Subject: Final Version - Re: ICGT'23 :: full acceptance of paper
From: Ansgar Scherp <ansgar.scherp@uni-ulm.de>
Date: 11.05.23, 21:37
To: ICGT 2023 <icgt2023@easychair.org>
CC: "Richerby, David M" <david.richerby@essex.ac.uk>, rau-jannik@gmx.de

Dear Chris and Maribel

Thank you for the very good news!

We are very happy that the reviewers acknowledge the efforts put into the paper and accepted it for ICGT 2023.

Regarding this follow-up comment the reviewer had, please forward our response:

Question: it seems that the authors didn't understand: >>> (Section 3.3) The definition of id and T above (1) seems incorrect. They are not used as equivalence over vertices. id and T are defined here as binary relations over *vertices* and then used as the second component of CSE which was defined (on the same page) to be an equivalence relation over *edge labels*.

Answer: There is a conceptual misunderstanding in how labeled graphs are represented in triples. The labels are modeled as vertices, i. e., with the triples (s, p, o) we have vertices at each position (the subject, predicate, and object). This can be best illustrated in the context of web graphs such as RDF, where the subject, predicate, and object of a triple are URIs, i. e., identifiers on the web. In a formal definition of a labeled graph, the predicates will "appear" as vertices, as we define it. (Indeed, vertices that appear as predicates/labels in one triple might appear in subject or object position in other triples.) Thus, id and T are indeed equivalence relations defined over vertices.

We hope this finally clarifies your questions. We also addressed the other comments.

If the reviewers wish to discuss this further with us at the conference, we are happy to do so.

In any case, we are very happy to come to Leicester to present our work.

On behalf of all authors,

Ansgar

PS: We also updated the paper on EasyChair.

Subject: ICGT'23 :: full acceptance of paper
From: "ICGT 2023" <icgt2023@easychair.org>
Date: 09.05.23, 19:10
To: Ansgar Scherp <mail@ansgarscherp.net>

Dear Ansgar Scherp,

Thank you very much for your efforts in revising your ICGT'23 submission, "Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis".

Following a round of checks by your referees, we are pleased to share that your paper is **accepted** (unconditionally) to the conference. We are looking forward to seeing you at ICGT'23!

Your referees had a couple of follow-up comments that they would like to bring to your attention. Please take them into consideration, and if you would like to make changes to your camera-ready sources, we are happy to give you until the end of Thursday 11th May (AoE) to do so (under Easychair --> ICGT'23 --> proceedings author).

Registration is now open via the STAF'23 website (https://conf.researchr.org/attending/staf-2023/registration). Do note the early bird deadline of 14th June, and that ICGT'23 requires at least a 2-day ticket.

Best wishes,
Chris and Maribel

****
ADDITIONAL COMMENTS
****

it seems that the authors didn't understand:

> (Section 3.3) The definition of id and T above (1) seems incorrect. They are not used as equivalence over vertices.

id and T are defined here as binary relations over ***vertices*** and then used as the second component of CSE which was defined (on the same page) to be an equivalence relation over ***edge labels***.

****

concerning the remark above: yes, indeed the second component of their schema must be a relation on edge labels - rather than vertices - so using id respectively T in the middle of a three-tuple seems wrong. This is easy to fix - by taking the identity on edge labels respectively the relation containing all pairs of edge labels - but it has to be done.

I have another issue: on the same page (p.6) the authors explain what a summary graph is:
"This results in a summary graph that has one vertex for each equivalence class in each equivalence relation defined within the CSE. Let C_i and C_i+1 be equivalence classes in the i- and (i + 1)-bisimulation relations. There is an edge with
label p between C_i and C_i+1 if there is an edge with that label between some u ∈ C_i and v ∈ C_i+1."

This confuses me a bit: there are edges between equivalence classes of subsequent relations, but not between equivalence classes of the same relation? I would have expected that you take the equivalence classes of the k-bisimulation and then connect two classes iff there is an edge between its nodes. (I.e., the original graph is quotiented by k-bisimulation.)

****

Response letter to the Conditional Acceptance of "Computing k-Bisimulations for Large Graphs" to ICGT 2023

Dear Chris and Maribel,

Dear anonymous reviewers,

Thank you for carefully reading our paper on "Computing k-Bisimulations for Large Graphs". We highly appreciate the valuable comments and suggestions made.

We are happy to state that we have addressed all comments.

Below are specific responses to the four questions in the meta-review, and some additional responses to specific reviewers' points.

