

RESEARCH

Open Access



BC tree-based spectral sampling for big complex network visualization

Jingming Hu¹, Tuan Tran Chu², Seok-Hee Hong^{1*}, Jialu Chen¹, Amyra Meidiana¹, Marnijati Torkel¹, Peter Eades¹ and Kwan-Liu Ma³

*Correspondence:
seokhee.hong@sydney.edu.au

¹ School of Computer Science, University of Sydney, Sydney, Australia
Full list of author information is available at the end of the article

Abstract

Graph sampling methods have been used to reduce the size and complexity of big complex networks for graph mining and visualization. However, existing graph sampling methods often fail to preserve the connectivity and important structures of the original graph. This paper introduces a new divide and conquer approach to spectral graph sampling based on graph connectivity, called the BC Tree (i.e., decomposition of a connected graph into biconnected components) and spectral sparsification. Specifically, we present two methods, spectral vertex sampling *BC_SV* and spectral edge sampling *BC_SS* by computing effective resistance values of vertices and edges for each connected component. Furthermore, we present *DBC_SS* and *DBC_GD*, graph connectivity-based distributed algorithms for spectral sparsification and graph drawing respectively, aiming to further improve the runtime efficiency of spectral sparsification and graph drawing by integrating connectivity-based graph decomposition and distributed computing. Experimental results demonstrate that *BC_SV* and *BC_SS* are significantly faster than previous spectral graph sampling methods while preserving the same sampling quality. *DBC_SS* and *DBC_GD* obtain further significant runtime improvement over sequential approaches, and *DBC_GD* further achieves significant improvements in quality metrics over sequential graph drawing layouts.

Keywords: Graph sampling, Spectral sparsification, Connectivity

Introduction

Big complex networks are abundant in many application domains, such as social networks and systems biology. Examples include social networking websites, protein-protein interaction networks, biochemical pathways and web graphs. However, good visualization of big complex networks is challenging due to scalability and complexity. For example, visualizations of big complex networks often produce hairball-like visualization, making it difficult to understand the structure of the graphs.

Graph sampling methods have been widely used to reduce the size of graphs in graph mining (Hu and Lau 2013; Leskovec and Faloutsos 2006). Popular graph sampling methods include Random Vertex sampling, Random Edge sampling and Random Walk. However, previous work based on random sampling methods often fails to

preserve the connectivity and important structures of the original graph, in particular for visualization (Wu et al. 2017).

Spectral sparsification is a technique to reduce the number of edges in a graph while retaining its structural properties (Spielman and Teng 2011). More specifically, it is a sampling method which uses the *effective resistance* values of edges, which is closely related to the commute distances of graphs. However, computing effective resistance values of edges is rather complicated, which can be very slow for big graphs (Eades et al. 2017b).

Another method to address scalability issues in computing is *distributed computing*. With the advent of technology, cloud computing services are becoming much cheaper and convenient for anyone to use, leading to the widespread use of distributed computing on the cloud. Recent works have used cloud computing for distributed graph drawing (Arleo et al. 2019). However, communication overhead still poses a problem, impacting the efficiency gains.

This paper introduces divide and conquer algorithms for spectral sparsification, based on the graph *connectivity*, called the BC (Block Cut-vertex) tree decomposition, which represents the decomposition of a graph into biconnected components. More specifically, the main idea is to divide a big complex network into biconnected components, and then compute the spectral sparsification for each biconnected component in parallel to reduce the runtime as well as to maintain the graph connectivity. Namely, the effective resistance values of edges are computed for each biconnected component, as an approximation of the effective resistance values of the original graph. We also introduce a BC tree-based distributed framework for spectral sparsification and graph drawing, designed for use in a cloud-based distributed computing environment.

The main contributions of this paper are summarized as follows:

- 1 We present two new variations of spectral sparsification based on connectivity, *spectral edge sampling BC_SS* (BC Spectral Sampling) and *spectral vertex sampling BC_SV* (BC Spectral Vertex). Spectral edge sampling mainly sparsifies the edge set, while the spectral vertex sampling focuses on reducing the size of the vertex set.
- 2 We present *DBC_SS* (Distributed BC Spectral Sampling), a distributed algorithm for *spectral sparsification* integrating BC tree decomposition. Effective resistance values of edges are computed independently on each biconnected component in parallel to improve the runtime efficiency of spectral sparsification.
- 3 We present *DBC_GD* (Distributed BC Graph Drawing), a distributed *graph drawing* algorithm integrating BC tree decomposition. Using a *radial tree drawing* of the BC tree of a graph as an initial layout, biconnected components are drawn independently in parallel, and each vertex of the BC tree is replaced with the drawing of the biconnected component it represents to obtain a drawing of the whole graph.
- 4 We implement and evaluate *BC_SS* and *BC_SV* in comparison to sequential spectral sparsification. Experimental results demonstrate that our *BC_SS* and *BC_SV* methods are significantly faster than the original *SS* (Spectral Sparsification) (Eades et al. 2017b) and *SV* (Spectral Vertex sampling) (Hu et al. 2019) at about 72% faster on average, while preserving the same sampling quality, using a comparison of the effec-

tive resistance values and rankings of edges (resp., vertices), sampling quality metrics, graph similarity, and visual comparison.

- 5 We implement and evaluate *DBC_SS* in comparison to sequential effective resistance value computation *SS*. Our experiments show that *DBC_SS* obtains runtime improvements over *SS* by 27% and 47% when running on 2 servers and 5 servers respectively.
- 6 We implement and evaluate *DBC_GD* with three variations based on the layout used to draw each biconnected component: *DBC_FR* (Fruchterman-Reingold), *DBC_SM* (Stress Majorization), and *DBC_FM3* (FM^3). Compared to the sequential layouts, *DBC_FR* is 26 times and 52 times faster than FR when running on 2 servers and 5 servers respectively, *DBC_SM* is 31 times and 71 times faster than SM when running on 2 servers and 5 servers respectively, and *DBC_FM3* is 2 times and 4 times faster than FM^3 when running on 2 servers and 5 servers respectively. *DBC_GD* also surprisingly obtains better quality metrics than the original layouts.

Related work

Graph sampling

Graph sampling methods have been extensively studied in graph mining to reduce the size of big complex graphs. Consequently, many stochastic sampling methods are available (Hu and Lau 2013; Leskovec and Faloutsos 2006). For example, most popular stochastic sampling include Random Vertex sampling and Random Edge sampling. However, it was shown that random sampling methods often fail to preserve connectivity and important structure in the original graph, in particular for visualization (Wu et al. 2017).

Newer sampling methods have attempted to improve upon stochastic sampling, such as by using topology-based decomposition to preserve connectivity (Hong et al. 2018) or by preserving edges representing significant deviations in the weight distribution of a weighted graph (Serrano et al. 2009).

Spectral sparsification

Spectral graph theory is concerned with the eigenvalues and eigenvectors of matrices associated with graphs (Spielman 2007). The spectrum of a graph is the list of eigenvalues of its *Laplacian matrix* L , which is defined as $L = D - A$ where D is the *degree matrix* and A is the *adjacency matrix*. The spectrum of a graph is closely related to important structural properties such as connectivity, clustering, stress, and commute distance. Graph spectrum has been used to draw aesthetics-focused graph layouts with fast runtime (Koren 2003) as well as to support various types of graph analysis (Forman et al. 2005; Gera et al. 2018; Galimberti et al. 2019).

A *spectral sparsifier* is a subgraph whose Laplacian quadratic form is approximately the same as the original on all real vector inputs. Every n -vertex graph has a spectral approximation with $O(n \log n)$ edges (Spielman and Teng 2011).

Spectral sparsification selects edges based on their *effective resistance*. Modelling a graph as an electrical network, the effective resistance of the edge is defined as the voltage drop across the edge and its value is equivalent to the probability of the edge to be

included in a random spanning tree of the graph (Spielman and Srivastava 2011). However, computing effective resistance values of edges is quite complicated and can be very slow for big graphs (Eades et al. 2017b).

Spectral sparsification-based sampling has been shown to obtain superior sampling quality compared to stochastic sampling (Eades et al. 2017b; Hu et al. 2019), as well as compared to sampling based on graph centrality (Hong and Lu 2020).

BC (Block Cut-vertex) tree decomposition

The *BC tree* represents the tree decomposition of a connected graph G into biconnected components, which can be computed in linear time (Hopcroft and Tarjan 1973). A biconnected graph is a graph without a cut vertex, that is a vertex whose removal disconnects the graph; a biconnected component is a maximal biconnected subgraph. There are two types of nodes in the BC tree T ; a cut vertex c and a biconnected component B . A *cut vertex* is a vertex whose removal from the graph makes the resulting graph disconnected. A *biconnected component* (or *block*) is a maximal biconnected subgraph.

Graph sampling quality metrics

There are a number of quality metrics for graph sampling (Hu and Lau 2013). For our experiment, we use the following most popular quality metrics:

- *Degree Correlation Associativity (Degree)*: a basic structural metric, which computes the likelihood that vertices link to other vertices of similar degree, called positive degree correlation (Newman 2003).
- *Closeness Centrality (Closeness)*: a centrality measure of a vertex in a graph, which sums the length of all shortest paths between the vertex and all the other vertices in the graph (Freeman 1978).
- *Average Neighbor Degree (AND)*: the measure of the average degree of the neighbors of each vertex (Barrat et al. 2004).
- *Clustering Coefficient (CC)*: measures the degree of vertices which tend to cluster together (Saramki et al. 2007).

