Understanding Graph Computation Behavior to Enable Robust Benchmarking

Fan Yang and Andrew A. Chien*
Department of Computer Science
University of Chicago
Chicago, IL, U.S.A.
{fanyang, achien}@cs.uchicago.edu
*also Mathematics and Computer Science, Argonne National Laboratory

ABSTRACT
Graph processing is important for a growing range of applications. Current performance studies of parallel graph computation employ a large variety of algorithms and graphs. To explore their robustness, we characterize behavior variation across algorithms and graph structures at different scales. Our results show that graph computation behaviors, with up to 1000-fold variation, form a very broad space. Any inefficient exploration of this space may lead to narrow understanding and ad-hoc studies. Hence, we consider constructing an ensemble of graph computations, or graph-algorithm pairs, to most effectively explore this graph computation behavior space. We study different ensembles of parallel graph computations, and define two metrics to quantify how efficiently and completely an ensemble explores the space. Our results show that: (1) experiments limited to a single algorithm or a single graph may unfairly characterize a graph-processing system, (2) benchmarks exploring both algorithm and graph diversity can significantly improve the quality (30% more complete and 200% more efficient), but must be carefully chosen, (3) some algorithms are more useful than others in benchmarking, and (4) we can reduce the complexity (number of algorithms, graphs, runtime) while conserving the benchmarking quality.

Categories and Subject Descriptors
C.4 [**Computer Systems Organization**]: Performance of systems – *Performance attributes*.

General Terms
Algorithms, Measurement, Performance, Experimentation.

Keywords
Graph processing, Graph-algorithms, Benchmarking.

1. INTRODUCTION
Rapid growth of WWW and social networking has given rise to massive graph datasets in domains such as social networks, recommender systems, web graphs, and more. These large-scale graphs along with complex algorithms create a critical need for graph-processing systems. In response, numerous graph-processing systems have been created (Pregel [13], Giraph [1], GraphLab [5], SNAP [17], TurboGraph [9], and GraphChi [11]).

Confounding factors are wide variation in both the properties of graphs and graph algorithms. For example, graph size can differ by 6 orders of magnitude [17, 19]. Compute and communication intensity usually vary a lot across vertices and algorithms. Hence, many performance studies of graph processing system, which employ various sets of graphs and algorithms, have produced incomparable results. For example, Elser [4] finds that GraphLab outperforms Giraph, Han [8] finds that their performance are comparable, and Guo [7] produces no overall conclusion. It is difficult to gain any clear perspective on which systems are preferable. Even experts cannot claim any deep understanding of graph computation performance. So, our goal is to provide a more systematic understanding of the impact of graph and algorithm on graph-processing systems, and enable robust evaluation.

To achieve this goal, we use GraphLab as an experimental vehicle on a cluster with 768 cores, and execute 11 graph algorithms over a collection of synthetic graphs of varying size and structure. We measure key properties ranging from vertex activity to compute intensity. Our results show that not only do algorithms behave differently, but also input graphs contribute a lot to behavior variation. Understanding performance over such broad space is challenging and also necessary for robust benchmarking of graph-processing systems. So, to systematically understand the behavior of parallel graph computation, we define a vector space using fundamental graph computation properties, and also define two metrics, *spread* and *coverage*, that capture how well a set of graph-algorithm pairs explores the behavior space. With these metrics, we evaluate the benchmarking quality of different ensembles of graph-algorithm pairs, and search for best-qualified ensembles. Our study suggests that a small set of carefully chosen experiments could sample the behavior space much more efficiently, which presents an opportunity to build a high-quality benchmark suite. Specific contributions of this paper include:

(1) We conduct a series of experiments varying algorithm and graphs on a 768-core cluster, which demonstrates 1000-fold variation across 5 dimensions of graph computation behavior.

(2) We define two ensemble metrics, *spread* and *coverage*, to measure how efficiently and thoroughly an *ensemble* of experiments explores the graph computation behavior space.

(3) We demonstrate that an ensemble drawn from a single algorithm or a single graph is a poor benchmark set, and an ensemble exploring both algorithm diversity and graph diversity gives 200% better *spread* and 30% better *coverage*.

HPDC'15, June 15 - 19, 2015, Portland, OR, USA
 ISBN 978-1-4503-3550-8/15/06…$15.00
DOI: http://dx.doi.org/10.1145/2749246.2749257

(4) Our insights into the best ensembles with high spread and coverage show that some algorithms, including K-Means, Alternating Least Squares, and Triangle Counting, are more useful than other algorithms in behavior space exploration.