A PDF with the critical changes made is provided. The PDF marks the **changes in red** together **with comments on the changes**.

Response to the metareview:

Question: (Review #1) A more elaborate comparison to related work is needed to establish the novelty of the contribution. (Please see the review for some specific pointers.)

Answer: We revised the related work section and discussed the recommended literature and beyond. In summary, the existing approaches each propose a single, specific algorithm for computing a single graph summary based on a bisimulation. In contrast, we suggest a generic algorithm for computing k-bisimulation and any other equivalence definable in the FLUID language and show its advantages. Also, our approach allows the bisimulation model to be specified in a declarative way and parallelizes otherwise sequential algorithms like the one by Kaushik et al. making it efficient to compute.

Question: (Review #1) The algorithms, in particular the parallel BRS algorithm has to be defined more carefully. In particular it uses functions such as MergeAndHash and an oplus operation. What exactly are these operators? Are there any requirements on the hash function? This is all left quite open and unspecified. Also: I would have been interested in more details of the parallelization, which seems to be the main contribution. In particular: how does the Signal/Collect paradigm work and why is it suitable for what you are doing?

Answer: We have clarified the semantics of the mentioned function and operator. For the hash function, it has to be permutation-invariant w.r.t. to the input, which we also make clearer. In the algorithms section and the implementation section, we add details about the use of Signal/Collect for bisimulation.

Question: (Review #1) You are comparing the so-called Schätzle algorithm, which computes an edge-labeled forward k-bisimulation with the Kaushik algorithm determining a vertex-labeled backward k-bisimulation. First I wonder why these two variants: why is forward combined with edge labels and backward with vertex labels? Second: in my opinion it does not make too much sense to compare the runtimes of these two algorithm, simply because they compute different relations. It would have made more sense to compare algorithms which both compute edge-labeled/forward or vertex-labeled/forward or ...

Answer: We clarify the motivation of using Schätzle et al. and Kaushik et al. as they are two representative algorithms for the typical cases of backward vs. forward bisimulation and

bisimulation over vertex labels vs. edge labels. Please note, we do not directly compare the Schätzle et al. and Kaushik et al. algorithms, as they compute different things. Rather, we compare the implementation of the native Schätzle et al. algorithm with our generic counterpart, i.e., the BRS-Schätzle. Thus, we do compare what is requested by comparing a specific edge-label algorithm with another but generic algorithm for edge-labeled bisimulation. Likewise, we do so for the vertex-labeled Kaushik et al. algorithm. We made this now clearer in the paper.

Question: (Review #2) for k-bisimulation, the authors choose the k=10 case only. Without justification or analysis, this looks like an arbitrary choice, and one may wonder if 10 isn't too small (i.e., bisimulation doesn't summarize the graph well) and if 10 isn't too large (i.e., the same result could have been obtained with a smaller k). A proper evaluation should analyze the change of the size of the summary and the execution time, both with respect to k. In particular, it is important to see if k=10 is just a rough summary or close to full bisimulation.

Answer: We now provide a detailed analysis of the choice of k and the influence of k on the different bisimulation algorithms. This resulted in heavily rewriting the discussion. Note, we have computed the bisimulations for *all* values of k=1 to 10 and recorded the data per iteration. This could be seen already, e.g., in Figure 2, which is our key result, but we had to improve the presentation to make this clear. Thank you for pointing out this comment, it significantly helped us to improve our reflection and presentation of the paper. We have justified 10 as the largest value of k considered: essentially, it is not too big because we also consider all smaller values. It is not too small as, in the context of the web graphs that we consider for graph summarization, vertices that are 10-bisimilar are indistinguishable by paths of length 10, and information stored at an even greater distance from the vertex seems unlikely to be practically useful to distinguish them.

Responses to further questions:

Question: (Reviewer #1) - p.6, l.-6: Use double subscripts: B_{i1} -> B_{i_1}:

Answer: B_{ij} refers to the jth block of partition i, whereas B_{i_j} would refer to "block number i_j" for some variable i_j.

Question: (Reviewer #2) Firstly, the authors say that an advantage of the proposed algorithm is that it is parallelizable, but they do not show parallel speedup or scalability (with respect to the number of cores), which is very important for papers on parallel programs/algorithms for real data.

Answer: We demonstrate the scalability of the hash-based BRS algorithm with respect to the graph size and increasing value of k. Our experiments suggest that the algorithm scales linearly with respect to the two factors. In this paper, we used all cores available on the machine, i.e., 32 cores. The influence of the different numbers of cores on the parallelization of the base algorithm used for our hash-based BRS algorithm has been shown in the Ph.D. thesis of Till Blume, https://oparu.uni-ulm.de/xmlui/handle/123456789/46050.

