone into distributed algorithms for PageRank for partitioned graphs. Broder et al. [6] use an approach similar to BlockRank, but they compute approximate PageRank and not the actual PageRank. Their method involves computing eigenvectors, which can get costly for larger graphs. Qiuhong et al. [25] explore the notion of locality in a distributed system when running PageRank on MapReduce. They gain performance by reducing the number of MapReduce jobs per iteration, which is achieved by using subgraph as a processing unit. However, this approach resembles the naïve PageRank algorithm using a Subgraph Centric Abstraction, and MapReduce does not offer a natural way to represent iterative graph processing without costly disk I/O.

Davis et al. [8], try to estimate the global PageRank of a community by selectively crawling the non-local vertices after computing the local PageRanks to convergence in the community. Communities have a block-like structure with high internal edge degree and sparse remote edge-cut. They avoid running the PageRank algorithm on the whole graph but like us, they compute the local PageRanks is a community and then use these to estimate the Global PageRanks of the vertices in that community. We have generalized this approach to operate on dense or sparse subgraphs, and demonstrated the efficacy of this technique to diverse graphs. Kamvar et al. [21] define adaptive methods for PageRank convergence. They use the idea that several vertices reach their convergence well ahead of others, and hence they speed-up the rest of the algorithm by pruning these converged-vertices. Using this heuristic, they avoid redundantly re-computing the PageRank to offer performance gains. Such pruning can be adopted to our Subgraph Rank algorithm too as an extensions, as has been shown for performing top-k betweenness centrality using a subgraph-centric model [24].

6. DISCUSSION AND CONCLUSIONS

In this paper, we have identified deficiencies in natively mapping PageRank and BlockRank algorithms to a subgraph-centric model, and propose alternate algorithmic strategies for BlockRank. One such Subgraph Rank (SGRK) variation demonstrably outperforms the baseline and other BlockRank alternatives. Algorithmically, Subgraph Rank is better than BlockRank as it omits unnecessary input-specific optimizations; computing block-level PageRank which is negatively impacts the general case.

Considering the wide applications of PageRank, we needed a more general and better algorithm which improves the scalability and performance of PageRank. The global PageRank computation step of the BlockRank algorithm is costly since it requires more supersteps to converge than the Subgraph Rank (and even PageRank) algorithm due to the poor initialization values. Blocks of varying sizes and imbalance in the remote edges deteriorates the performance of the BlockRank algorithm, and this was motivates the need for Subgraph Rank. Subgraph Rank is a more general solution than BlockRank, and also scales better, partially due to the intelligent choice of initialization vector for the Global PageRank phase.

SGRK shows a performance gain of between $23 - 74\%$ for equivalent PageRank quality, for AMZN and WIKI graphs; its performance matches SGPR for PageRank qualities of $\Delta_{0.001}$ or better for CARN. SGRK successfully exploits the subgraph structure of the graph to offer high quality initial PageRank values from the LPR and BlockRank phases that help it to rapidly converge during the GPR phase. As such, our work shows the need for developing new algorithms or adapting existing ones to best utilize the innovative graph programming abstractions and storage models that are emerging.

Small-world networks like WIKI graph have a high clustering coefficient, and are sparsely connected with a low diameter. For such graphs the performance improves for Subgraph Rank due to the head-start they get in the local-PageRank (in-memory) phase, i.e., when well partitioned, the high clustering coefficient and sparse connectivity enables local PageRank values to be more significant. However, if the number of partition is much higher than the diameter of the graph, then the initial Global PageRank values can deteriorate the performance, and this is something we will study in the future. Similarly, for Barabasi-Albert model (Power Law) graphs, we have low diameters and the clustering decreases as the size of the graph increases. Hence with an optimal number of partitions for a given graph, the performance of Subgraph Rank will be better than PageRank. The effect of increase in the number of partitions is something that can be studied both empirically and theoretically.

As part of future work, we propose to examine the scalability of the SGRK algorithm to see if shows weak scaling with any of the topological features of the graph such as edge cuts. We also plan to investigate other graph algorithms that can benefit from subgraph centric abstractions. Heuristics like graph pruning are also viable in offering further performance gains.