Graph drawing algorithms

One of the most popular graph drawing algorithms is the *force-directed* or *spring* algorithm, which models a graph as a system with attraction forces between neighboring vertices and repulsion forces between all pairs of vertices (Eades 1984). Although spring algorithms are able to produce high-quality graph layouts, traditional spring algorithms do not scale well to larger graphs due to the repulsion force computation taking $O(n^2)$ runtime.

Multi-level algorithms have been developed which improves the runtime to $O(n \log n)$ in practice using a *coarsening* and *refinement* technique. Here, vertices are gradually clustered in the coarsening phase until a simple graph that can be drawn easily is obtained, and in the refinement step, the drawing of the “coarse” graph is then used as an initial layout for the next level with more vertices. An example is the FM³ (Fast Multipole Multilevel Method) algorithm (Hachul and Jünger 2004).

Other layout types include *stress-based* layouts, which utilize the stress function from MDS (multi-dimensional scaling). These methods compute a layout by minimizing an adapted stress function, which is lower when the geometric distances between vertices are proportional to their graph theoretic distances (Gansner et al. 2004).

For drawing trees, specific algorithms have been introduced. An example is the radial tree drawing algorithm (Eades 1991). Starting at a root vertex, subtrees are assigned a wedge around the root with width proportional to its size, and further subtrees are recursively assigned a sub-area of its parent's wedge.

BC tree-based spectral graph sampling

In this section, we introduce a new divide and conquer algorithm for spectral sparsification, by tightly integrating the BC tree decomposition, aiming to reduce the runtime for computing the effective resistance values as well as to maintain the graph connectivity. We present two variations, called *BC_SS* (for spectral sparsification of *edges*) and *BC_SV* (for spectral sampling of *vertices*).

More specifically, we divide a big complex graph into a set of biconnected components, and then compute the spectral sparsification (i.e., effective resistance values) for each biconnected component in parallel. Namely, the *effective resistance* values of the edges are computed for each biconnected component, as a fast *approximation* of the effective resistance values of the original graph.

Let $G = (V, E)$ be a graph with a vertex set V ($n = |V|$) and an edge set E ($m = |E|$). The *adjacency matrix* of an n -vertex graph G is the $n \times n$ matrix A , indexed by V , such that $A_{uv} = 1$ if $(u, v) \in E$ and $A_{uv} = 0$ otherwise. The *degree matrix* D of G is the diagonal matrix where D_{uu} is the degree of vertex u . The *Laplacian matrix* of G is $L = D - A$. The *spectrum* of G is the eigenvalues of L , $\lambda_1, \lambda_2, \dots, \lambda_n$. Suppose that we regard a graph G as an electrical network where each edge e is a $1-\Omega$ resistor, and a current is applied. The effective resistance $r(e)$ of an edge e is the voltage drop over the edge e , see (Spielman and Teng 2011).

Algorithm *BC_SS*

Let $G = (V, E)$ be a connected graph with a vertex set V and an edge set E , and let $G_i, i = 1, \dots, k$, denote biconnected components of G .

The *BC_SS* algorithm first computes the BC tree decomposition, and then adds the cut vertices and their incident edges to the spectral sparsification G' of G . This is due to the fact that the cut vertices play important roles in preserving the connectivity of the graph as well as in social network analysis, such as brokers or important actors connecting two different communities.

Next, it computes a spectral sparsification G'_i for each biconnected component $G_i, i = 1, \dots, k$ of G . Specifically, for each component G_i , we compute the effective resistance values $r(e)$ of the edges, and then sample the edges with the largest effective resistance values. Finally, it merges $G'_i, i = 1, \dots, k$ to obtain the spectral sparsification G' of G . The *BC_SS* algorithm is described as follows:

Algorithm: *BC_SS*

1. *Partitioning*: Divide a connected graph G into biconnected components, G_i , $i = 1, \dots, k$.
2. *Cut vertices*: Add the cut vertices and their incident edges to the spectral sparsification G' of G .
3. *Spectral sparsification*: For each component G_i , compute a spectral sparsification G'_i of G_i . Specifically, compute the effective resistance values $r(e)$ of the edges, and then sample the edges with the largest effective resistance values.
4. *Aggregation*: Merge all G'_i of G_i to compute the spectral sparsification G' of the original graph G .

Algorithm BC_SV

The *BC_SV* algorithm is a divide and conquer algorithm that uses spectral sampling of vertices (Hu et al. 2019): i.e., adapt the spectral sparsification approach, by sampling *vertices* rather than edges. More specifically, we define an *effective resistance* value $r(v)$ for each vertex v as the sum of effective resistance values of the incident edges, i.e., $r(v) = \sum_{e \in E_v} r(e)$, where E_v represents a set of edges incident to a vertex v .

The *BC_SV* algorithm first computes the BC tree decomposition, and then adds the cut vertices and their incident edges to the spectral sampling G' of G . Next, it computes a spectral vertex sampling G'_i for each biconnected component G_i , $i = 1, \dots, k$ of G . Specifically, for each component G_i , we compute the effective resistance values $r(v)$ of the vertices, and then sample the vertices with the largest effective resistance values. Finally, it merges G'_i , $i = 1, \dots, k$ to obtain the spectral sampling G' of G . The *BC_SV* algorithm is described as follows:

Algorithm: BC_SV

1. *Partitioning*: Divide a connected graph G into biconnected components, G_i , $i = 1, \dots, k$.
2. *Cut vertices*: Add the cut vertices and their incident edges to the spectral sampling G' of G .
3. *Spectral vertex sampling*: For each component G_i , compute spectral vertex sampling G'_i of G_i . Specifically, compute the effective resistance values $r(v)$ of the vertices, and then sample the vertices with largest effective resistance values.
4. *Aggregation*: Merge all G'_i of G_i to compute the Spectral vertex sampling G' of the original graph G .

BC tree-based distributed framework

In this section, we introduce the *BC-P* (BC Parallel) framework, a framework integrating BC tree decomposition and cloud computing to achieve better scalability and efficiency for analysis and visualization of big complex graphs. We apply a Divide-and-Conquer method to divide a connected graph into biconnected components using BC tree decomposition. We then assign the components to a set of servers in a balanced manner based on the sizes of the components.

Framework: BC-P

- 1 *BC Tree Decomposition*: Divide a connected graph G into biconnected components $C_i, i = 1, \dots, k$
- 2 *Partitioning*: Partition the set of components to be assigned to a set of servers $S_j, j = 1, \dots, h$ based on the sizes of the components, in a balanced manner.
- 3 *Parallel computing*: in parallel, each server independently performs the main analytical computation on its set of assigned components
- 4 *Aggregation*: merge results from each server to obtain the result for the whole graph

Using the BC tree decomposition, communication overheads can be reduced. This is because the graph is divided into biconnected components, which only intersect on cut vertices and share no edges. Thus, each server can perform computations on their assigned components independently.

The BC tree can also be exploited in the merge phase to emphasize the connectivity and topological structure of the graph. For example, sampling can be made to always include cut vertices and edges incident on them, or a drawing of the graph may be computed in a way that emphasizes the connectivity structure between the biconnected components.

Distributed BC tree-based spectral sparsification

We present *DBC_SS*, a distributed algorithms for *spectral sparsification*. The input graph is decomposed into biconnected components using BC tree decomposition, and the effective resistance values of edges are computed per biconnected component in parallel. The *DBC_SS* algorithm is described as follows:

Algorithm: DBC_SS

- 1 Divide a connected graph G into biconnected components $C_i, i = 1, \dots, k$
- 2 Partition the set of components to be assigned to a set of servers $S_j, j = 1, \dots, h$ based on the sizes of the components, in a balanced manner, using a greedy algorithm.
- 3 In parallel, each server independently computes the effective resistance values of edges per biconnected component.
- 4 Combine all the results from all server using a master instance to produce the final result.

Using BC tree decomposition, communication overheads can be removed for parallel effective resistance values computation. This is because the biconnected components do not share any edges, thus the effective resistance values can be computed independently for each component.

Distributed BC tree-based graph drawing

We present *DBC_GD*, a parallel algorithm for graph drawing. We use BC tree decomposition on the graph and apply a radial tree drawing algorithm on the BC tree to obtain

an initial layout for the original graph. Each biconnected component is drawn separately, and then each vertex in the radial tree drawing representing a block vertex is replaced with the drawing of the corresponding biconnected component. The *DBC_GD* algorithm is described as follows:

Algorithm: DBC_GD

- 1 Divide a connected graph G into biconnected components $C_i, i = 1, \dots, k$
- 2 Draw the BC tree using Radial Tree algorithm, each Block cut vertex representing either a cut vertex or a biconnected component.
- 3 Partition the set of components to be assigned to a set of servers $S_j, j = 1, \dots, h$ based on the sizes of the components, in a balanced manner, using a greedy algorithm.
- 4 Each server independently computes the layout of each biconnected component.
- 5 Merge all the layout results of each biconnected component into the radial tree drawing by replacing each block vertex in the BC tree with the drawing of its corresponding biconnected component to produce the final layout of the original graph.