(5) We find that careful reduction in algorithm diversity minimizes loss of *spread* and *coverage* with further optimization possible by reducing runtime.

The remainder of the paper is organized as follows. We detail our methodology in Section 2 and show the experimental results in Section 3. In Section 4, we consider how to best design ensembles for thorough graph system evaluation. Related work is discussed in Section 5, and Section 6 summarizes and suggests future work.

2. METHODS

We employ GraphLab v2.2 [5] as our graph-processing platform. We execute programs using the synchronous mode. Each graph algorithm is executed on a variety of graphs. All the experiments were performed on the Midway system in Research Computing Center of the University of Chicago. We used up to 48 nodes, connected by a fully non-blocking FDR-10 Infiniband network. Each node has two eight-core 2.6GHz Intel Xeon E5-2670 "Sandy Bridge" processors with 32GB of memory.

2.1 Workload

We use synthetic graph generators to create graphs for each application domain. We consider both graph size and degree distribution as main parameters of our generators, which have dominant impact on graph computation behavior. Graph size is measured in number of edges (*nedges*), which determines the scale of a graph problem. Assuming all graphs in our experiments are scale-free networks [22], the fraction $P(k)$ of vertices with degree k goes for large values of k as

$$P(k) \sim k^{-\alpha}, \qquad (1)$$

Where α is a constant typically ranging *2.0-3.0* in the real world. Hence, parameters of our graph generators include both *nedges* and α. We set *nedges* to different orders of magnitude (10^5 to 10^9). We also set α to match real-world scale-free graphs (2.0 to 3.0), accepting slight variation in the number of vertices. The vertex data and edge weights are generated randomly in Gaussian distribution. To limit the number of experiments, we use 4 different sizes, and 5 different degree distributions for each domain. The range of variables is shown in Table 1.

We selected graph algorithms from distinct application domains for their widely varying behaviors.

(1) *Graph Analytics (GA)* focuses on data mining, especially relationships from large graphs. We choose six algorithms, including Connected Components (CC), K-Core decomposition (KC), Triangle Counting (TC), Single-Source Shortest Path (SSSP), PageRank (PR), and Approximate Diameter (AD).

(2) *Clustering* classifies objects based on similarity. We choose K-Means (KM) that partitions n vertices into k clusters such that each vertex belongs to the cluster with the nearest mean [21].

Table 1. Graph Feature Variables

Domains	Algorithms	Variables	Values
Graph Analytics	CC, TC, KC, SSSP, PR, AD	*nedges*	10^6, 10^7, 10^8, 10^9
		α	2.0, 2.25, 2.5, 2.75, 3.0
Clustering	KM	*nedges*	10^6, 10^7, 10^8, 10^9
		α	2.0, 2.25, 2.5, 2.75, 3.0
Collaborative Filtering	ALS, NMF, SGD, SVD	*nedges*	10^5, 10^6, 10^7, 10^8
		α	2.0, 2.25, 2.5, 2.75, 3.0

(3) *Collaborative Filtering (CF)* is used by recommender systems to predict the rating that user would give to an item, e.g. a movie or a book [20]. A general method for CF is to learn user-factor vectors and item-factor vectors through matrix factorization. We select four algorithms: Alternating Least Squares method (ALS), Non-negative Matrix Factorization (NMF), Stochastic Gradient Descent (SGD) and Singular Value Decomposition (SVD).

2.2 Computation Model

Our algorithms are implemented in the Gather-Apply-Scatter (GAS) model [5, 13]. Graph computation is expressed in a vertex-centric fashion. Each vertex can be active or inactive. Only active vertices can perform computation. In each iteration, vertices receive data through adjacent edges, called *edge reads*; then vertices perform user-defined computation and update their values, which are called *vertex updates*; finally vertices send signals to activate neighbors, which are called *messages* in this paper. In our experiments, all tested algorithms converge in finite number of iterations, except NMF and SGD. So we set a maximum 20 iterations for them.

There are also other computation models used in current graph-processing systems [14, 18], but the fundamental behavior of graph computation is conserved, such as transferring information through edges, performing computation on an independent unit (vertex/edge), and activations.

2.3 Performance Metrics

To characterize the fundamental behavior of parallel graph computations, we define five performance metrics.

