On structure preserving sampling and approximate partitioning of graphs

Wouter van Heeswijk, George H. L. Fletcher, Mykola Pechenizkiy
Department of Computer Science
Eindhoven University of Technology
P.O. Box 513, NL-5600MB, Eindhoven, the Netherlands
woutervh@gmail.com, g.h.l.fletcher@tue.nl, m.pechenizkiy@tue.nl

ABSTRACT

Massive graphs are becoming increasingly common in a variety of domains such as social networks and web analytics. One approach to overcoming the challenges of size is to sample the graph, and perform analytics on the smaller graph. However, to be useful, the sample must maintain the properties of interest in the original graph. In this paper, we analyze the quality of five representative sampling algorithms in how well they preserve graph structure, the bisimulation structure of graphs in particular. As part of this study, we also develop a new scalable algorithm for computing bisimulation partitions of massive graphs. We empirically demonstrate the superior performance of our new algorithm in both sequential and distributed settings.

CCS Concepts

•**Information systems** → *Data mining;*

Keywords

networks, graphs, bisimulation, sampling

1. INTRODUCTION

Large graphs are increasingly common in data analytics in a wide variety of domains (e.g., web analytics, social media, state spaces of complex systems, citation networks, etc). Traditional solutions for graph analytics are often not suited to deal with the sheer size of modern graphs. In this paper we study three basic approaches to specifically deal with the scale of such graphs by reducing their size, while aiming to preserve their basic topological structure: sampling, approximation, and distribution.

Many metrics exist that attempt to capture structural properties of a graph [10]. As a case study, we focus on preserving the *bisimulation* structure of graphs. Bisimulation is an equivalence relation on the nodes of a graph which finds basic applications in many disciplines, such as bioinformatics, sociology, structured data management, and data visualization [6, 7, 17]. For example, standard structural notions used in social networks for role analysis correspond to variations of bisimulation [14, 19]. In this example the bisimulation partition assigned to a node in the graph represents the person's role in the social network. The algorithms we study for computing bisimulation partitions (BSPs) are easily modified to compute the partition of a graph modulo other equivalence relations.

Structure-preserving sampling. The American Academy of Sciences highlights both the importance of sampling for data gathering and for data reduction, as well as the lack of principled structure-preserving sampling solutions for massive graphs [15]. Towards addressing this, we investigate how well five representative sampling methods perform at maintaining bisimulation structure. We introduce two metrics (correctness and coverage) which measure the accuracy with which the sampled subgraph represents the bisimulation structure of the original graph. These two metrics correspond to the precision and recall for the set of partition blocks of the sample graph and the original graph. Since it is reasonable to expect a power law distribution for the size of partition blocks modulo bisimulation [12], we also introduce and study two weighted variants of these metrics.

Approximate partitioning. Our second approach is to approximate the BSP of a graph, since traditional exact solutions have linearithmic to quadratic time complexity, which becomes impractical on massive graphs. We introduce a new algorithm which builds on existing "signature"-based partition algorithms [11]. Our solution sacrifices correctness guarantees in order to become more efficient. Rather than comparing actual node signatures (as defined in Section 2) we compare hash values of the signatures instead. This may lead to hash collisions which could affect the partition computed by the approximate algorithm. The quality of the hashing function which is chosen is therefore of great importance to the accuracy of this algorithm. We consider it beyond the scope of this paper to discuss the effect of different hashing functions in more detail and instead refer to [21]. Our algorithm achieves essentially linear running time in the size of the graph (while producing high quality partitions, as we demonstrate in Section 5), and is thus better suited in practice than state-of-the-art solutions.

Distributed partitioning. Our third approach is to distribute the workload onto multiple machines. In particular, we study how well our approximate algorithm works in a dis-

SAC 2016, April 04-08, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2851613.2851650

tributed context, compared to traditional signature-based distributed solutions [4]. To compare the performance of the distributed algorithms we adopt the three basic metrics (Makespan, Data Shipment, Visit Times) proposed in [13]. We derive analytical complexity bounds on the Makespan (running time) of the exact and approximate algorithms in both sequential and distributed settings. We also derive analytical bounds for the Data Shipment and Visit Times measurements for both algorithms in a distributed setting. Our experimental study demonstrates the superior performance of our solution over the-state-of-the-art.

Contributions. We initiate the principled study of structure-preserving sampling of graphs,

- introducing intuitive quality metrics for graph structure preservation and analyze representative sampling algorithms with respect to these metrics;
- presenting a new scalable algorithm for computing approximate BSPs of massive graphs which is significantly faster than traditional algorithms, while still giving solutions which are completely correct for the graphs we study; and,
- empirically demonstrating the superior performance of our new algorithm compared to the standard algorithm in both sequential and distributed settings.

For further study, all algorithmic contributions of this work are made available online as open-source code.[1]

Related work. We have indicated closely related research in our discussion above. More generally, our study builds on the rich literature on graph sampling [2, 10], bisimulation partitioning [1, 11], graph similarity measures [7, 20], graph sparsification and summarization [5, 8, 9, 18], and distributed graph algorithms [13]. To our knowledge, our work is the first to study the ability of graph sampling methods to preserve "global" topological structure (such as bisimulation), and is the first effort to consolidate research in these areas with a focus on structure-preserving graph reduction.

2. PRELIMINARIES

Graphs and bisimulation. We consider node- and edge-labeled graphs $G = (V, E)$, with node set V and edge set E. By $\lambda_V(u)$ we denote the label of a node $u \in V$, and by $\lambda_E(e)$ we denote the label of an edge $e \in E$. We will use $u \xrightarrow{e} v$ to denote an edge e going from node u to node v.