To ensure that the layout for each biconnected component can be computed independently without causing issues in the merge step, we use a modified version of the radial tree layout where we scale the geometric size of the vertices of the BC tree by the size of the biconnected component represented. This size is then used to modify the size of the wedge assigned to the sub-tree as needed. This ensures that the drawings of the biconnected component will not overlap in the merge step.

BC_SS and BC_SV experiments

We first design experiments to compare the Spectral Sparsification (SS) (Eades et al. 2017b), *BC_SS* and Random Edge sampling (RE) (resp., Spectral Vertex sampling (SV) (Hu et al. 2019), *BC_SV* and Random Vertex sampling (RV)), implemented in Java. Analysis of experimental results such as metrics and statistics are implemented in Python, using the NetworkX library (Hagberg et al. 2008). All programs were run on a MacBook Pro with 2.2 GHz Intel Core i7, 16 GB 1600 MHz DDR3, and macOS Sierra version 10.12.6.

SS and SV have already been shown to obtain better quality samples compared to other well-known sampling methods (Eades et al. 2017b; Hu et al. 2019). Furthermore, an extensive comparison has been done showing the superior performance of spectral sparsification-based sampling to sampling based on various types of other graph analysis (Hong and Lu 2020). Our main focus is then to show that *BC_SS* and *BC_SV* obtain the same level of quality as SS and SV respectively with much better runtime. *The main hypotheses* of our *BC_SS* and *BC_SV* experiments include:

- **H1:** *BC_SS* computes effective resistance values of edges faster than SS (Eades et al. 2017b).
- **H2:** The effective resistance values and the rankings of edges (resp., vertices) computed by *BC_SS* (resp., *BC_SV*) are good approximations of those computed by SS (resp., SV), and their similarity increases with the sampling ratio.

- **H3:** Graph samples of *BC_SS* (resp., *BC_SV*) have almost the same sampling quality as *SS* (resp., *SV*), and significantly better than *RE* (resp., *RV*).
- **H4:** Graph samples computed by *BC_SS* (resp., *BC_SV*) produce almost the same visualization as *SS* (resp., *SV*).

The main rationale behind the hypotheses is that the graph samples computed by *BC_SS* and *SS* (resp., *BC_SV* and *SV*) would be very similar, since the ranking of edges (resp., vertices) based on the resistance values are highly similar. We experiment with benchmark real world graphs (Eades et al. 2017b) and synthetic data sets, see Table 1. The real world graphs are scale-free graphs with highly imbalanced sizes of biconnected components (i.e., big biconnected component). The graphs include social networks (*facebook*, *oflights*, *p2pG*, *soch*, *wiki*) and biological networks (*G4*, *G15*, *yeastppi*). The synthetic graphs are generated with balanced sizes of biconnected components. The structure is created by first generating a *k*-ary tree, then replacing the vertices with either biconnected components or cut vertices, alternating between levels; the generative model is similar to that used in Hong et al. (2018).

Runtime improvement

Figure 1 shows significant runtime improvement for computing effective resistance values by *BC_SS* over *SS*. We compute the runtime improvement of *BC_SS* over *SS* using the formula $RI = \frac{t(SS) - t(BC_SS)}{t(SS)}$, where *t(SS)* and *t(BC_SS)* are the runtimes of *SS* and *BC_SS* respectively. The runtime improvement is much higher for the synthetic graphs, achieving above 99% on average, while the improvement on the real world graphs, on average 45%, varies depending on their structures. For example, *BC_SS* improved 77% of the runtime for the *G4* graph, while it improved 23% for the Facebook graph due to the existence of the giant biconnected component. Overall, the average runtime improvement over all datasets is 68%.

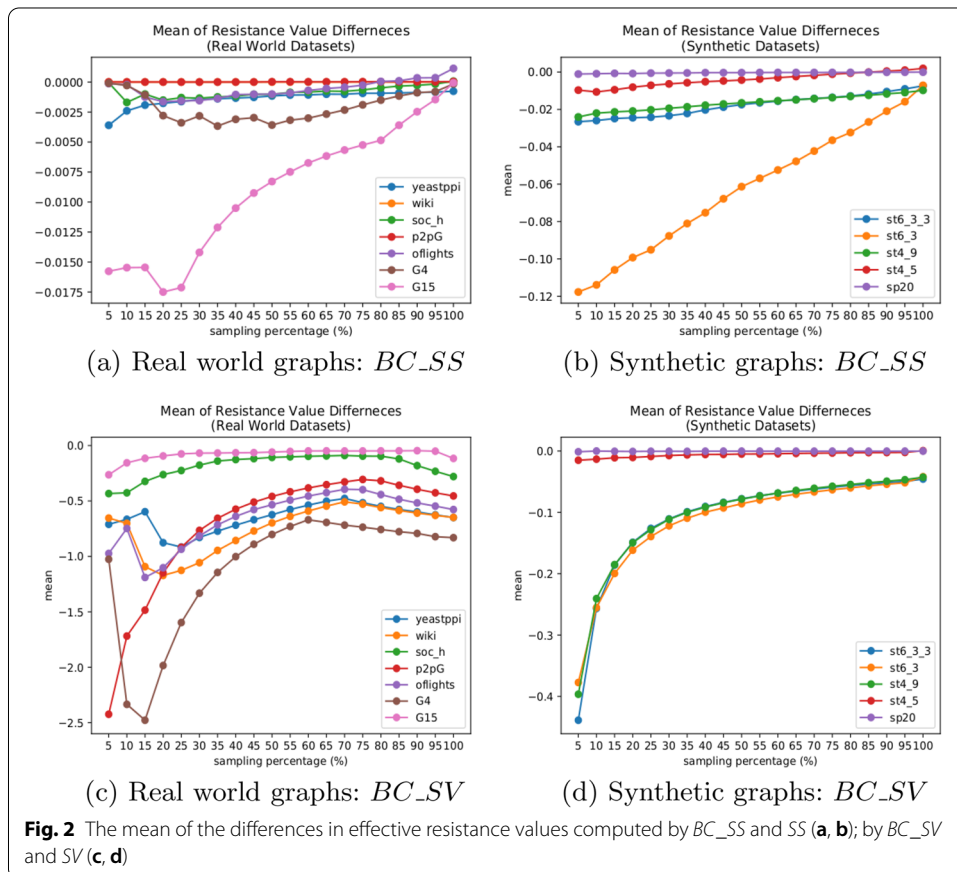
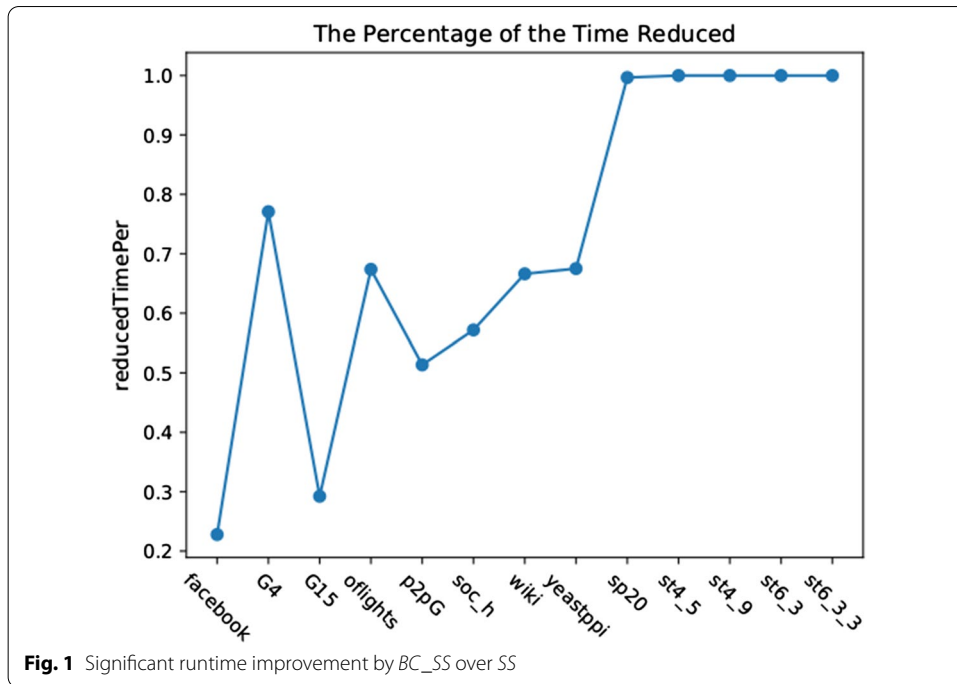
Overall, our experiments show that BC_SS is significantly faster than SS, supporting hypothesis H1.