On one hand, in order to capture the behavior variation over time, we define *active fraction*, which is the ratio of active vertices to all vertices in a single iteration that directly shows the activity of vertices across the whole lifecycle. For many algorithms, active fraction varies over time.

On the other hand, in order to capture the overall behavior, we define *{UPDT, WORK, EREAD, MSG}*, which reflect the compute intensity and communication intensity of a graph computation.

(1) *UPDT*: average number of vertex updates per iteration

(2) *WORK*: average CPU time for computing per iteration.

(3) *EREAD*: average number of edge reads per iteration

(4) *MSG*: average number of messages per iteration

Moreover, we divide each of these four metrics by the number of edges to capture the per-edge behavior. We also normalize these metrics to [0.0, 1.0] for highlighting the relative difference, rather than absolute values.

3. BEHAVIOR VARIATION IN PARALLEL GRAPH COMPUTATION

In this section, we present our experimental results. Due to paper length constraints, we present a selection of results. The full results are available in our technical report [23].

In our experiments, all algorithms have a characteristic shape of active fraction that varies significantly across algorithms (see Figure 1). The convergence rate also differs a lot across domains, by up to three orders of magnitude (TC vs. DD). Figure 2 shows

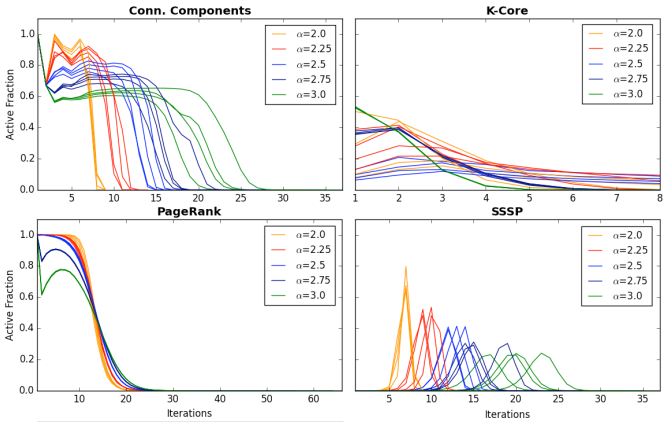

Figure 1. Active fraction for 4 algorithms x 20 graphs. Curves of same color correspond to graphs with same *α* but different size.

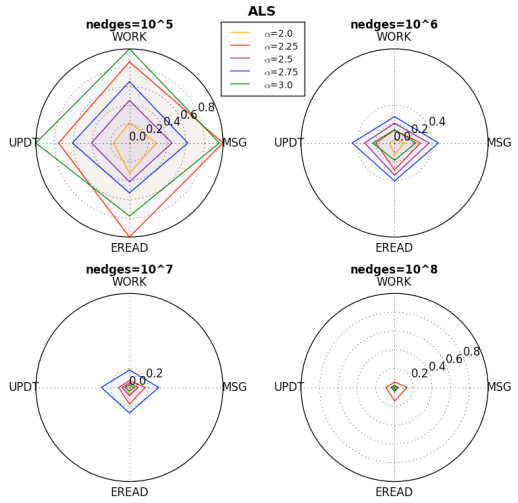

Figure 2. ALS metric values over 20 graphs.

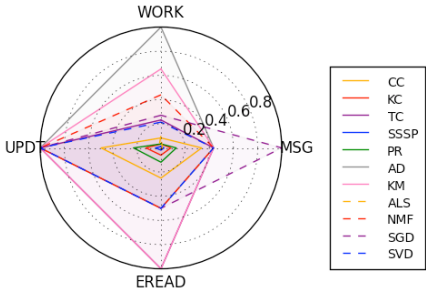

Figure 3. Metric values of 11 algorithms over same graph.

an interesting benchmark from our collection, ALS. Its behavior strongly depends on graph size and degree distribution. We observe high variation in all 4 metrics. Figure 3 shows the different shapes of 4 performance metrics for all algorithms. The values of all 4 metrics are much smaller in ALS, SSSP, KC, PR and LBP than in other algorithms. AD requires the most work for updating vertices, KM requires the most data transferring, and SGD requires the most message transferring.

Overall, graph computation behavior exhibits a wide diversity across algorithms and graphs, forming a very broad space. Any single experiment only samples a small part of that space, leading to ad-hoc performance studies that are unable to systematically understand the performance of graph-processing systems.