DEFINITION 1. *Let k be a non-negative integer and $G = (V, E)$ be a graph. We say nodes $u, v \in V$ are k-*bisimilar *(denoted by $u \approx^k v$) if and only if the following holds:*

1. $\lambda_V(u) = \lambda_V(v)$,
2. *if $k \geq 1$ then for every outgoing edge $u \xrightarrow{e} u'$ there exists an edge $v \xrightarrow{e'} v'$ such that $\lambda_E(e) = \lambda_E(e')$ and $u' \approx^{k-1} v'$, and*
3. *if $k \geq 1$ then for every outgoing edge $v \xrightarrow{e} v'$ there exists an edge $u \xrightarrow{e'} u'$ such that $\lambda_E(e) = \lambda_E(e')$ and $v' \approx^{k-1} u'$.*

As it can easily be shown that the relation $\approx^k$ is an equivalence relation on V, we can partition the node set into partition blocks (equivalence classes). We call this partition the k-BSP of G. In Figure 1 we show an example graph. In Figure 2 we show the 0- and 1-BSPs of the example graph. In the case of $k = 0$ all we need to consider are the node labels, hence the nodes from each category are in their own partition blocks. For the case of $k = 1$ we also look at the nodes reachable after a single step from each node, and therefore Author C is split from Author A and Author B.

[1]All algorithmic contributions available as open-source at https://github.com/woutervh-/BisimulationSampling.

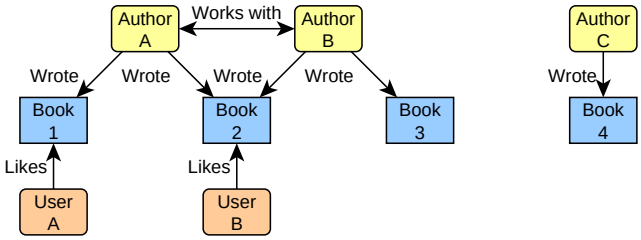


Figure 1: Example of a graph with labeled nodes Author, Book, and User, and edges representing relations between those nodes labeled with Wrote, Works with, and Likes.

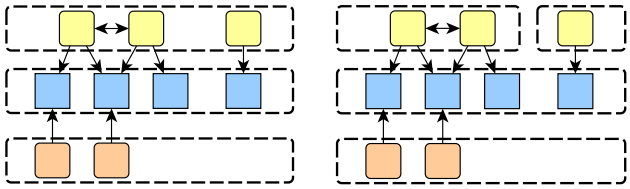


Figure 2: The example graph partitioned by 0-bisimulation (left) and 1-bisimulation (right).

Clearly, as k increases, the partition induced becomes refined. That is, for $i < j$ the partition blocks of the j-BSP are subsets of those of the i-BSP. In the interest of computation, we want to know which value of k is sufficient to give us the most refined (unbounded) BSP of a graph. Proposition 1 establishes that there exists a bounded value k_{max} which gives us this partition.

PROPOSITION 1. *Let $G = (V, E)$ be a graph. There is a value $0 \leq k_{max} < |V|$ for which the following holds:*

1. *the k_{max}-BSP of G refines any k-BSP of G, where $0 \leq k < k_{max}$, and*
2. *the k_{max}-BSP of G is the same as any k-BSP of G, where $k_{max} < k$.*

I.e. the k_{max}-BSP of G is the coarsest stable BSP of G.

Note that the 2-BSP of the example graph of Figure 1 would be the same as the 1-BSP; therefore we say that for the example graph we have $k_{max} = 1$. In applications such as social network analysis or structured data management, it can be useful to look at bounded k-bisimulation. Hence we will keep making the distinction between different levels of bounded bisimulation.

To compute the k-BSP of a graph efficiently we must be able to efficiently determine whether or not two nodes are k-bisimilar. We iteratively compute signatures (Definition 2), which helps us decide this [11, 4]. In Proposition 2 we establish that these signatures indeed capture k-bisimilarity.

DEFINITION 2. *Let k be a non-negative integer and $G = (V, E)$ be a graph. The k-bisimulation signature of node $u \in$*

V *is the pair* $sig_k(u) = (\lambda_V(u), L)$ *where:*

$$L = \begin{cases} \emptyset & k = 0, \\ \left\{ (\lambda_E(e), sig_{k-1}(v)) \middle| u \xrightarrow{e} v \right\} & k \geq 1. \end{cases}$$

PROPOSITION 2. *Let* k *be a non-negative integer and* $G = (V, E)$ *be a graph. For any two nodes* $u, v \in V$ *we have* $u \approx^k v$ *if and only if* $sig_k(u) = sig_k(v)$.

Partition quality. In order to assess the quality of graph sampling algorithms in maintaining bisimulation structure we introduce two intuitive metrics, *correctness* and *coverage* (Definition 3) to measure the accuracy with which a subgraph (sample) represents the original graph in terms of their BSPs. Depending on the application of a sampler it could matter how many nodes are in the partition blocks that are maintained through sampling. In most datasets we observe a power law behavior in the size of the partition blocks, meaning that a few partition blocks contain most nodes and there are many small partition blocks [12]. To accommodate for this situation we also introduce weighed versions of our metrics which take into account the number of nodes in the partition blocks. The values for (un)weighted correctness and coverage range between 0 and 1. Ideally a sampled graph has exactly those partition blocks that the original graph has, making all metrics equal to 1. In the worst case the sampled graph has none of the original graph's partition blocks, making all metrics equal to 0.