Approximation of the effective resistance values

Figure 2a and b show the mean of the differences in effective resistance values computed by *SS* and *BC_SS* with sampling ratio from 5% to 100%. Figure 2c and d show the mean of

Table 1 Data sets

(a) Real world graphs			Synthetic graphs			
Graph	V	E	Graph	Abbr	V	E
Facebook	4039	88234	syn_path20_150_200_True	sp20	3462	5410
G4	2075	4769	syn_tree4_5_10_10_100_True	st4_5	21,955	34,957
G15	1789	20459	syn_tree4_9_10_10_30_True_2	st4_9	18,936	40,411
Oflights	2939	14458	syn_tree6_3_4_10_30_True_2	st6_3	11,679	24,868
p2pG	8846	31839	syn_tree6_3_4_10_30_True_3	st6_3_3	13,237	41,936
soch	2426	16,630				
wiki	7115	100,762				
yeastppi	2361	6646				



the differences in effective resistance values computed by *SV* and *BC_SV* with sampling ratio from 5% to 100%.

Overall, it clearly shows that the mean of the differences in effective resistance values computed by *BC_SS* and *SS* (resp., *BC_SV* and *SV*) is very small for most of the data sets, supporting hypothesis *H2*.

More specifically, for *BC_SS*, smaller than -0.0175 for real world graphs; smaller than -0.12, with one outlier, for synthetic graphs. For *BC_SV*, smaller than -2.5 (compared to resistance values of edges, -2.5 is equivalent to -0.06) for real world graphs; smaller than -0.5 (equivalent to -0.08) for synthetic graphs.

Interestingly, in contrast to the runtime improvement results, real world graphs have a better similarity in the effective resistance values than synthetic graphs, due to the existence of the big giant component. Namely, the effective resistance values computed from the big biconnected component are very similar to the effective resistance values computed for the whole graph. In the case of sparser graphs (e.g. *G4*), graphs with a very large giant biconnected component and thus very unbalanced biconnected component sizes (e.g. *G15*), or graphs with a very large number of cut vertices (e.g. *st_4_9*), the mean difference does not always change linearly.

It should be noted that even at 100% sampling rate, *BC_SS* and *BC_SV* still compute effective resistance per biconnected component. Thus, it is possible to still have some minor differences compared to the effective resistance values computed by *SS* and *SV*, which computes the effective resistance based on the whole graph.

Approximation of the ranking of edges and vertices

We define the *sampling accuracy* based on the proportion of the common sampled edges (resp., vertices) between two graph samples computed by *SS* and *BC_SS* (resp., *SV* and *BC_SV*). A high sampling accuracy indicates that both graph samples are highly similar. Namely, the sampling accuracy shows how well the effective resistance values computed by *BC_SS* (resp., *BC_SV*) can serve as a good approximation of the values computed by *SS* (resp., *SV*).

Figure 3a and b (resp., (c) and (d)) show the sampling accuracy of *BC_SS* (resp., *BC_SV*) with sampling ratio from 5% to 100%. It is easy to observe that for all data sets, the sampling accuracy increases as the sampling ratio increases, supporting hypothesis *H2*.

Specifically, for *BC_SS*, synthetic graphs perform better: they achieve above 50% sampling accuracy at sampling ratio 5%, and then quickly rise up to 80% at sampling ratio 15%, with steady improvement towards 100% as the sampling ratio increases.

The performance of *BC_SS* on the real world graphs shows different patterns, depending on their structure. Interestingly, the Facebook graph has excellent sampling accuracy at all sampling ratios, achieving above 80%, while it performs the worst on the runtime improvement and the difference in resistance values. On the other hand, graph *G4* shows very high performance on runtime improvement and the difference in resistance values is quite small, however the sampling accuracy is low when the sampling ratio is smaller than 20%.

For *BC_SV*, synthetic graphs show excellent performance overall, achieving above 70% sampling accuracy at sampling ratio 5%, and then rise up to 90% at sampling ratio 10%,

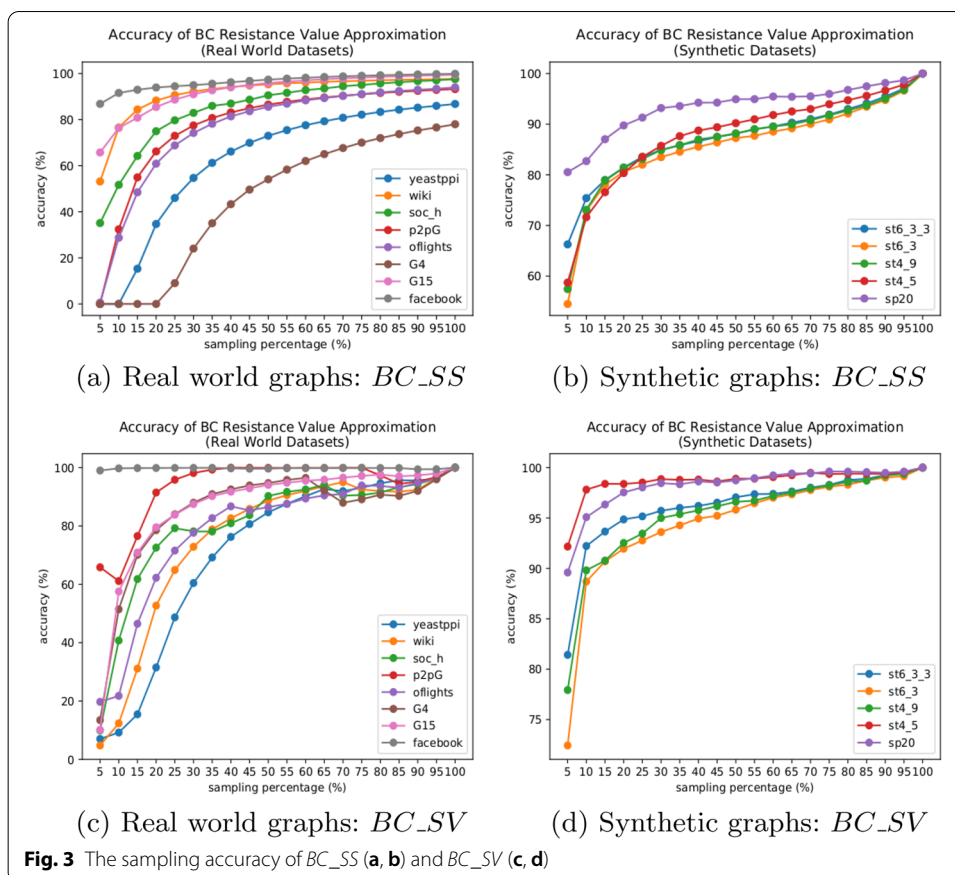


Fig. 3 The sampling accuracy of BC_{SS} (a, b) and BC_{SV} (c, d)

with steady improvement towards 100% as the sampling ratio increases. For real world graphs, BC_{SV} achieves above 70% sampling accuracy at sampling ratio 20% for all graphs. Particularly, the Facebook graph shows excellent sampling accuracy at all sampling ratios, achieving above 99%. Similar to the mean differences, BC_{SS} and BC_{SV} computing effective resistance values per biconnected component may lead to cases where the accuracy is not 100% even at 100% sample size, such as with sparse graphs with uneven component sizes (e.g. G4).

Furthermore, the correlation analysis of the ranking of edges (resp., vertices) based on resistance values computed by SS and BC_{SS} (resp., SV and BC_{SV}) shows strong and positive results for all data sets. Overall, our experiments and analysis confirm that BC_{SS} (resp., BC_{SV}) computes good approximations on the rankings of edges (resp., vertices) based on effective resistance values, compared to SS (resp., SV), supporting hypothesis H2.

Graph sampling quality metrics comparison

We use well known sampling quality metrics (Hu and Lau 2013): Degree Correlation (Degree), Closeness Centrality (Closeness), Clustering coefficient (CC), and Average Neighbor Degree (AND). More specifically, we use the *Kolmogorov-Smirnov* (KS) distance to compute the distance between two Cumulative Distribution Functions (CDFs) (Gammon et al. 1967). The KS distance value is between 0 to 1: the lower KS value means

the better result. Namely, a KS distance value closer to 0 indicates a higher similarity between CDFs.

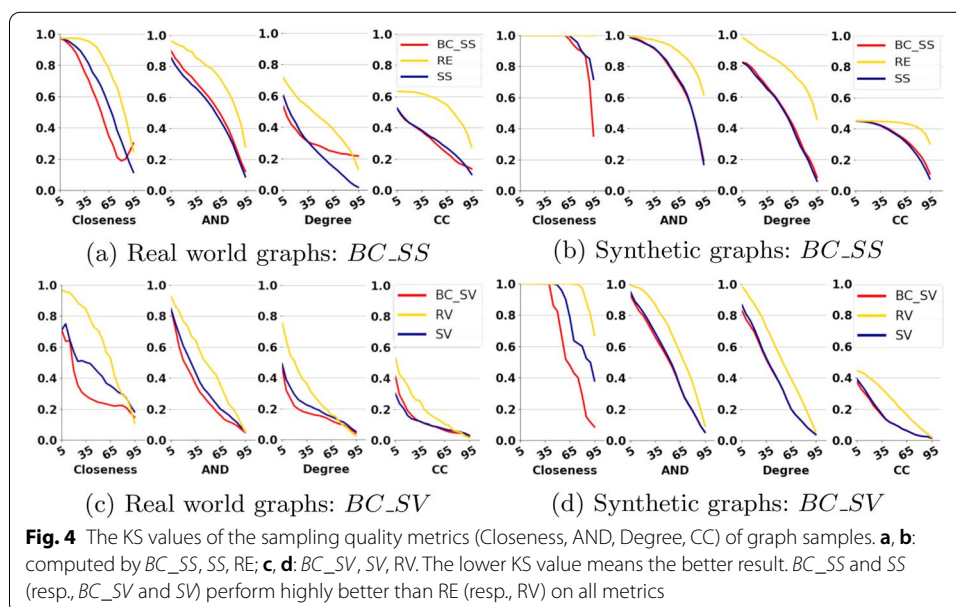
Figure 4a and b show the average (over all data sets) of the KS distance values of the graph samples computed by *BC_SS* (red), *SS* (blue), and Random Edge sampling (RE) (yellow), with four sampling quality metrics. Clearly, *SS* and *BC_SS* perform consistently better than RE for both types of graphs, as we expected. More importantly, the performance of *SS* and *BC_SS* are almost identical across all the metrics, especially for synthetic graphs. For the real world graphs, the performance of *SS* and *BC_SS* are highly similar on Closeness, AND, and CC metrics.