Acknowledgment

We would like to thank Neel Choudhury from the DREAM:Lab, IISc for feedback on the paper and Arjun Kamath for offering insights on Giraph++. This work was supported through an AWS in Education Research Grant from Amazon.

7. REFERENCES

[1] Apache giraph. `giraph.apache.org`.

[2] The graph 500 list. http://www.graph500.org/, June 2014.

[3] T. Aittokallio and B. Schwikowski. Graph-based methods for analysing networks in cell biology. *Briefings in bioinformatics*, 7(3):243–255, 2006.

[4] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *SIGMOD*, 2011.

[5] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the web. In *World Wide Web*, 2008.

[6] A. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient pagerank approximation via graph aggregation. *Information Retrieval*, 9(2):123–138, 2006.

[7] F. Chung and W. Zhao. Pagerank and random walks on graphs. In G. Katona, A. Schrijver, T. Sznyi, and G. Sgi, editors, *Fete of Combinatorics and Computer Science*, volume 20 of *Bolyai Society Mathematical Studies*, pages 43–62. Springer Berlin Heidelberg, 2010.

[8] J. V. Davis and I. S. Dhillon. Estimating the global pagerank of web communities. In *ACM SIGKDD*, 2006.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[10] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 2012.

[11] D. Ediger and D. A. Bader. Investigating graph algorithms in the bsp model on the cray xmt. In *IPDPS Workshops*, 2013.

[12] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *JPDC*, 22(2):251–267, 1994.

[13] D. F. Gleich. Pagerank beyond the web. *CoRR*, abs/1407.5107, 2014.

[14] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, 2012.

[15] D. Gregor and A. Lumsdaine. The Parallel BGL: A generic library for distributed graph computations. In *POOSC*, 07/2005 2005.

[16] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *IPDPS*, 2014.

[17] P. Harish and P. Narayanan. Accelerating large graph algorithms on the gpu using cuda. In *HiPC*, 2007.

[18] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526. ACM, 2002.

[19] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun. Green-marl: A dsl for easy and efficient graph analysis. In *ASPLOS*, 2012.

[20] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, 2003.

[21] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of pagerank. *Linear Algebra and its Applications*, 386:51–65, 2004.

[22] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank. *Stanford University Technical Report*, 2003.

[23] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *SC*, page 29, 1995.

[24] A. G. Kumbhare, M. Frincu, C. S. Raghavendra, and V. K. Prasanna. Efficient extraction of high centrality vertices in distributed graphs. In *HPEC*, 2014.

[25] Q. Li, W. Wang, P. Wang, K. Dai, Z. Wang, Y. Wang, and W. Sun. Combination of in-memory graph computation with mapreduce: A subgraph-centric method of pagerank. In *WAIM*, 2013.

[26] 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. *PVLDB*, 5(8):716–727, Apr. 2012.

[27] L. Mainetti, L. Patrono, and A. Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *SoftCOM*, pages 1–6. IEEE, 2011.

[28] 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 *SIGMOD*, 2010.

[29] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[30] S. Salihoglu and J. Widom. Optimizing graph algorithms on pregel-like systems. In *VLDB*, 2014.

[31] S. Seufert, A. Anand, S. Bedathur, and G. Weikum. Ferrari: Flexible and efficient reachability range assignment for graph indexing. In *ICDE*, 2013.

[32] B. Shao, H. Wang, and Y. Li. Trinity: A distributed graph engine on a memory cloud. In *SIGMOD*, 2013.

[33] J. Shun and G. E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. *SIGPLAN Not.*, 48(8), 2013.

[34] Y. Simmhan, A. Kumbhare, C. Wickramaarachchi, S. Nagarkar, S. Ravi, C. Raghavendra, and V. Prasanna. Goffish: A sub-graph centric framework for large-scale graph analytics. In *EuroPar*, 2014.

[35] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson. From think like a vertex to think like a graph. *PVLDB*, 7(3), 2013.

[36] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[37] D. Yan, J. Cheng, Y. Lu, and W. Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *PVLDB*, 7(14), 2014.