DEFINITION 3. *Let* $G = (V, E)$ *be a graph and let* $S = (V', E')$ *be a subgraph of* G*, i.e., a graph with nodes* $V' \subseteq V$ *and edges* $E' \subseteq E$*. Furthermore, let* $\mathcal{P}_G$ *and* $\mathcal{P}_S$ *denote the sets of signatures of the partition blocks of the BSPs of* G *and* S*, respectively. The correctness (*P*) and coverage (*R*) of* S *are defined as follows:*

$$P = \frac{|\mathcal{P}_G \cap \mathcal{P}_S|}{|\mathcal{P}_S|}, \qquad R = \frac{|\mathcal{P}_G \cap \mathcal{P}_S|}{|\mathcal{P}_G|}.$$

The weighted correctness (WP*) and weighted coverage (*WR*) of* S *are defined as follows:*

$$WP = \frac{|\{v \in V' | v \in P \in \mathcal{P}_G \cap \mathcal{P}_S\}|}{|V'|},$$

$$WR = \frac{|\{v \in V | v \in P \in \mathcal{P}_G \cap \mathcal{P}_S|}{|V|}$$

where P *denotes an equivalence class of nodes in* G.

As an example of our metrics, consider subgraph S of the graph G of Figure 1 consisting of the nodes Author C and Book 4 and the single Wrote edge between them. Under 1-bisimulation as visualized in Figure 2 (right), we have that S has correctness $\frac{2}{2}$ (i.e., it only has equivalence classes from G) and coverage of $\frac{2}{4}$ (i.e., it misses the User equivalence class and the equivalence class of those Authors that have worked with other authors). If we consider the weighted metrics, we have that S still has 100% correctness while its coverage increases to $\frac{5}{9}$.

Distributed metrics. We measure the complexity of our distributed algorithms using three metrics: Makespan, Data Shipment, and Visit Times as in [13]. **Makespan** measures the time it takes for the distributed algorithms to compute the partition, from start to finish. **Data Shipment** measures the total size of the messages that are transferred between machines. **Visit Times** measures the number of messages sent between machines, indicating the complexity of interactions. It is in our interest to minimize all three of these metrics. However, by minimizing one we inevitably increase another. Therefore we have to find a trade-off which balances the three.

3. SAMPLING METHODS

As indicated in Section 1 our first approach is to sample the graph and measure how well the bisimulation structure is preserved. For our experiments we used five representative sampling techniques: Random Node (RN), Random Edge (RE), Random Walk with Teleport (RWT), Breadth-First Explore (BFE), and Distinct Label Breadth-First Explore (DLBFE). These sampling techniques are illustrated in Figure 3 on the graph of Figure 1.

In Random Node (**RN**) sampling, we select a number of nodes randomly uniformly, and obtain the sample graph by keeping the edges between those nodes. This is a very common graph sampling technique and is used as a baseline comparison. This method has a time complexity of $O(|V|)$ for the sampling, and $O(|S| + deg(S))$ for the subgraph induction; by $deg(S)$ we denote the sum of the degrees of the nodes in the sample set S.

In Random Edge (**RE**) sampling, we select a number of edges randomly uniformly, and the nodes attached to them are kept. We obtain the sample graph by keeping the edges between those nodes. This is yet another very common graph sampling technique, which serves as a baseline with higher expectations compared to RN sampling. This method has a time complexity of $O(|E|)$ for the sampling, and $O(|S| + deg(S))$ for the subgraph induction.

In Random Walk with Teleport (**RWT**) sampling we explore the graph by means of visiting a randomly chosen outgoing neighbor of the selected node. There is also a small probability that we teleport, i.e. choose a completely random node as our next node instead of choosing an outgoing neighbor. If there are no outgoing edges from the selected node, we also teleport to a randomly chosen node.

We continue doing this until we have visited some desired number of nodes. We obtain the sample graph by keeping the nodes that we visited and the edges between them. In our study we do not allow duplicates in the set of sampled nodes and pick nodes with replacement, which implies an unbounded worst-case time complexity. For low sampling fractions such as the ones we use in our experiments below, however, it is expected that the sampling method ends in reasonable time.

In Breadth-First Explore (**BFE**) we visit nodes by exploring the graph in a breadth-first manner. We continue to visit nodes until we have visited some desired number of nodes. We obtain the sample graph by keeping the nodes that we visited and the edges between them. Because bisimilarity depends very much on the outgoing edges of a node, we expect that this sampling technique will yield good results. This method has a time complexity of $O(|V| + |E|)$ for the sampling, and $O(|S| + deg(S))$ for the subgraph induction.

In Distinct Label Breadth-First Explore (**DLBFE**) we also do BFE, but we do not select every outgoing neighbor of a node and add it to the queue of nodes to visit. Instead, we add only one node for every distinct pair of node-

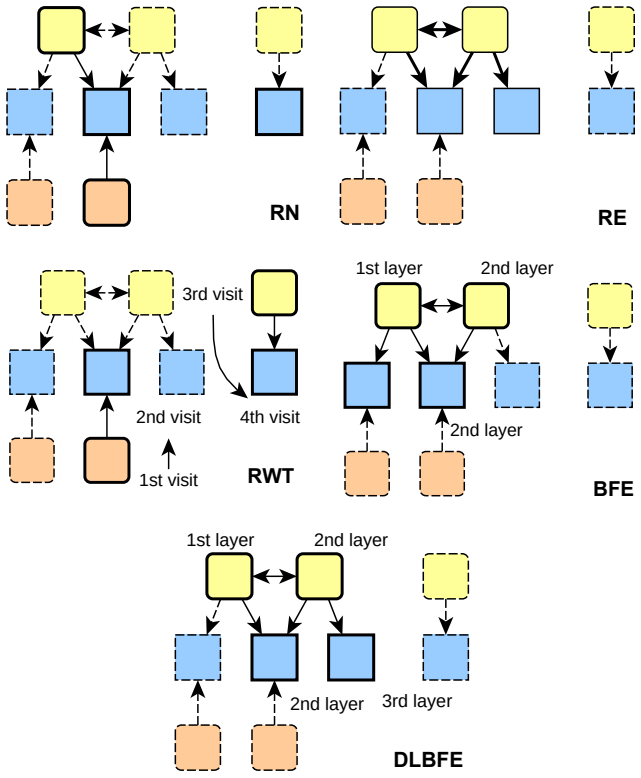


Figure 3: Illustration of 5 sampling methods on the example of Figure 1. Bold lines indicate selected nodes or edges; solid lines – edges or nodes in the induced subgraph; and, dashed lines – nodes and edges that are not in the sample.

and edge-label to the queue. We continue doing this until we have visited some desired number of nodes. We obtain the sample graph by keeping the nodes that we visited and the edges between them. In an attempt to improve on the BFE sampling technique, we apply this distinct label restriction as a way of "looking ahead" one step. This method has a time complexity of $O(|V|+|E|)$ for the sampling, and $O(|S|+deg(S))$ for the subgraph induction.