Figure 4c and d show the average (over all data sets) of the KS distance values of the graph samples computed by *BC_SV* (red), *SV* (blue), and Random Vertex sampling (RV) (yellow), with four sampling quality metrics. Clearly, *SV* and *BC_SV* perform consistently better than RV for both types of graphs, as we expected. More importantly, the performance of *SV* and *BC_SV* are almost similar on all the metrics, especially for AND, Degree, and CC. In particular, we observe that *BC_SV* shows the largest improvement on Closeness, significantly better than *SV*.

In summary, our experimental results with sampling quality metrics confirm that both SS and BC_SS (resp., SV and BC_SV) outperform RE (resp., RV), and the graph samples computed by BC_SS (resp., BC_SV) have almost the same sampling quality as those computed by SS (resp., SV), supporting hypothesis H3.

Jaccard similarity index comparison

We also computed the Jaccard similarity index (Jaccard 1912) for testing the similarity between the original graph G and the graph samples G' and G'' computed by *SS* and *BC_SS* (resp., *SV* and *BC_SV*). More specifically, it is defined as the size of the intersection divided by the size of the union of the neighbourhood of vertices in the two graphs (value 1 indicates that two graphs are the same): $MJS(G_1, G_2) = \frac{1}{|V|} \sum_{u \in V} \frac{|N_1(u) \cap N_2(u)|}{|N_1(u) \cup N_2(u)|}$



where $N_1(u)$ (resp. $N_2(u)$) is the neighbourhood of u in G_1 (resp. G_2) (Eades et al. 2017a). Our usage of Jaccard similarity is consistent with previous work on graph sampling and spectral sparsification, where its effectiveness for comparing graphs have been demonstrated (Hu et al. 2019; Meidiana et al. 2019).

Figure 5 shows the *average* Jaccard similarity values for real world graphs and synthetic graphs for BC_SS and SS (resp. BC_SV and SV), with sampling ratio from 5% to 95%. Clearly, for both data sets, *the Jaccard similarity index linearly increases with the sampling ratio, and SS and BC_SS (resp., SV and BC_SV) perform almost the same, supporting hypothesis H3.*

Visual comparison: SS versus BC_SS and SV versus BC_SV

We conduct visual comparisons of graph samples computed by SS , BC_SS , SV , and BC_SV using the *Backbone* layout, specifically designed to untangle the hairball drawings of large graphs (Nocaj et al. 2015).

Table 2 shows graph samples with sampling ratio at 20% for real world graphs (*facebook*, *G15*, *G4*, *oflights*, *soc_h*, *yeastppi*) as well as a synthetic graph *st4_5*. *Visual comparison clearly shows that SS and BC_SS (resp., SV and BC_SV) produce almost identical visualizations, supporting hypothesis H4.*

Overall, spectral edge sampling methods (SS and BC_SS) and spectral vertex sampling methods (SV and BC_SV) produce visually highly similar graph samples. For some cases, the density of graph samples are slightly different, depending on the density of the original graphs. For example, for graphs *G4* and *yeastppi*, spectral vertex sampling methods (SV and BC_SV) compute graph samples which better captures the dense structure of the original graph.

***DBC_SS* experiments**

We next conduct experiments comparing the efficiency of *DBC_SS* to sequential spectral sparsification SS . We ran effective resistance value computation on three different settings: sequential SS , *DBC_SS* on 2 servers, and *DBC_SS* on 5 servers; the usage of 2 and 5 servers is consistent with the experimental setup of previous topology-based and spectral sparsification-based distributed sampling works (Hong et al. 2018; Meidiana et al. 2019), chosen as a starting point to demonstrate the difference in runtime on

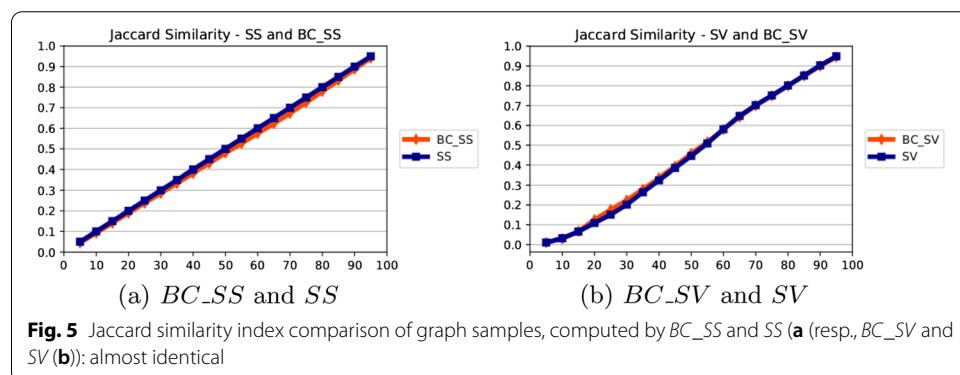









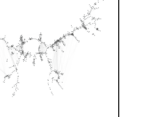
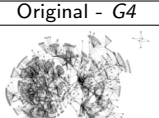
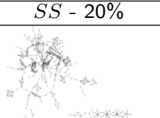
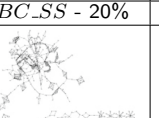
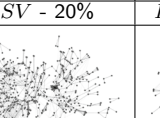
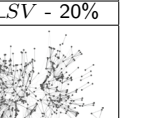
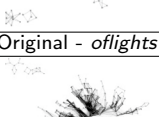
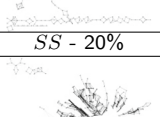
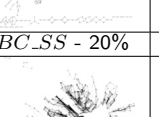
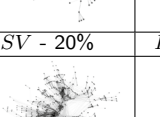
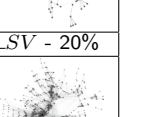

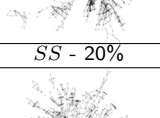
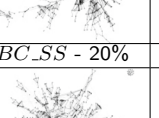
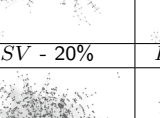
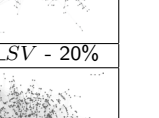
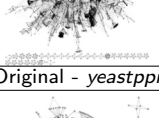
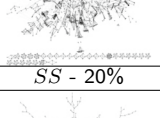
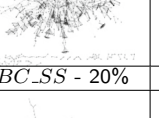
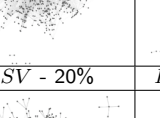
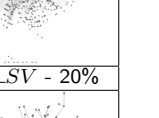
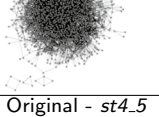
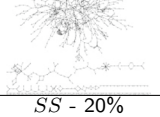
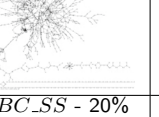
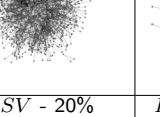
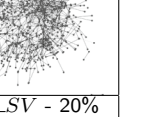


Table 2 Comparison of graph samples of real world graphs and a synthetic graph with 20% sampling ratio, computed by *SS*, *BC_SS*, *SV* and *BC_SV*

Original - facebook	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				
Original - G15	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				
Original - G4	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				
Original - oflights	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				
Original - soc_h	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				
Original - yeastppi	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				
Original - st4.5	<i>SS</i> - 20%	<i>BC_SS</i> - 20%	<i>SV</i> - 20%	<i>BC_SV</i> - 20%
				