Question: (Reviewer #2) (Section 3.2) This definition of bisimulation is not a standard/original one (which should be a relation between two labelled transition systems). Some remark is appropriate to avoid confusion.

Answer: For example, Kaushik et al. define bisimulation as an equivalence on vertices. We clarified this in the paper by adding a citation.

Question: (Reviewer #2) (Section 3.3) The definition of id and T above (1) seems incorrect. They are not used as equivalence over vertices.

Answer: We define a CSE (complex schema element) to take three equivalence relations as input. id and T define equivalence relations.

Question: (Reviewer #2) (Table 2) I don't understand why (in some graphs) r(l_V) is larger than |\Sigma_V| when l_V(v) is taken from \Sigma_V.

Answer: The r(l_V) is the number of subsets in \Sigma_V that are observed in the dataset.

Question: (Reviewer #2) (Algorithm 3) When an algorithm is presented, it is standard to discuss its correctness, termination, and complexity. Any results or comments on these?

Answer: We have added a statement of the complexity, citing the full version of the paper on ArXiv. Correctness is because the algorithm is a modification of one from [6]. Termination is clear, as each loop is bounded ("for each vertex", "for each neighbor" or "for each value in 1..k").

Question: (Reviewer #2) (Table 2) In some graphs, the standard deviation is so large and subtracting it from the mean would result in a negative value. Some comments would be appropriate.

Answer: There are some supervertices in the datasets with a very high degree (up to 1,000). This makes the standard deviation very high while the vast majority of the vertices have a (relatively) low degree.

Question: (Reviewer #2) (page 1) What does BRS stands for? Please spell out.

Answer: We added it. The original work is from Blume, Richerby, and Scherp and is extended by the first author, ie., Rau, of this paper.

Question: (Reviewer #2) (page 2, line 2) Can you give examples of arbitrary (but motivated)user-defined equivalence relations that can't be handed by k-bisimulation?

Answer: Any equivalence relation where vertices at different distances are treated differently would not be a k-bisimulation. For example, one could define a graph summary extending the one-hop model of SchemEX by Konrath et al. by considering a second hop, but where the second hop does not consider RDF types while the first hop does consider RDF types. For SchemEX, see https://www.sciencedirect.com/science/article/abs/pii/S1570826812000716

Question: (Reviewer #2) (page 5, line -2) Please explain what the suffixes (s, p, o) stand for.

Answer: In web graphs based on the W3C standard RDF it is common practice that *(s, p, o)* denotes a triple with two vertices s (=subject) and o (=object) connected via a predicate p. This became standard notation, i.e., the specific variable names could be different.

Question: (Reviewer #2) (page 6, end of Section 3.3) Examples or some explanation of (1) and(2) would be appropriate.

Answer: We clarified in the paper that Equations (1) and (2) follow from Definitions 1 and 2. We elsewhere justify the choice of the two bisimulation models of Schätzle et al. and Kaushik et al.

Furthermore, we fixed all typos and clarified smaller questions, e.g., rounded rectangles, “combines of”, etc that Reviewer #2 had.

Once again, we like to thank you for your thoughtful and thorough review. Your comments have been insightful, and your suggestions have been invaluable. We hope that you will find the revised manuscript to be an improvement over the original, and we look forward to your continued feedback.

We will release the source code upon acceptance of the paper.

Sincerely,

Jannik Rau (student author), David Richerby, and Ansgar Scherp
Ulm, May 1st, 2023

Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis

Jannik Rau[1], David Richerby[2][0000−0003−1062−8451], and Ansgar Scherp[1][0000−0002−2653−9245]

[1] University of Ulm, Germany. {firstname.lastname}@uni-ulm.de
[2] University of Essex, UK. david.richerby@essex.ac.uk

Abstract. Summarizing graphs w.r.t. structural features is important to reduce the graph's size and make tasks like indexing, querying, and visualization feasible. Our generic parallel BRS algorithm efficiently summarizes large graphs w.r.t. a custom equivalence relation $\sim$ defined on the graph's vertices V. Moreover, the definition of $\sim$ can be chained $k \geq 1$ times, so the defined equivalence relation becomes a k-bisimulation. We evaluate the runtime and memory performance of the BRS algorithm for k-bisimulation with $k = 1, \ldots, 10$ against two algorithms found in the literature (a sequential algorithm due to Kaushik et al. and a parallel algorithm of Schätzle et al.), which we implemented in the same software stack as BRS. We use five real-world and synthetic graph datasets containing 100 million to two billion edges. Our results show that the generic BRS algorithm outperforms the respective native bisimulation algorithms on all datasets for all $k \geq 5$ and for smaller k in some cases. The BRS implementations of the two bisimulation algorithms run almost as fast as each other. Thus, the BRS algorithm is an effective parallelization of the sequential Kaushik et al. bisimulation algorithm.