4. APPROXIMATE GRAPH PARTITIONING

Algorithms which compute the k-BSP of a graph often iteratively refine an initial partition [16]. We focus on those refinement algorithms which use a signature to represent a node's partition blocks. In each refinement step a signature is computed for every node, representing the partition block it belongs to. These distinct signatures are then given unique identifiers to simplify the computation in the next iteration. For completeness sake we will also discuss the exact algorithm by which the approximate algorithm was inspired.

Exact algorithm. Algorithm 1 shows pseudocode for an exact k-BSP algorithm based on signatures. The algorithm requires a mapping from signatures to identifiers, which is represented by the variable ID (line 1). First the algorithm computes the 0-BSP (lines 2–5). By Definition 2 it is easy to see that the $k=0$ partition only depends on the labels of the nodes. As such, all we do is compute a simple signature with just the node label, and have L be the empty set. Then the algorithm refines the initial partition until it is stable (lines

Algorithm 1: Exact k-bisimulation

```
1  ID ← ∅
2  foreach u ∈ V do
3  |  sig ← (λ_V(u), ∅)
4  |  Insert or get value for ID(sig)
5  |  P_new[u] ← ID(sig)
6  end
7  repeat
8  |  P_old ← P_new
9  |  foreach u ∈ V do
10 |  |  L ← {(λ_E(e), P_old[v]) | u -e-> v}
11 |  |  sig ← (λ_V(u), L)
12 |  |  Insert or get value for ID(sig)
13 |  |  P_new[u] ← ID(sig)
14 |  end
15 until P_new = P_old
```

Algorithm 2: Approximate k-bisimulation

```
1  foreach u ∈ V do
2  |  sig ← (λ_V(u), ∅)
3  |  P_new[u] ← ħ(sig)
4  end
5  repeat
6  |  P_old ← P_new
7  |  foreach u ∈ V do
8  |  |  L ← {(λ_E(e), P_old[v]) | u -e-> v}
9  |  |  sig ← (λ_V(u), L)
10 |  |  P_new[u] ← ħ(sig)
11 |  end
12 until P_new = P_old
```

7–15). It does this by iteratively computing the signatures of all the nodes in the graph. Checking whether or not the previous partition and the current partition are equal (line 15) can be done by comparing the number of partition blocks, since a stable partition won't gain more partition blocks if it is refined further. This algorithm has a worst-case time complexity of $O(k\cdot(|V|^2+|E|))$, where k is the number of iterations of the refinement step (lines 7–15).

Approximate algorithm. Assigning identifiers to distinct signatures can be a costly process, as they would be compared to check for equivalence. Our approximate algorithm does not check for equality among signatures, but instead assigns a value computed by hashing the signature as its identifier. This may lead to hash collisions, meaning that although two nodes have distinct signatures, they may be assigned to the same partition block. As such the approximate algorithm has better efficiency, but does not guarantee correctness. It goes without saying that the quality of the approximate algorithm is highly dependent on the hash function chosen. In Algorithm 2 we denote the hash function by $\hbar$. In our experiments we implemented the hash function $\hbar$ using the CRC32 algorithm. For our datasets this hash function was of good enough quality to achieve a 100% accuracy with our approximate algorithm. That is: the approximate algorithm computed the exact k-BSP of our datasets using this function. This algorithm has a worst-case time complexity of $O(k\cdot(|V|+|E|))$, where k is the number of iterations of the refinement step (lines 5–12).

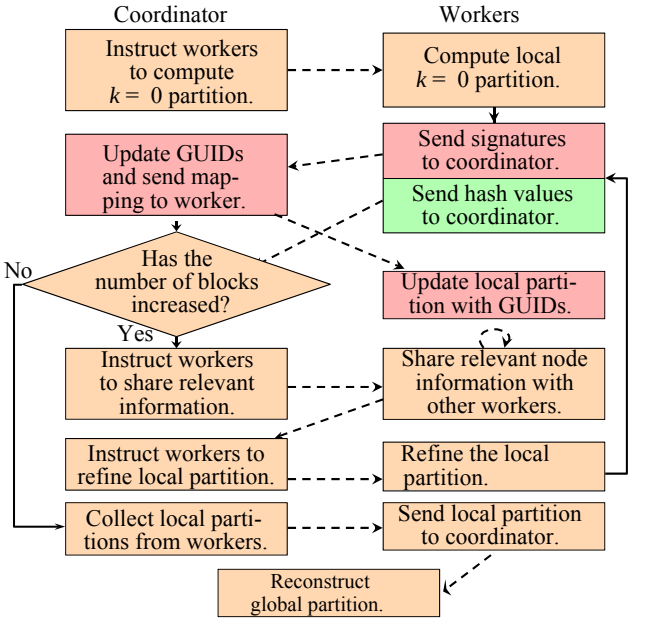
Distributed algorithms. As indicated in Section 1 our third approach is to distribute the workload of computing the BSP. In this section we will describe the distributed variants of the two previously described algorithms. To model our distributed algorithms we consider a single coordinator machine, and W worker machines. The coordinator machine will mostly do administrative work, whereas the worker machines will be computing partial partitions. In practice the coordinator machine could also be run as a separate thread on one of the worker machines. It is assumed that the nodes of the graph are spread evenly among the worker machines (that is: the difference between the number of nodes stored on each machine is minimal) and that there are at least as many nodes as there are workers (that is: $|V| \geq W$).