SS and *BC_SS* (resp. *SV* and *BC_SV*) produce almost identical visualizations

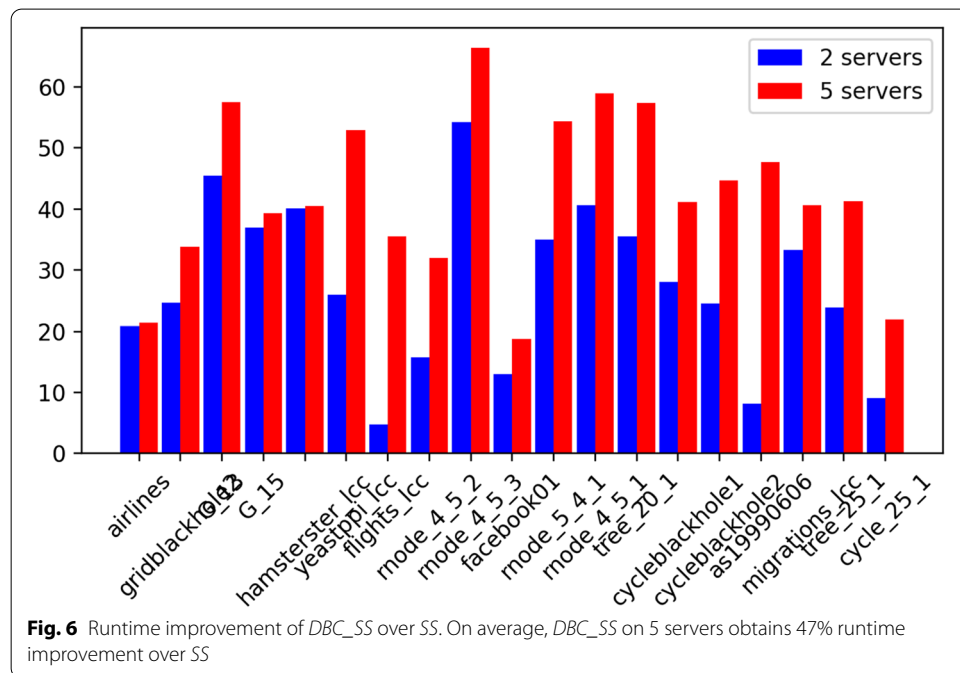
multiple configurations even on a small number of servers. We then compare the runtime improvements of *DBC_SS* on 2 and 5 servers to the baseline *SS*.

We conducted experiments using a mix of real world and synthetic datasets, described in Table 3. The real world datasets have been taken from the library of Hachul and Jünger (2007), the sparse matrices collection (Davis and Hu 2011), and the network repository (Rossi and Ahmed 2015), while the synthetic datasets were created with a globally sparse structure with locally dense “blobs”.

We expect that the parallel *DBC_SS* will provide better runtime efficiency over *SS*. We also expect *DBC_SS* to compute sparsifications which preserve the global skeleton structure of the full graphs. We formulate the following hypothesis:

Table 3 Data sets for *DBC_SS* experiments

Graph	V	E	Graph	V	E
Airlines	235	2101	rnode_5_4_1	4348	65,810
gridblackhole2	1535	14939	rnode_4_5_1	4622	86,690
G_13	1647	6487	tree_20_1	4702	95,727
G_15	1785	20459	cycleblackhole1	5080	51,767
hamsterster_lcc	2000	16097	cycleblackhole2	5080	51,829
yeastppi_lcc	2224	7049	as19990606	5188	10,974
flights_lcc	3397	19231	migrations_lcc	6025	9378
rnode_4_5_2	3466	19247	tree_25_1	6960	116,687
rnode_4_5_3	3881	36560	cycle_25_1	28,766	223,903
facebook01	4039	88,234			



- **H5:** *DBC_SS* runs faster than *SS*.
- **H6:** *DBC_SS* computes sparsified graphs with similar visualisation to the full graphs.

Runtime improvement

We compute the runtime improvement of *DBC_SS* over *SS* using the formula $RI = \frac{t(SS) - t(DBC_SS)}{t(DBC_SS)}$ where $t(SS)$ is the runtime of *SS* and $t(SS)$ is the runtime of *DBC_SS*.

Figure 6 shows the runtime improvement of *DBC_SS* over sequential effective resistance value computation. *DBC_SS* runs faster than *SS* on all datasets, on average with 27% runtime improvement on a 2-server configuration and 42% runtime improvement on a 5-server configuration. Larger improvements when using 5 servers over 2 servers can be seen when the graph has evenly-sized biconnected components (e.g. *cycleblackhole*

graphs), compared to graphs with a giant biconnected component (e.g. *airlines*, *hamsterster*). Overall, the results show that *DBC_SS* run significantly faster than *SS*, supporting hypothesis *H5*.

Visual comparison

Table 4 shows a visual comparison between the drawings of the original graph *G* and the sparsified graph *G'* for a few data sets. *G'* is computed by using *DBC_SS* to compute the effective resistance values per biconnected component of a graph *G*, adding all edges incident to the cut vertices of *G*, and then, for each biconnected component, adding edges in descending order of effective resistance until the desired sparsification rate is achieved. It can be seen that the sparsified graphs produced by *DBC_SS* display a similar global structure to the original graphs, supporting hypothesis *H6*.

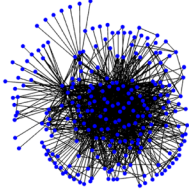
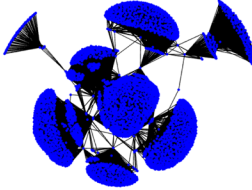
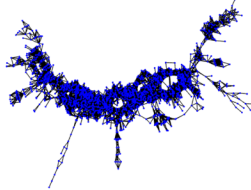
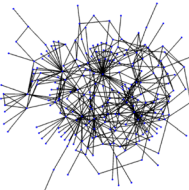
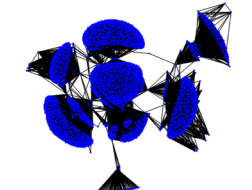
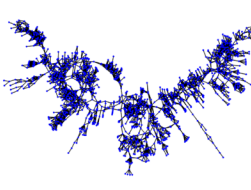
DBC_GD experiments

To evaluate the efficiency and effectiveness of *DBC_GD*, we executed experiments comparing *DBC_GD* to sequential graph drawing algorithms. We selected three original graph layouts of different types to be compared: *FR* (*Fruchterman-Reingold*) (Fruchterman and Reingold 1991), a force-directed layout; *FM³* (Hachul and Jünger 2004), a multi-level layout; and *SM* (*Stress Majorization*) (Hachul and Jünger 2004), a stress-based layout. We denote the respective *DBC_GD* variants as *DBC_FR*, *DBC_FM3*, and *DBC_SM*.

For each original graph layout, we draw the graphs using the layout and the corresponding *DBC_GD* variants. We then compare the results on runtime and quality metrics.

For this experiment, we created synthetic datasets where the BC tree decomposition produces a balanced tree. The details of the graphs are shown in Table 5, named in the

Table 4 Visual comparison between drawings of full graphs *G* and their sparsifications *G'* computed by *DBC_SS*

<i>airlines</i> (<i>G</i>)	<i>facebook</i> (<i>G</i>)	<i>G_15</i> (<i>G</i>)
		
<i>airline</i> (<i>G'</i>)	<i>facebook</i> (<i>G'</i>)	<i>G_15</i> (<i>G'</i>)
		

DBC_SS computes sparsifications that preserve the global skeleton structure of the graph

Table 5 Dataset for *DBC_GD* experiments

Graph	V	E	Graph	V	E
treev_3_3_1000	12795	39144	treev_4_3_1000	16,424	49,554
treev_3_3_700	8842	26374	treev_4_3_700	11,592	35,852
treev_3_3_500	6469	20397	treev_4_3_500	8341	25,308
treev_3_3_300	3971	12666	treev_4_3_300	5001	15,195
treev_4_2_1000	12765	39517	treev_4_4_1000	20,673	62,441
treev_4_2_700	9028	27370	treev_4_4_700	14,282	44,336
treev_4_2_500	6422	19241	treev_4_4_500	10,269	29,920
treev_4_2_300	3815	11448	treev_4_4_300	6335	18,852

format *treev*_[# of children per BC tree node]_[# of levels]_[approx. # of vertices per biconnected component].

We expect that the *DBC_GD* algorithms will run faster than the sequential graph layouts. Furthermore, we expect that they will obtain drawings that are at least of similar quality as the sequential graph layouts, due to the BC tree decomposition emphasizing the connectivity between the biconnected components. We formulate the following hypotheses:

- **H7:** *DBC_FR*, *DBC_SM*, and *DBC_FM3* run faster than FR, SM, and FM³ respectively.
- **H8:** *DBC_FR*, *DBC_SM*, and *DBC_FM3* computes drawings on the same level of quality as FR, SM, and FM³ respectively.