4. EXPLORING A BEHAVIOR SPACE

To systematically understand such wide variety of parallel graph computations, we define a behavior space that covers the fundamental properties we have measured in Section 3 as follows:

$$Behavior\ (GC_i) = <UPDT,\ WORK,\ EREAD,\ MSG> \quad (2)$$

Where GC_i is a graph computation represented as a *graph-algorithm* pair. While our behavior space is designed to capture fundamental graph computation behavior, doing so optimally is an open research challenge; we define only one[1] vector performance space, and evaluate its efficacy. Possible uses of our graph computation behavior characterization include basic algorithm analysis, algorithm comparison, performance analysis, graph computation optimization, performance prediction, system benchmarking, and even system design. Here we consider *how to use the behavior performance space to design efficient, high-quality benchmarking of graph-processing systems*.

We define an ensemble as a set of graph computations, $\{GC_1, GC_2, ...\}$ to model a benchmark suite and characterize any set of experiments. More formally we define:

$$Ensemble_k = \{GC_1, GC_2, ..., GC_N\} \quad (3)$$

Given this definition, then we would like to describe how well an ensemble characterizes graph computation behavior. To this end, we define two ensemble metrics, *spread* and *coverage*, that quantitatively characterize the quality of an ensemble to efficiently and thoroughly explore graph computation behavior.

Spread of an ensemble is defined as the mean pairwise distance between the *Behavior* vectors in an ensemble (see below).

$$Spread(Ensemble_k) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} d(Behavior(GC_i), Behavior(GC_j))}{N(N-1)} \quad (4)$$

Where *d()* is the Euclidean distance between two graph computation *Behavior* vectors. Intuitively, spread represents a form of "dispersion" of an ensemble. If the ensemble's elements are tightly clustered, then it will be low. If ensemble members are uniformly dispersed, it will be high.

Coverage of an ensemble is defined as average minimum distance from all points in the space to the nearest point in the ensemble.

$$Coverage(Ensemble_k) = \frac{N_S}{\sum_{i=1}^{N_S} \min_{k=1...N} \{d(Sample_i, Behavior(GC_k))\}} \quad (5)$$

Where sample points, $Sample_i$ ($i=1...N_S$), are taken randomly and uniformly throughout the space to compute the coverage. (We set N_S, the number of sample points to 1 million.) Intuitively, the notion of good coverage is the idea that no matter where a behavior falls in the space, it will be close to a point in the ensemble. For an ensemble, a high coverage indicates that we have sampled the behavior space thoroughly.

So, in order to construct an ensemble of graph computations with best benchmarking quality, for a given ensemble size, the goal is to maximize coverage. For a given coverage, the goal is to maximize spread to improve efficiency.

[1] Ours is the first formally defined behavior space for graph computations of which we are aware.

4.1 Ensembles of Single Algorithms/Graphs

Many studies assess graph computation performance with a single algorithm or a single graph, which seems to be a simple choice for benchmarking. So first, we consider the question: *How well can an ensemble using such a single origin explore the behavior space?*

A *single-algorithm ensemble* consists of experiments with one algorithm over a variety of graphs, and a *single-graph ensemble* consists of experiments over one graph with a variety of algorithms. We exhaustively test, and pick the best ensemble for both types and for each given ensemble size (5, 10, 15 and 20). In Figure 4 and 5, for both single-algorithm and single-graph ensembles, spread decreases steadily, and coverage increases very slowly. This indicates that additional runs of a single algorithm or a single graph tend to be clustered, and do not spread out to cover a larger behavior space. The achieved spread of single-graph ensembles is significantly higher than with single algorithms, which indicates that graph appears to be a more important factor in behavior variation than algorithm.

To understand the quality of the achieved spread and coverage, we also plot an empirical upper bound for each given ensemble size. These are computed assuming ensemble members uniformly and maximally distributed in the behavior space. Both the spread and coverage achieved by single-algorithm and single-graph ensembles fall well below our empirical upper bound.

Implications: If restricted to a single algorithm or a single graph, only limited graph computation behavior can be evoked. Adding more runs based on either the same algorithm or the same graph, generally leads to similar graph computation behavior. If we look at single-algorithm ensembles, they exhibit even less behavior variation than single-graph ensembles. In short, particular graph algorithms often behave similarly across graph structures and scales. Ensembles created with these properties (as used in numerous published graph computation system performance studies) risk unfair comparisons or incomplete results, and potentially erroneous conclusions.