The problem for the distribution of BSPing is that in order to compute the partition block of a local node, information about its outgoing neighbors is required (which may or may not reside on the same machine). For example: suppose that our example graph from Figure 1 is split among three machines $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{U}$ which store the Author, Book, and User nodes respectively. Suppose we want to compute the exact BSP. In order to compute the signature of node Author C, it is necessary that machine $\mathcal{A}$ receives the signature of node Book 4 from the previous iteration. We solve this by adding a step to the refinement procedure of the previous algorithms which involves the sharing of local partition information between machines. Machines will only communicate the partition block of a node to another machine if there is an edge to the local node from a node on the other machine. In our example this means that $\mathcal{B}$ will send information about Book 4 to $\mathcal{A}$, but $\mathcal{A}$ will not send information about Author C to $\mathcal{B}$. Once this sharing step has been completed, local partition refinement can continue as normal.

Both of the distributed algorithms have been implemented using an event-driven approach; information between machines is shared via message passing. We have chosen for this approach because it is commonly used for the design of distributed algorithms. Despite this event-driven approach the behavior of each machine can be depicted nicely with a flowchart, and as such we have done so for each of our distributed algorithms.

Algorithm 3 shows the flowchart for a **distributed exact k-BSP** algorithm based on signatures. The routines used by the exact algorithm (but not taken by the approximate algorithm, discussed below) are colored red. The algorithm starts with the coordinator giving the signal to compute the $k = 0$ partition to each worker. Note that the $k = 0$ partition only requires knowledge of a node's label, and not of its outgoing edges. Hence, no sharing needs to have happened to compute this partition. The more difficult part about the distributed exact algorithm is that all the worker machines need to agree on how the signatures map to identifiers. Recall from Algorithm 1 that we could use a mapping on a single machine to give a new identifier for each new signature. We can then solve this by keeping a mapping on the coordinator machine, and letting the coordinator assign identifiers to signatures. These are called Globally Unique Identifiers (**GUIDs**). After receiving signatures from a worker machine, the coordinator decides on a GUID for each signature (if it didn't exist before), and lets the worker machine know about the mapping so it can also update the identifiers locally. This way, when sharing partition blocks between two workers, it is guaranteed that they agree on the identifiers

Algorithm 3: Distributed k-bisimulation



An illustration of the distributed algorithms, where messages between machines are indicated with dashed lines. Solid lines represent the continuation of control within one machine. Being event driven, the arrival of messages can lead to routines (rectangles) being executed by a machine. Background of rectangles is red if routine is used by the exact algorithm, green if by the approximate algorithm and orange for the common routines used by both algorithms.

of each signature. Recall that we measure the complexity of our distributed algorithms in terms of Makespan, Data Shipment, and Visit Times. Including the coordinator and all workers, the distributed exact algorithm has a Makespan complexity of $O(k \cdot (|E| + |V|^2))$, a Data Shipment complexity of $O(k \cdot |V|^2)$, and a Visit Times complexity of $O(k \cdot W^2)$.

Algorithm 3 also shows the flowchart for a **distributed approximate k-BSP** algorithm based on hash values. The routines used by the approximate algorithm (but not by the exact algorithm, discussed above) are colored green. As with the exact solution, the approximate algorithm starts with the coordinator signaling all workers to compute the $k = 0$ partition. Since all of the worker machines use the same hash function to compute the partition block for the nodes, it is not necessary for the coordinator to manage this globally, as was the case for the distributed exact algorithm. This saves on a lot of computation on the coordinator machine. The coordinator needs to know how many unique partition blocks there are after an iteration, therefore the hash values are communicated to it. Then it can simply decide to continue refining based on the current number of partition blocks and the previous number of partition blocks. Including the coordinator and all workers, the distributed approximate algorithm has a Makespan complexity of $O(k \cdot (|E| + |V|))$, a Data Shipment complexity of $O(k \cdot W \cdot |V|)$, and a Visit Times complexity of $O(k \cdot W^2)$.

5. EMPIRICAL STUDY

We empirically evaluate the accuracy of the sampling al-

Table 1: Overview of datasets used.

id	$\|V\|$	$\|E\|$	k_{max}	$\|\mathcal{P}_{\frac{1}{2}}\|$	$\|\mathcal{P}\|$	D	L's
1	26,475	53,381	7	16	5,060	13	-
2	5,242	14,496	34	1,092	1,111	16	-
3	9,877	25,998	30	2,443	2,459	20	-
4	9,000	7,500	5	30	60	5	N,E
5	27,770	352,807	8	3,352	20,093	37	-
6	10,000	9,956	15	1,924	2,204	91	-
7	484,610	>1M	3	34	228	4	E
8	608	1,828	3	4	19	5	E
9	2,625	100,000	1	1	12	1	E
10	6,301	20,777	7	14	2,167	20	-
11	7,844	9,358	76	1,607	2,158	120	N
12	7,220	112,139	4	2,741	4,444	10	N,E
13	7,115	100,762	8	331	4,103	10	-

gorithms (testing our first approach) and the performance of the partition algorithms (testing our second and third approaches). Here we discuss representative results of our experiments. Interactive exploratory tools for digging deeper into our detailed results can be found online.[2]

In Table 1 we summarize the following characteristics of the used datasets:

- $|V|$, $|E|$: the size of the graph w.r.t. nodes and edges,
- k_{max}: the value for k which produces the coarsest stable BSP,
- $|\mathcal{P}_{\frac{1}{2}}|$ and $|\mathcal{P}|$: the size of the partition (in terms of block count) for $\frac{1}{2}k_{max}$ and k_{max} respectively,
- D: the diameter of the graph (longest shortest path),
- L's: indicates whether nodes and/or edges are labeled.