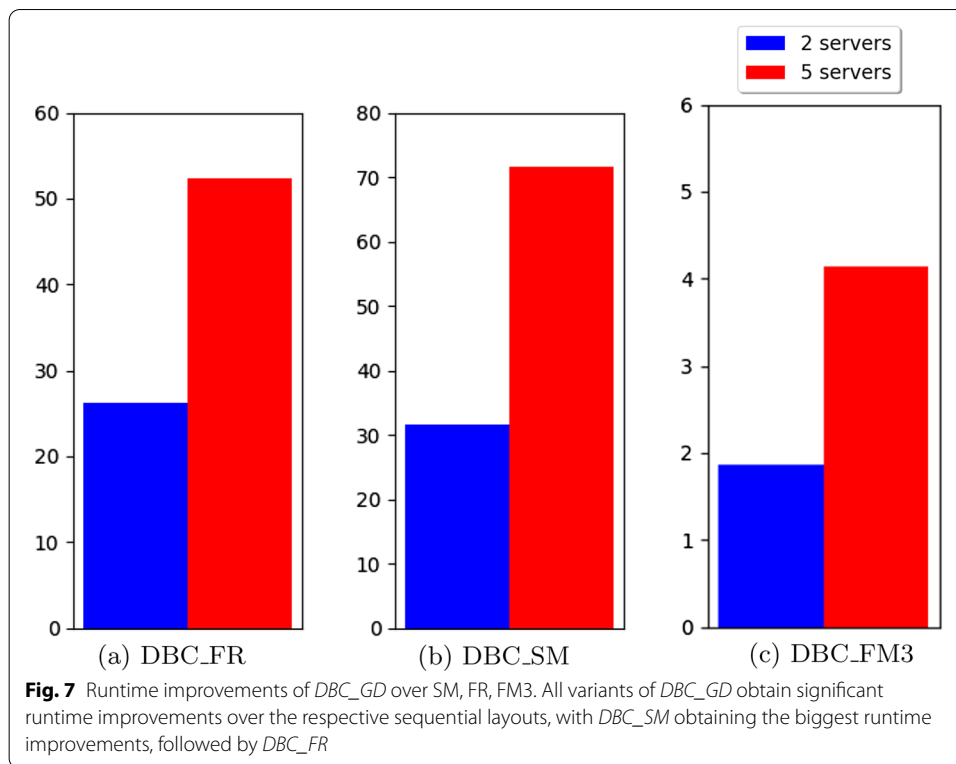
Runtime improvement

Figure 7 shows the runtime improvements of *DBC_GD* over the original sequential layouts. For this experiment, as the runtime is often very low compared to the original, the runtime improvement formula used for the *DBC_SS* experiments often does not adequately show the difference in efficiency between 2- and 5-server settings. We use the following formula: $RI = \frac{t(OL)}{t(DBC_GD)}$ where *t(OL)* is the runtime of the original sequential layout and *t(DBC_GD)* is the runtime of our algorithms.

Compared to the original sequential algorithms, *DBC_GD* obtains significant runtime improvements, as shown in Fig. 7. *DBC_FM3* on 2 and 5 servers is 1.86 times and 4.14 times faster than sequential FM³ respectively. More significant runtime improvements are seen with *DBC_FR* and *DBC_SM*: *DBC_FR* with 2 and 5 servers is 26.28 times and 52.31 times faster than FR respectively; *DBC_SM* on 2 and 5 servers is 31.61 times and 71.48 times faster than SM respectively. *In summary, all variants of DBC_GD run faster than their respective sequential layouts, supporting hypothesis H7.*

Visual comparison

Table 6 shows the visual comparison of the graph drawings using six algorithms: FR, SM, FM³, and their respective *DBC_GD* counterparts. With the synthetic datasets



with balanced BC trees, such as the first row of Table 6, *DBC_GD* with FR and SM obtain better shape drawings than their sequential counterparts, with fewer crossings than FR and with better separation between biconnected components than SM. Compared to FM^3 , *DBC_GD* obtains similar shape for the global skeleton, while also not collapsing each biconnected too extremely. These results support hypothesis H8.

We also show an example combining the layout of *DBC_GD* with the sparsification of *DBC_SS*, displayed in Fig. 8. It can be seen that not only the global skeleton of the graph is displayed, but the sparsification shows details of the intra-component structure that may be missed in the drawing for the original graph, where each biconnected component are displayed as “blobs”. Thus, combining *DBC_GD* with *DBC_SS* can be useful in further highlighting important topological structures of graphs.

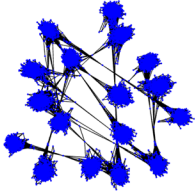
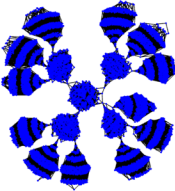
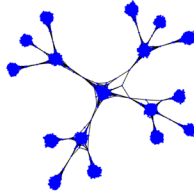
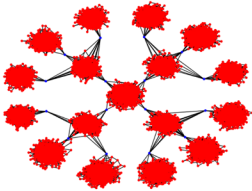
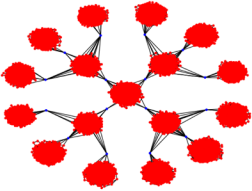
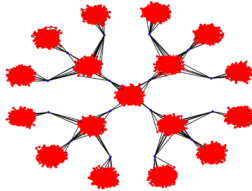
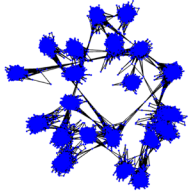

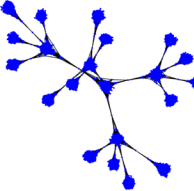
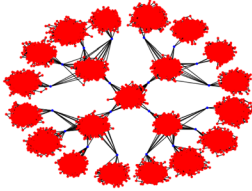
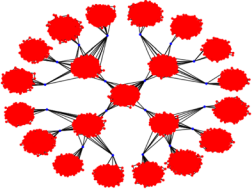
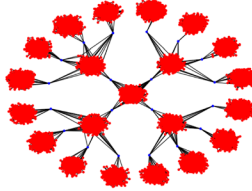
In summary, DBC_GD obtains drawings with better shape than the sequential counterparts, supporting hypothesis H8. Furthermore, a combination of DBC_SS and DBC_GD is able to both show the global skeleton and local structure of a graph.

Quality metrics comparison

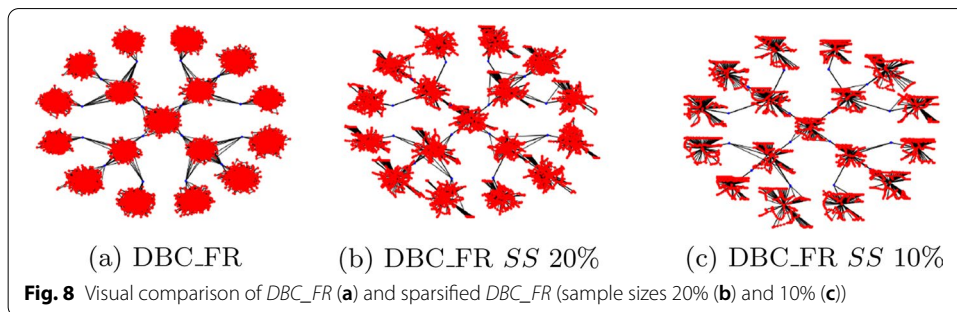
We also compute quality metrics for the drawings to further evaluate the effectiveness of *DBC_GD*. We use *edge crossing* and *shape-based* metrics. The *shape-based metric* (Eades et al. 2017a) measures the *faithfulness* of graph drawing, i.e., how well the *shape* of the drawing represents the structure of the graph.

Figure 9a and b show the improvements obtained by *DBC_GD* in edge crossing and shape-based metrics respectively. *DBC_GD* obtains significantly fewer edge crossings

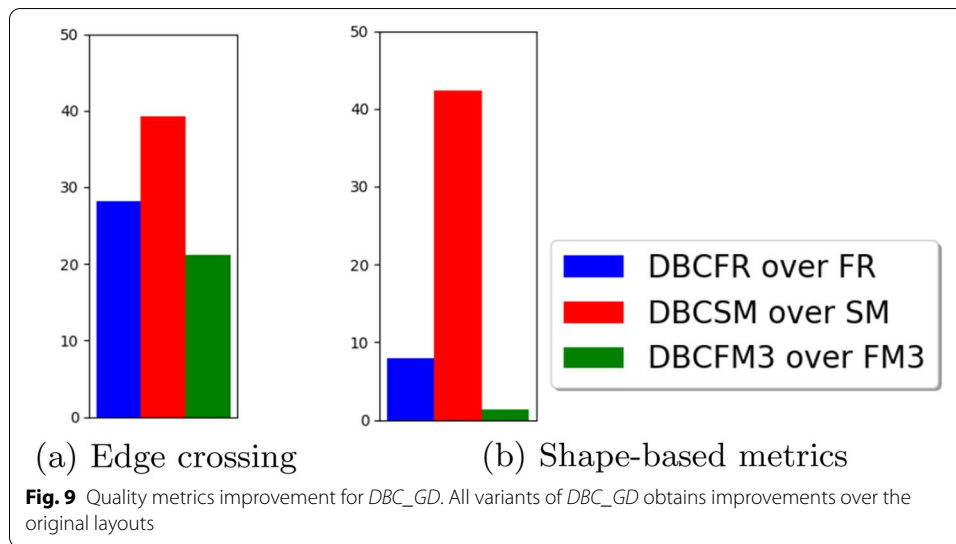
Table 6 Visual comparison between FR, SM, FM³ and DBC_GD variants

FR	SM	FM3
		
<i>DBC_FR</i>	<i>DBC_SM</i>	<i>DBC_FM3</i>
		
FR	SM	FM3
		
<i>DBC_FR</i>	<i>DBC_SM</i>	<i>DBC_FM3</i>
		

DBC_GD produces layouts with similar quality to FM³ and better than FR and SM



than SM, FR and FM3 at 39.63%, 28.2%, 21.18% respectively. With SM, *DBC_GD* obtains significantly higher shape-based metrics, at 58.38 %. Furthermore, *DBC_FR* obtains 13.84% higher shape-based metrics than FR. *In summary, the improvement in metrics shows that DBC_GD is even more effective than the corresponding sequential layouts, supporting hypothesis H8.*



Conclusion and future work

In this paper, we presented two new spectral sampling methods, *BC_SS* and *BC_SV*, tightly integrating the BC tree decomposition for fast computation and spectral sparsification to obtain high quality graph samples, preserving structural properties of graphs. We also designed, implemented, and evaluated *DBC_SS* and *DBC_GD*, distributed algorithms for spectral sparsification and graph drawing integrating BC tree decomposition of graphs with distributed computing on the cloud.