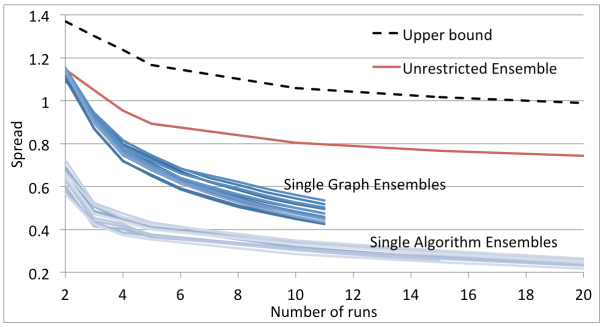

Figure 4. Spread for 3 types of ensembles.

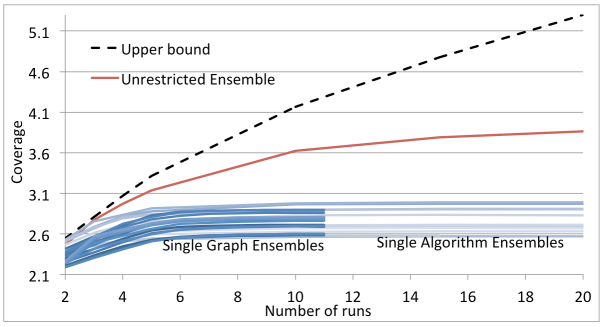

Figure 5. Coverage for 3 types of ensembles.

4.2 Unrestricted Diverse Ensembles

Given our broad exploration of graph computation behavior, we consider efficient and effective ensemble design retrospectively. Then we ask the question — *What are the ensembles we could have constructed? That is, the ensembles that would explore the behavior space most efficiently?* Intuitively, the best ensemble corresponds to the most efficient benchmark suite. We search all our 215 runs for the ensembles with best spread and coverage without any restriction on algorithms and graphs.

Allowed unrestricted choice across multiple algorithms and graphs, it is possible to sample the space much more efficiently and completely. Red curves in Figure 4 and 5 represent the achievable spread and coverage with unrestricted ensembles. Spread starts high (~1.2), and declines slowly to 0.75 for 20 members. Overall, there's a clear benefit in drawing richly from both algorithm and graph diversity, with as much as a three-fold greater spread. Considering coverage, ensembles that draw on both algorithm and graph diversity have clear advantage, delivering 30% better coverage than single-algorithm ensembles. The unrestricted ensembles achieve coverage that is significantly higher at as few as 5 runs, and the advantage grows even at 20 runs, achieving 3.9 overall.

Implications: Both algorithm diversity and graph diversity contribute to the behavior variation of graph computations. Hence, by exploiting both, we can identify multiple graph-algorithm pairs that exhibit diverse behaviors. We can use this information to construct an efficient and representative benchmark suite by carefully selecting graph-algorithm pairs that exhibit very different behavior and thoroughly sample the whole behavior space. In this way, this high-quality benchmark suite can exercise a broad graph computation behavior and systematically evaluate the performance of a graph processing system.

4.3 Understanding Diversity

While unrestricted diverse ensembles give much better results, we still want to understand *What aspects of diversity in algorithms and graphs contribute to this improvement*? Because algorithm usually dominants benchmarking complexity, we look into the representation of algorithms in the best ensembles (see Table 2).

Interestingly, some algorithms are consistently represented in the best ensembles, such as Alternating Least Squares for best spread, and K-Means for best coverage. The best ensembles are complicated – involving large numbers of algorithms and graphs. For example, the best 5-member ensemble for *spread* includes 4 algorithms and 5 different graphs. The best 5-member ensemble

Table 2. Ensembles Achieving Best Spread and Coverage (for ⩾10, algorithms only).

Type	Size	Runs (Algorithm[, Graph Size, α])
Best spread	5	<ALS, 10^5, 3.0>, <SGD, 10^8, 2.0>, <TC, 10^9, 2.0>, <SSSP, 10^9, 3.0>, <ALS, 10^5, 2.75>
	10	ALS, SGD, TC, SSSP, ALS, TC, SGD, ALS, KM, SVD
	15	SSSP, ALS, KM, SGD, ALS, TC, SGD, ALS, TC, SSSP, ALS, SGD, TC, SVD, ALS
	20	SSSP, ALS, TC, SGD, ALS, TC, SGD, ALS, KM, SSSP, ALS, SGD, KM, SVD, ALS, TC, SGD, ALS, SGD, TC
Best coverage	5	<TC, 10^6, 2.5>, <KM, 10^6, 2.25>, <AD, 10^7, 3.0>, <ALS, 10^8, 2.0>, <KC, 10^6, 2.5>
	10	AD, SVD, KM, ALS, TC, KC, KM, ALS, KM, NMF
	15	KM, NMF, ALS, AD, SVD, KC, KM, ALS, KM, KM, SVD, PR, ALS, TC, NMF
	20	AD, SVD, KM, ALS, TC, KC, KM, ALS, KM, SGD, NMF, KM, ALS, NMF, PR, TC, NMF, SSSP, ALS, AD