Dataset id's correspond to the same row numbers in each of the table with results. Datasets 1 (AsCaida), 2 (CA-GrQc), 3 (CAHepTh), 5 (CitHepTh), 11 (P2pGnutella08), 12 (WikiElec), and 13 (Wikivote) represent computer, citation, collaboration networks and are available from the SNAP repository.[3] Datasets 4 (Chains), 6 (ErdosRenyi), and 8 (LDAG) are synthetically generated. Dataset 7 (Jamendo[4]) is an RDF graph and dataset 9 (MovieLens100k[5]) is a bipartite graph.

5.1 Experiments: Sampling

Set up. In order to evaluate the quality of the sampling algorithms in maintaining bisimulation structure, we measure how accurately the sampled graph has maintained the partition blocks of the original graph. As we vary the sampling fraction α with the values 1%, 2%, 4%, 8%, 16%, 32% and 64%, we measure the correctness and coverage of the sample with respect to the original graph. As we explained before, in some cases it matters how many nodes are in the partition blocks that are maintained, hence we also measure the weighted variants of these two metrics. Lastly we vary the k value of BSPing from 0 to k_{max} where k_{max} depends on the dataset.

Correctness C and coverage R. In Figure 4 we look at C and R of two sampling methods on the LDAG dataset.

[2]See http://bit.ly/1iUXXoA and http://bit.ly/1k3k9Au
[3]http://snap.stanford.edu/data/
[4]http://dbtune.org/jamendo/
[5]http://grouplens.org/datasets/movielens/

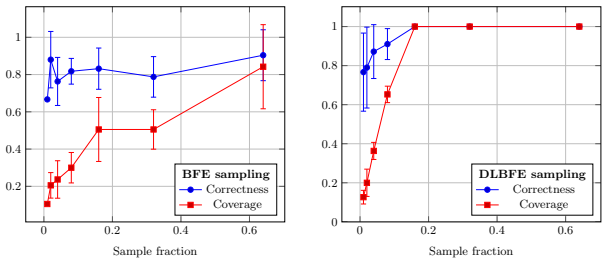


Figure 4: Correctness and coverage of k_{max}-BSPs of the sampled LDAG dataset graph as sample fraction α changes; averaged over 10 runs, error bars show SD.

As we can clearly see, DLBFE outperforms BFE on this dataset. Both sampling methods, however, are unable to achieve good coverage at lower sampling fractions α.

Because our result space is so large we freeze α to 8% and observe C and R for different combinations of datasets and samplers. In Table 2 we show C, R and WR for these combinations. Once again we see that DLBFE is better than BFE for the LDAG dataset, for both correctness and coverage. We can see a large difference between the plain and weighed coverage values. Thus it matters a lot whether one is interested in preserving the partition block count alone, or if the amount of nodes in those blocks matters.

As mentioned in Section 2, for some applications, such as graph database indexing, it makes more sense to look at limited k-BSP. Therefore, we also show the coverage when we compare partitions of $\frac{1}{2}k_{max}$-bisimulation. Naturally it is easier to preserve these partition through sampling than it is for k_{max} partitions, because the partitions are less refined.

In general, from these tables we can gain insight into what type of sampling works best for which kind of datasets. For example: for a bipartite graph like MovieLens100k it appears RE is the best sampling method, which makes sense as it samples both endpoints of the edges in a bipartite graph.

For a per-dataset per-algorithm analysis we refer to [21].

5.2 Experiments: Partitioning

Set up. In this section we empirically evaluate the performance of Algorithm 1 and Algorithm 2. We compare the Makespan (running time) to their distributed counterparts, of which we will also look at the Data Shipment and Visit Times behavior. Our experiments were run on a system with two Intel(R) Xeon(R) E5-2630 CPUs at 2.30 GHz with 64.0 GB memory running Windows 7 Enterprise Service Pack 1. Every measurement we present is from an average of 10 runs and we show the standard deviation as error bars.

For the distributed algorithms it matters how the graphs are split among the worker machines. This in itself is a rich topic of study [3, 22]. The goal is to minimize the number of edges between nodes on different machines, while keeping the number of nodes per machine balanced. During our experiments we have tested two methods: random split and exploration split. Random split simply distributes the nodes of a graph randomly among the worker machines. Exploration split starts with seed nodes and explores the graph from those nodes; as the graph is explored the nodes are subsequently added to a machine until we reach roughly $\frac{|V|}{W}$ nodes per machine. Exploration split generally gave the lowest Makespan, Data Shipment and Visit Times, and this

Table 2: Heat maps of P, R, and WR over 10 runs of the sampling algorithms at k-bisimulation with $\alpha = 8\%$.

Average correctness P, $k = k_{max}$

Data set	RN	RE	RWT	BFE	DLBFE
AsCaida	0.69	0.02	0.11	0.61	0.05
CAGrQc	0.93	0.19	0.56	0.66	0.65
CAHepTh	0.91	0.11	0.53	0.57	0.58
Chains	1	1	1	1	1
CitHepTh	0.16	0.02	0.06	0.08	0.06
ErdosRenyi	1	0.93	0.66	0.98	0.64
Jamendo	0.01	0	0.02	0.94	0.9
LDAG	0.52	0.37	0.23	0.82	0.91
MovieLens100k	0.48	1	0.71	1	1
P2pGnutella08	0.54	0	0.05	0.04	0.05
Petrinet	0.48	0.29	0.3	0.56	0.38
WikiElec	0.06	0.03	0.03	0.02	0.02
WikiVote	0.08	0	0.01	0.01	0.01