Keywords: structural graph summarization · bisimulation · large labeled graphs.

1 Introduction

Storing, indexing, querying, and visualizing large graphs is difficult [7]. One way to mitigate this challenge is *graph summarization* [9]. Graphs can be summarized w.r.t. so-called *graph summary models* [6] that define structural features (e. g., incoming/outgoing paths), statistical measures (e. g., occurrences of specific vertices), or frequent patterns found in the graph [9]. This gives a *summary graph*, which is usually smaller than the original but contains an approximation of or exactly the same information as the original graph w.r.t. the selected features of the summary model. Tasks that were to be performed on the original graph can be performed on the summary but much faster. Use cases are optimizing database queries [22], data visualization [12], and OWL reasoning [28].

Blume, Richerby, and Scherp developed a generic structural summarization approach, here referred to as BRS [5, 6]. The BRS algorithm summarizes an

input graph w.r.t. an arbitrary user-defined equivalence relation specified in its formal language FLUID [6]. The FLUID language supports all features of structural graph summarization found in the literature [6]. There are two groups of these features. The first comprises a vertex's *local* information, e. g., its label set, its direct neighbors, and the labels of its incoming or outgoing edges. The second group considers a vertex's *global* information at distance $k > 1$. This includes, e. g., local information about reachable vertices up to distance k or information about incoming or outgoing paths of length up to k. We use stratified k-bisimulations (formally described in Section 3.2) to summarize a graph w.r.t. global information and group together vertices that have equivalent structural neighborhoods up to distance k. Several existing approaches use k-bisimulations to incorporate global information into structural graph summarization [6, 9]. The BRS algorithm generalizes these approaches and can chain any definable equivalence relation k times, such that the resulting equivalence classes can be efficiently computed by global information up to distance k [4, 24].

However, it is not known if a general approach like the BRS algorithm sacrifices performance. We choose two representative algorithms as examples to demonstrate the capabilities of our generic BRS algorithm. First, we have reimplemented the efficient, parallel single-purpose k-bisimulation algorithm of Schätzle, Neu, Lausen, and Przjaciel-Zablocki [25] and investigate whether it outperforms our generic BRS algorithm. Second, we investigate the sequential algorithm for bisimulation by Kaushik, Shenoy, Bohannon, and Gudes [16]. Being sequential, it is naturally disadvantaged against parallel algorithms such as ours. However, we show in this work that the bisimulation of Kaushik et al. [16] can be declaratively specified and executed in the generic BRS algorithm. This effectively parallelizes the algorithm "for free". We evaluate the performance of the BRS-based parallelized computation of the Kaushik et al. graph summary model and compare it with their sequential native algorithm. We also compare both Kaushik et al. variants, native and BRS-based, with the parallel native algorithm of Schätzle et al. [25] and a BRS implementation of Schätzle et al. (see also [4]). Thus, we have four k-bisimulation algorithms. For a fair comparison, we reimplemented the existing native algorithms in the same graph processing framework as the BRS algorithm. We execute the four algorithms on five graph datasets – two synthetic and three real-world – of different sizes, ranging from 100 million edges to billions of edges. We evaluate the algorithms' performance for computing k-bisimulation for $k = 1, \ldots, 10$. We measure running time per iteration and the maximum memory consumption.

The questions we address are: Do the native bisimulation algorithms have an advantage over a generic solution? How well do the native and generic algorithms scale to large real-world and synthetic graphs? Is it possible to effectively scale a sequential algorithm by turning it into a parallel variant by using a general formal language and algorithm for graph summaries?

We discuss related work next. Section 3 defines preliminaries, while the algorithms are introduced in Section 4. Section 5 outlines the experimental apparatus. Section 6 describes the results, and these are discussed in Section 7.