for *coverage* includes 5 algorithms and 4 graphs. At the level of 10-member ensembles, the complexity already exceeds that of most comparative graph performance studies (6 algorithms for spread and 7 for coverage) [4, 7, 8, 9,15].

To reliably assess "diversity contribution" of an algorithm, we would like to minimize shadowing effects. That is, considering only the best ensembles, a particular algorithm that is useful for *spread* or *coverage* might be shadowed by others that are slightly better, but when they're removed, really contribute greatly to diversity. We consider the question: *Which algorithms contribute most often to the best ensembles for spread and coverage?* To minimize shadowing, we expand our consideration of the best ensemble of size n to the 100 best ensembles of size n for each – spread and coverage. Within the 100 best ensembles, we use the frequency of appearance of each algorithm as an indication of contribution to diversity (see Figure 6 and 7).

These results demonstrate that not all algorithms contribute significantly to a good spread or coverage. For example, K-Means, Alternating Least Squares, and Triangle Counting among our suite contribute to efficient and thorough behavior space exploration.

Implications: not all graph algorithms are equally useful in high-quality benchmarking. Some algorithms exhibit a greater diversity of behaviors across graphs (e.g. ALS appears several times in one best-spread ensemble), while others exhibit behaviors varying little and thus characterize a narrow class of behaviors. On the other hand, some algorithms explore a unique region of the behavior space, making them valuable and difficult to replace in a thorough benchmarking (e.g. KM appears in almost all best-coverage ensembles). Therefore, these "special" algorithms should be the first choice in the construction of a high-quality benchmark suite.

4.4 Reducing Ensemble Complexity

Because the best ensembles require complex combinations of algorithms and graphs, it is worthwhile to consider simpler combinations to reduce benchmarking complexity. So the next question is: *How does restricting our ensemble complexity impact achievable spread and coverage?* Here we consider three dimensions of constraints, limited algorithms, graphs, and runtime (see Figure 8 and 9).

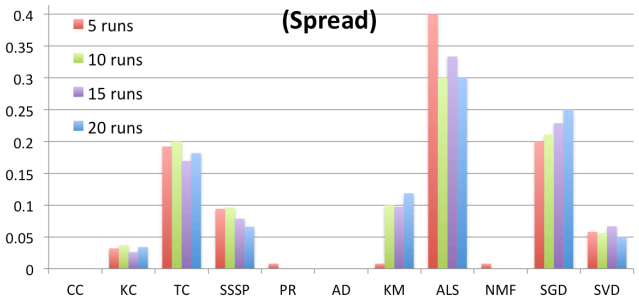

Figure 6. Frequency of algorithms in Top100 sets (spread).

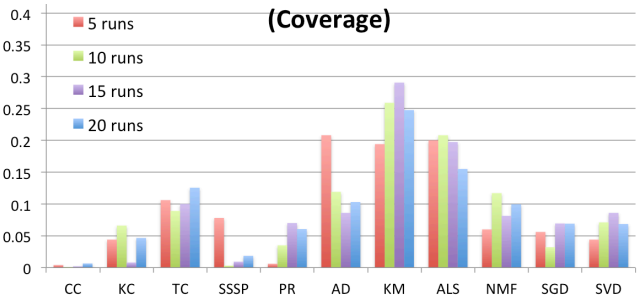

Figure 7. Frequency of algorithms in Top100 sets (coverage).

First, we limit ensembles to three algorithms, selecting those that contribute most to both spread and coverage, including KM, ALS and TC. The algorithm-limited suites maintain a high spread, and a slight advantage over single algorithms.

Second, we limit ensembles to three graphs. The best ensembles use the graphs of size 10^7, 10^8 and 10^9 with $\alpha = 2.0$. The results indicate that limiting the number of graphs decreases spread rapidly and produces poor coverage – even lower than single algorithms like KC and CC.