Average coverage R, $k = k_{max}$

Data set	RN	RE	RWT	BFE	DLBFE
AsCaida	0	0.01	0.01	0.03	0.01
CAGrQc	0.01	0.07	0.02	0.06	0.03
CAHepTh	0	0.04	0.01	0.04	0.02
Chains	0.37	0.73	0.99	1	1
CitHepTh	0	0.01	0	0.01	0
ErdosRenyi	0	0.01	0.01	0.1	0.01
Jamendo	0.01	0.04	0.04	0.3	0.3
LDAG	0.13	0.37	0.19	0.3	0.65
MovieLens100k	0.85	0.97	0.79	0.19	0.42
P2pGnutella08	0	0	0	0	0
Petrinet	0	0	0.01	0.05	0.02
WikiElec	0	0.02	0	0	0
WikiVote	0	0	0	0	0

Average weighted coverage WR, $k = k_{max}$

Data set	RN	RE	RWT	BFE	DLBFE
AsCaida	0.51	0.53	0.55	0.57	0.52
CAGrQc	0.72	0.79	0.76	0.78	0.77
CAHepTh	0.7	0.75	0.73	0.74	0.73
Chains	0.37	0.73	0.99	1	1
CitHepTh	0.26	0.27	0.26	0.26	0.25
ErdosRenyi	0.62	0.72	0.71	0.76	0.71
Jamendo	0.31	0.97	0.96	1	1
LDAG	0.09	0.32	0.23	0.29	0.64
MovieLens100k	1	1	1	0.93	0.98
P2pGnutella08	0.65	0.65	0.65	0.65	0.65
Petrinet	0.49	0.48	0.63	0.75	0.81
WikiElec	0.24	0.27	0.25	0.15	0.15
WikiVote	0.62	0.72	0.71	0.76	0.71

Average coverage R, $k = \frac{1}{2}k_{max}$

Data set	RN	RE	RWT	BFE	DLBFE
AsCaida	0.54	1	0.99	0.94	1
CAGrQc	0.01	0.06	0.02	0.07	0.03
CAHepTh	0	0.04	0.01	0.03	0.02
Chains	0.76	1	1	1	1
CitHepTh	0.05	0.5	0.14	0.16	0.12
ErdosRenyi	0	0.01	0.02	0.11	0.01
Jamendo	0.12	0.69	0.39	0.71	0.76
LDAG	0.55	1	0.88	0.93	1
MovieLens100k	1	1	1	1	1
P2pGnutella08	0.51	1	0.99	0.99	0.99
Petrinet	0	0	0.03	0.07	0.02
WikiElec	0.01	0.59	0.03	0.02	0.02
WikiVote	0.11	0.69	0.29	0.25	0.28

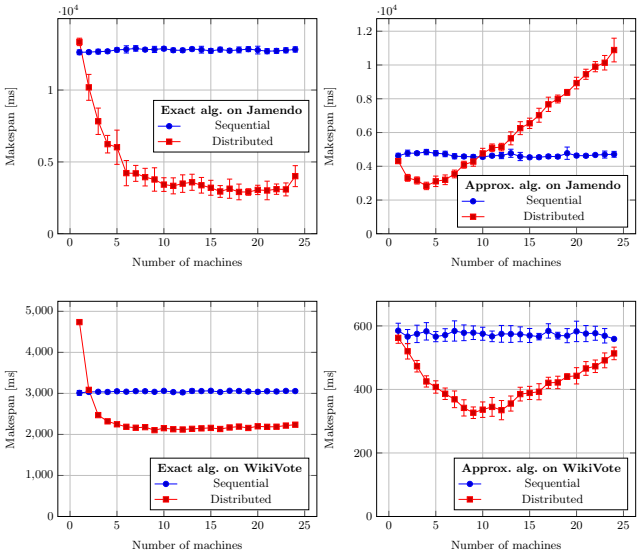


Figure 5: Makespan (in milliseconds) of the sequential and distributed algorithms for the computation of the k_{max}-BSP.

is the method we used for our results presented here.

Makespan. Figure 5 shows the Makespan of both the exact and the approximate algorithms on two representative datasets. For both datasets we see that the distributed algorithms is more efficient. However, having too many machines creates overhead. The number of machines at which distribution is not effective depends on the dataset, and how compute intensive the individual workloads for the workers are. We also see that our approximate algorithm outperforms the exact algorithm. Table 3 shows a summary of speedup values for the sequential algorithms. With the only exception being the MovieLens100k dataset, our approximate algorithm outperforms the exact algorithm.

Table 3: Makespan for the sequential exact and approximate algorithms to compute the k_{max}-BSP on all datasets.

Dataset	Exact [ms]	Approx. [ms]	Speedup
AsCaida	$1.70 \cdot 10^4$	$4.76 \cdot 10^2$	35.76
CAGrQc	$4.50 \cdot 10^3$	$4.45 \cdot 10^2$	10.11
CAHepTh	$1.48 \cdot 10^4$	$7.83 \cdot 10^2$	18.86
Chains	$1.08 \cdot 10^2$	$8.41 \cdot 10^1$	1.29
CitHepTh	$3.58 \cdot 10^4$	$2.25 \cdot 10^3$	15.91
ErdosRenyi	$1.37 \cdot 10^4$	$2.53 \cdot 10^2$	54.38
Jamendo	$1.26 \cdot 10^4$	$4.62 \cdot 10^3$	2.73
LDAG	$9.60 \cdot 10^0$	$6.30 \cdot 10^0$	1.52
MovieLens100k	$8.97 \cdot 10^1$	$1.10 \cdot 10^2$	0.82
P2pGnutella08	$1.10 \cdot 10^3$	$1.39 \cdot 10^2$	7.91
Petrinet	$5.24 \cdot 10^4$	$9.66 \cdot 10^2$	54.23
WikiElec	$1.87 \cdot 10^3$	$3.47 \cdot 10^2$	5.39
WikiVote	$3.01 \cdot 10^3$	$5.85 \cdot 10^2$	5.15

Data shipment, visit times and approximation accuracy. Figure 6 shows the Data Shipment and Visit Times for the MovieLens100k dataset. As expected, the Visit Times for the approximate algorithm is lower than that of the exact algorithm, since it uses strictly fewer messages per refinement step. The amount of information shipped is also less.