Extensive experimental results with both real world graphs and synthetic graphs demonstrate that our new BC tree-based spectral sampling approaches, *BC_SS* and *BC_SV*, are significantly faster than existing methods, on average 72% faster, while preserving highly similar quality sampling results, based on the comparison of resistance values, rankings of edges/vertices, graph sampling quality metrics, Jaccard similarity index, and visual comparison.

Furthermore, we also validate the effectiveness of our BC tree-based distributed framework: *DBC_SS* obtains significant runtime improvements over sequential spectral sparsification, at 47% on average when running on 5 servers. *DBC_GD* runs 4, 52, and 71 times faster than sequential FM3, FR, and Stress Majorization respectively, while also producing graph drawings with similar or better quality than the sequential graph drawing algorithms.

For future work, we plan to design new graph sampling methods for big graph visualization, by combining other graph partitioning methods. For example, see (Meidiana et al. 2019) for an edge sampling method integrating spectral sparsification with the decomposition of biconnected graphs into triconnected components.

Abbreviations

AND: Average neighbor degree; BC: Block cut-vertex; BC_P: Block cut-vertex parallel; BC_SS: Block cut-vertex spectral sparsification; BC_SV: Block cut-vertex spectral vertex; CC: Clustering coefficient; CDF: Cumulative distribution function; DBC_SS: Distributed block cut-vertex spectral sparsification; DBC_GD: Distributed block cut-vertex graph drawing; FR: Fruchterman–Reingold; FM³: Fast multipole multilevel method; KS distance: Kolmogorov–Smirnov distance; RE: Random edge; RV: Random vertex; SM: Stress majorization; SS: Spectral sparsification; SV: Spectral vertex sampling.

Acknowledgements

The conference version of this paper was published in Hu et al. (2020).

Authors' contributions

JM.H. designed, implemented, and evaluated the *BC_SV* algorithm. J.L.C. designed, implemented, and evaluated the *BC_SS* algorithm. M.C. evaluated the *BC_SS* and *BC_SV* algorithms. T.T.C. and A.M. designed and evaluated the *DBC_SS* and *DBC_GD* algorithms, and T.T.C. implemented the *DBC_SS* and *DBC_GD* algorithms. S-H.H., P.E., and K-L.M. designed and evaluated the *BC_SV* and *BC_SS* algorithms. S-H.H. also designed and evaluated the *DBC* algorithms. All authors read and approved the final manuscript.

Funding

This work is supported by an ARC (Australian Research Council) DP (Discovery Project) grant.

Availability of data and materials

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations**Competing interests**

The authors declare that they have no competing interests.

Author details

¹School of Computer Science, University of Sydney, Sydney, Australia. ²Hanoi University of Science and Technology, Hanoi, Vietnam. ³University of California at Davis, Davis, USA.

Received: 2 March 2021 Accepted: 18 June 2021

Published online: 21 August 2021

References

- Arleo A, Didimo W, Liotta G, Montecchiani F (2019) A distributed multilevel force-directed algorithm. *IEEE Trans Parallel Distrib Syst* 30(4):754–765. <https://doi.org/10.1109/TPDS.2018.2869805>
- Barrat A, Barthelemy M, Pastor-Satorras R, Vespignani A (2004) The architecture of complex weighted networks. *Proc Natl Acad Sci USA* 101(11):3747–3752
- Davis TA, Hu Y (2011) The university of Florida sparse matrix collection. *ACM Trans Math Softw (TOMS)* 38(1):1
- Eades P (1984) A heuristic for graph drawing. *Congr Numer* 42:149–160
- Eades P (1991) Drawing free trees. International Institute for Advanced Study of Social Information Science, Fujitsu Limited
- Eades P, Hong S-H, Nguyen A, Klein K (2017a) Shape-based quality metrics for large graph visualization. *J Graph Algorithms Appl* 21(1):29–53
- Eades P, Nguyen QH, Hong S (2017b) Drawing big graphs using spectral sparsification. *Proc GD 2017*:272–286
- Forman JJ, Clemons PA, Schreiber SL, Haggarty SJ (2005) Spectralnet—an application for spectral graph analysis and visualization. *BMC Bioinform* 6(1):1–13
- Freeman LC (1978) Centrality in social networks conceptual clarification. *Soc Netw* 1(3):215–239
- Fruchterman TMJ, Reingold EM (1991) Graph drawing by force-directed placement. *Softw Pract Exp* 21(11):1129–1164. <https://doi.org/10.1002/spe.4380211102>
- Galimberti E, Madeddu C, Bonchi F, Ruffo G (2019) Visualizing structural balance in signed networks. In: International conference on complex networks and their applications. Springer, pp 53–65
- Gammon J, Chakravarti IM, Laha RG, Roy J (1967) Handbook of methods of applied statistics
- Gansner ER, Koren Y, North S (2004). Graph drawing by stress majorization. In: International symposium on graph drawing. Springer, pp 239–250
- Gera R, Alonso L, Crawford B, House J, Mendez-Bermudez J, Knuth T, Miller R (2018) Identifying network structure similarity using spectral graph theory. *Appl Netw Sci* 3(1):1–15
- Hachul S, Jünger M (2004). Drawing large graphs with a potential-field-based multilevel algorithm. In: International symposium on graph drawing. Springer, pp 285–295
- Hachul S, Jünger M (2007) Large-graph layout algorithms at work: an experimental study. *J Graph Algorithms Appl* 11(2):345–369
- Hagberg A, Swart P, Chult DS (2008) Exploring network structure, dynamics, and function using networkx
- Hong S-H, Lu S (2020) Graph sampling methods for big complex networks integrating centrality, k-core, and spectral sparsification. In: Proceedings of the 35th annual ACM symposium on applied computing, pp 1843–1851
- Hong S-H, Nguyen Q, Meidiana A, Li J, Eades P (2018) BC tree-based proxy graphs for visualization of big graphs. In: 2018 IEEE pacific visualization symposium (PacificVis). IEEE. <https://doi.org/10.1109/pacificvis.2018.00011>
- Hopcroft J, Tarjan R (1973) Algorithm 447: efficient algorithms for graph manipulation. *Commun ACM* 16(6):372–378
- Hu P, Lau WC (2013) A survey and taxonomy of graph sampling. *CoRR*, abs/1308.5865
- Hu J, Hong S, Eades P (2019) Spectral vertex sampling for big complex graphs. In: Proceedings of the complex networks, pp 216–227
- Hu J, Hong S-H, Chen J, Torkel M, Eades P, Ma K-L (2020). Connectivity-based spectral sampling for big complex network visualization. In: International conference on complex networks and their applications. Springer, pp 237–248

- Jaccard P (1912) The distribution of the flora in the alpine zone. 1. *New Phytol* 11(2):37–50
- Koren Y (2003) On spectral graph drawing. In: *International computing and combinatorics conference*. Springer, pp 496–508
- Leskovec J, Faloutsos C (2006) Sampling from large graphs. In: *Proceedings of the SIGKDD*. ACM, pp 631–636
- Meidiana A, Hong S, Huang J, Eades P, Ma K (2019) Topology-based spectral sparsification. In: *Proceedings of the LDAV*, pp 73–82
- Newman MEJ (2003) Mixing patterns in networks. *Phys Rev E* 67(2):26126
- Nocaj A, Ortmann M, Brandes U (2015) Untangling the hairballs of multi-centered, small-world online social media networks. *JGAA* 19(2):595–618
- Rossi R, Ahmed N (2015) The network data repository with interactive graph analytics and visualization. In: *Twenty-Ninth AAAI conference on artificial intelligence*, pp 4292–4293
- Saramki J, Kivelä M, Onnela J-P, Kaski K, Kertész J (2007) Generalizations of the clustering coefficient to weighted complex networks. *Phys Rev E* 75(2):27105
- Serrano MÁ, Boguná M, Vespignani A (2009) Extracting the multiscale backbone of complex weighted networks. *Proc Natl Acad Sci* 106(16):6483–6488
- Spielman DA (2007). Spectral graph theory and its applications. In: *48th annual IEEE symposium on foundations of computer science (FOCS'07)*. IEEE, pp 29–38
- Spielman DA, Srivastava N (2011) Graph sparsification by effective resistances. *SIAM J Comput* 40(6):1913–1926
- Spielman DA, Teng S-H (2011) Spectral sparsification of graphs. *SIAM J Comput* 40(4):981–1025
- Wu Y, Cao N, Archambault DW, Shen Q, Qu H, Cui W (2017) Evaluation of graph sampling: a visualization perspective. *IEEE TVCG* 23(1):401–410

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

Submit your next manuscript at ▶ [springeropen.com](https://www.springeropen.com)