Third, we consider shortening runtime. Some of our algorithms, including AD, KM, NMF, SGD, and SVD, have constant, repetitive behavior (i.e. a constant active fraction = 1.0). Their runs could be shortened, so ensembles using these 5 algorithms, can much more efficiently probe the behavior space. Because of the variety of these repetitive algorithms, these runtime-constrained suites still achieve high spread and coverage.

Implications: 1) A small collection of "special" algorithms can exercise a broad computation behavior. By employing these algorithms, we can reduce the number of algorithms used in benchmarking with minimal loss of quality. 2) Restricting graph selection significantly decreases benchmarking quality, because many algorithms exhibit similar behavior on the same graph structure. 3) Some algorithms that have unvaried behavior also have a high benchmarking quality, which enables us to further reduce runtime. Overall, by choosing these constrained runs to sample the behavior space efficiently, we can benchmark systems with minimum computational effort.

5. DISCUSSION AND RELATED WORK

General benchmark suites such as SPEC [16], NPB [3] and HPCC [10] employ a wide collection of programs, often with single inputs of varied sizes. Though these benchmarks are carefully selected, none are designed to systematically characterize performance, and efficient sampling of the full behavior space is beyond reach. Closer, there are also graph-oriented benchmarking efforts, including Graph 500 [6], LDBC [12], and LinkBench [2].

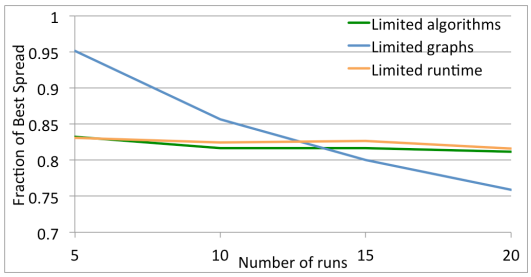

Figure 8. Spread for limited algorithms, graphs, runtime

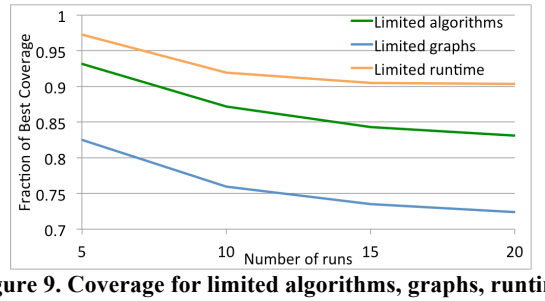

Figure 9. Coverage for limited algorithms, graphs, runtime

However, Graph 500 uses only a single program, on a single graph typically. LDBC is only emerging, and LinkBench provides general infrastructure, but not a specific benchmark.

Comparative graph processing systems studies are numerous. We summarize a few of the most complete efforts here. M. Han et al. [8] compare Giraph, GPS, Mizan, and GraphLab with 4 simple benchmarks and 5 real graphs. S. Salihoglu et al. [15] made similar study on Pregel-like systems. But neither Han nor Salihoglu give any clear rationale for benchmark selection, and no claims of thoroughness of behavior space exploration are made.

B. Elser et al. [4] compare 5 platforms using K-core program and 7 graphs. W. Han et al. [9] compare TurboGraph and GraphChi using simple benchmarks (PageRank, Connected Components, etc.) and 3 graphs. Our results show that, neither a single complex algorithm such as K-Core, nor several simple benchmarks such as PR and CC explore behavior thoroughly. For W. Han's experiments, the small number of graphs also narrows their study.

Y. Guo et al. [7] propose a comprehensive benchmarking method by exploring dataset, algorithm and platform diversity. They use 5 algorithms and 7 graphs to compare 6 platforms. Guo's work is perhaps closest to ours. They recognize the need to explore fully, and claim to do so. However, they present no clear framework of formal metric for assessing the thoroughness of exploration. In contrast, we have formulated a space and clear metrics for assessing thoroughness.

Our study suggests how to construct ensembles of graph computations that thoroughly explores the behavior space of graph computations. Beyond thoroughness, we describe how to further optimize an ensemble for efficiency and simplicity – limiting graphs, algorithms, etc. These ensembles can of course be used for benchmarking, and by exercising graph computing system behavior broadly, such a benchmark suite promises to give a comprehensive view of the performance of graph-processing systems, as well as an objective comparison between systems.