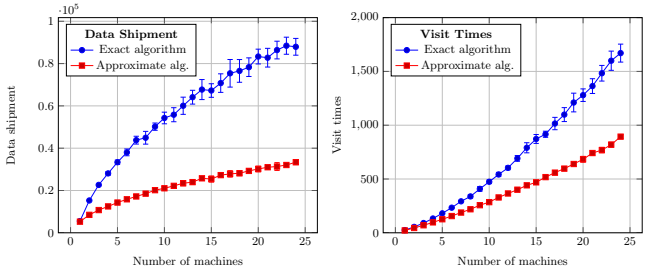


Figure 6: Data Shipment and Visit Times for the k_{max}-BSP on the MovieLens100k dataset.

Although the MovieLens100k dataset was the only dataset for which the exact algorithm is faster than the approximate algorithm, the latter still uses less communication.

Now that we have demonstrated its efficiency it really comes down to how accurate the approximate algorithm is. We have tested a number of ways to compute the hash value of a signature, such as XOR-ing and adding the node/edge label values. With the CRC32 hash algorithm we were able to achieve 100% accuracy on all our datasets (that is: the approximate algorithm gave the same partition as the exact algorithm). It is still an interesting research issue to explore which hash functions are best suited for certain graphs.

6. CONCLUDING REMARKS

In this paper we investigated possibilities for structure-preserving reduction of graphs, with a focus on bisimulation structure as a representative case study. Bisimulation is a fundamental structural notion arising in many areas of investigation, such as social network analysis. The algorithms we have developed here can easily be adapted to compute partitions of graphs under other equivalence relations.

Our first approach was to sample the graph using traditional sampling methods. We compared the quality of the sampling methods in terms of novel (un)weighted correctness and coverage metrics for a variety of datasets. We varied the boundedness (k) of bisimulation as well as the sampling fraction. Overall we saw mixed results, where some samplers work well for certain types of datasets but not for others. This indicates the need for further study of sampling algorithms specifically for preserving topological structure.

We next introduced a new partitioning algorithm which sacrifices correctness guarantees for efficiency. We found that our approximate algorithm is indeed more efficient than the traditional exact algorithm, and with use of CRC32 to compute the hash value of a signature, it even achieves 100% accuracy on all datasets we experimented with.

Our last approach was to distribute the problem onto multiple machines. Since large graphs are often already stored on multiple machines, this is also an interesting solution. An existing distributed exact algorithm was compared to our proposed novel distributed approximate algorithm on three basic performance metrics. Once again, the approximate algorithm yielded better performance results than the exact algorithm. We do note, however, that increasing the number of machines endlessly has a diminishing returns effect, and at some point the overhead is not worth the gain.

In summary, our results indicate that high quality structure-preserving reduction of large graphs is indeed possible in practice. Looking ahead to further research, our work here raises the fundamental question of the meaning of (global) "structure" in graphs. Investigating other notions of structure beyond bisimulation would be fruitful.

Acknowledgments. Many thanks to Rui Castro, Herman Haverkort and Sander Leemans for their help.

7. REFERENCES

[1] L. Aceto, A. Ingolfsdottir, and J. Srba. The algorithmics of bisimilarity. In *Advanced Topics in Bisimulation and Coinduction*, pages 100–172. 2011.

[2] N. K. Ahmed, J. Neville, and R. Kompella. Network sampling: From static to streaming graphs. *ACM TKDD*, 8(2):7:1–7:56, 2013.

[3] K. Andreev and H. Räcke. Balanced graph partitioning. In *SPAA 2004*, pages 120–124.

[4] S. Blom and S. Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. *STTT*, 7(1):74–86, 2005.

[5] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *STOC 2011*, pages 71–80.

[6] J. F. Groote and F. van Ham. Interactive visualization of large state spaces. *IJSTTT*, 8(1):77–91, 2006.

[7] R. Jin et al. Scalable and axiomatic ranking of network role similarity. *ACM TKDD*, 8(1):3, 2014.

[8] D. Koutra et al. VOG: summarizing and understanding large graphs. In *SDM 2014*, pages 91–99.

[9] K. LeFevre and E. Terzi. GraSS: Graph structure summarization. In *SDM 2010*, pages 454–465.

[10] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD 2006*, pages 631–636.

[11] Y. Luo et al. External memory k-bisimulation reduction of big graphs. In *CIKM 2013*, pages 919–928.

[12] Y. Luo et al. Regularities and dynamics in bisimulation reductions of big graphs. In *GRADES 2013*, pages 13:1–13:6.

[13] S. Ma, Y. Cao, J. Huai, and T. Wo. Distributed graph pattern matching. In *WWW 2012*, pages 949–958.

[14] M. Marx and M. Masuch. Regular equivalence and dynamic logic. *Social Networks*, 25(1):51–65, 2003.

[15] National Research Council. *Frontiers in Massive Data Analysis*. The National Academies Press, 2013.

[16] M. Newman. *Networks: An Introduction*. OUP Oxford, 2010.

[17] F. Picalausa et al. Principles of guarded structural indexing. In *ICDT 2014*, pages 245–256.

[18] M. Riondato et al. Graph summarization with quality guarantees. In *ICDM 2014*, pages 947–952.

[19] R. Rossi and N. Ahmed. Role discovery in networks. *IEEE TKDE*, 27(4):1112–1131, 2015.

[20] S. Soundarajan et al. A guide to selecting a network similarity method. In *SDM 2014*, pages 1037–1045.

[21] W. van Heeswijk. Structure preserving graph sampling and methods for partitioning graphs, MSc Thesis, Eindhoven University of Technology, 2014.

[22] L. Wang et al. How to partition a billion-node graph. In *ICDE 2014*, pages 568–579.