To the best of our knowledge, our study is the first and the only one that quantitatively characterize the behavior of parallel graph computation and to propose an evaluation methodology for systematic benchmarking.

6. SUMMARY AND FUTURE WORK

We have studied the behavior of a variety of graph algorithms (graph analytics, clustering, collaborative filtering, etc.) on a diverse collection of graphs (varying size and degree distribution). Our results characterize the variation of behavior across graph structures, application, domains, and graph algorithms. Given the broad range of graph computation behavior, we consider how well an ensemble of graph computations (algorithms and graphs) explores this space, in terms of spread and coverage. Since ad-hoc sets are unlikely to fairly characterize a graph-processing system, our study gives suggestions on systematically choosing a small set of algorithms and graphs to exercise broad computational behavior. We find that ensembles over single algorithms and single graphs are inefficient in sampling the behavior space, while a set of carefully selected runs exploring both algorithm diversity and graph diversity give significantly better results. We propose more efficient ensembles by considering constraints such as limited algorithms, graphs, and even runtime.

With understanding of how ensembles of experiments sample the behavior space, more ambitious goals are within reach. Can we design optimal ensembles? Can we model precisely a graph computation's behavior, and predict its performance?

7. ACKNOWLEDGEMENT

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, Department of Energy, under Award DE- SC0008603, as well as generous donations from HP Corporation and Agilent/Keysight Corporation. This work was completed in part with resources provided by: the University of Chicago Research Computing Center.

8. REFERENCES

[1] Apache, Apache Giraph, http://incubator.apache.org/giraph/.

[2] T.Armstrong et al. LinkBench: a Database Benchmark Based on Facebook Social Graph, *SIGMOD'13*, June 22-27, 2013.

[3] D. Bailey et al. THE NAS PARALLEL BENCHMARKS, Techical Report, NASA Ames Research Center, Jan 1991.

[4] B. Elser et al. An evaluation study of BigData frameworks for graph processing, *Big Data 2013*, pp. 60-67, 2013.

[5] J. E. Gonzalez et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs, *OSDI '12*, 2012.

[6] Graph500, Graph500 Benchmark, http://www.graph500.org/.

[7] Y. Guo et al. How Well do Graph-Processing Platforms Perform? An Empirical Performance Evaluation and Analysis, *IPDPS'14*, 2014.

[8] M.K. Han, Daudjee, et al. An Experimental Comparison of Pregel-like Graph Processing System, *the VLDB Endowment*, Vol. 7, No. 12, 2014.

[9] W. Han, et al. TurboGraph: A Fast Parallel Graph Engine Handling Billion-scale Graphs in a Single PC, *KDD'13*, 2013.

[10] HPCC, HPC Challenge Benchmark, http://en.wikipedia.org/wiki/HPC_Challenge_Benchmark.

[11] A. Kyrola, G. Blelloch, and C. Guestrin, "GraphChi: Large-scale Graph Computation on Just a PC," in OSDI, 2012.

[12] LDBC Organization, LDBC, http://www.ldbc.eu/.

[13] G. Malewicz, M. Austern, et al. Pregel: A system for large-scale graph processing, *SIGMOD'10*, pp. 135-146, 2010.

[14] A. Roy, et al. X-Stream: Edge-centric Graph Processing using Streaming Partitions, *SOSP'13*, Nov03-06, 2013.

[15] S. Salihoglu and J. Widom, Optimizing Graph Algorithms on Pregel-like Systems, Technical Report, Stanford, 2013.

[16] SPEC, SPEC CPU2006 benchmark suite, https://www.spec.org/cpu2006/Docs/readme1st.html.

[17] Stanford University, The Stanford Network Analysis Project (SNAP), http://snap.stanford.edu/index.html.

[18] Y. Tian, et al. From Think Like a Vertex to Think Like a Graph, *the VLDB Endowment*, Vol. 7, No. 3, 2014.

[19] Web Data Commons, Web Data Commons — Hyperlink Graphs. http://webdatacommons.org/hyperlinkgraph/.

[20] Wikipedia, Collaborative filtering, http://en.wikipedia.org/wiki/Collaborative_filtering.

[21] Wikipedia, K-means clustering, http://en.wikipedia.org/wiki/K-means_clustering.

[22] Wikipedia, Scale-free network, http://en.wikipedia.org/wiki/Scale-free_network.

[23] F. Yang and A. Chien, Understanding Graph Computation Behavior to Enable Robust Benchmarking, Technical Report, UChicago Computer Science, 2015.