2 Related Work

Summary graphs can be constructed in several ways. Čebirić et al. [9] classify existing techniques into structural, pattern-mining, statistical, and hybrid approaches. In this paper, we consider only structural approaches based on quotients. Other structural summarization techniques, not based on quotients, are extensively discussed by Čebirić et al. [9]. Structural approaches summarize a graph G w.r.t. an equivalence relation $\sim \subseteq V \times V$ defined on the vertices V of G [7,9]. The resulting summary graph SG consists of vertices VS, each of which corresponds to a equivalence class of the equivalence relation $\sim$.

Response to Reviewer #1 with a more elaborate comparison to related work, and edge- vs vertex-based bisimulation.

One can observe that k-bisimulation is a popular feature for structural graph summarization [6]. Bisimulation comes in three forms: backward k-bisimulation classifies vertices based on incoming paths of length up to k, forward bisimulation considers outgoing paths, and backward-forward bisimulation considers both. Bisimulation may be based on edge labels, vertex labels, or both, but this makes no significant difference to the algorithms. A notion of k-bisimulation w.r.t. graph indices is introduced by seminal works such as the k-RO index [21] and the T-index summaries [20]. Milo and Suciu's T-index [20], the $A(k)$-Index by Kaushik et al. [16], and others are examples that summarize graphs using backward k-bisimulation. We chose as representative the sequential algorithm by Kaushik et al. [16], which uses vertex labels, as described in Section 4.2. Conversely, the k-RO index, the Extended Property Paths of Consens et al. [11], the SemSets model of Ciglan et al. [10], Buneman et al.'s RDF graph alignment [8], and the work of Schätzle et al. [25] are based on forward k-bisimulation. We note that Schätzle et al. use edge labels, as described in Section 4.1. Tran et al. compute a structural index for graphs based on backward-forward k-bisimulation [27]. Moreover, they parameterize their notion of bisimulation to a forward-set L_1 and a backward-set L_2, so that only labels $l \in L_1$ are considered for forward-bisimulation and labels $l \in L_2$ for backward-bisimulation.

Luo et al. examine structural graph summarization by forward k-bisimulation in a distributed, external-memory model [18]. They empirically observe that, for values of $k > 5$, the summary graph's partition blocks change little or not at all. Therefore they state, that for summarizing a graph with respect to k-bisimulation, it is sufficient to summarize up to a value of $k = 5$ [17]. Finally, Martens et al. [19] introduce a parallel bisimulation algorithm for massively parallel devices such as GPU clusters. Their approach is tested on a single GPU with 24 GB RAM, which limits its use on large datasets. Nonetheless, their proposed blocking mechanism could be combined with our vertex-centric approach to further improve performance.

Each of the works proposes a single algorithm for computing a single graph summary based on a bisimulation. Some of the algorithms have bisimulation parameters such as the height and label parameterization in Tran et al. [27]. We suggest a generic algorithm for computing k-bisimulation and show its advantages. Also, our approach allows the bisimulation model to be specified in a declarative way and parallelizes otherwise sequential computations like in Kaushik et al. [16] into a parallel computation.

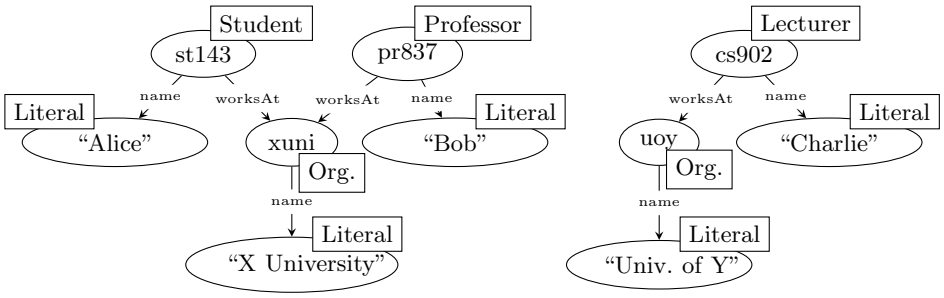


Fig. 1: An example graph $\boldsymbol{G}$ displaying two universities and three employees. Vertices are denoted by ellipses and edges by arrows. Vertex labels are marked with rectangles and edge labels are written on the edge.

3 Preliminaries

3.1 Data Structures

The algorithms operate on multi-relational, labeled graphs $G = (V, E, l_V, l_E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices and $E \subseteq V \times V$ is a set of directed edges between the vertices in V. Each vertex $v \in V$ has a finite set of labels $l_V(v)$ from a set Σ_V and each edge has a finite set of labels $l_E(u, v)$ from a set Σ_E.

Figure 1 shows an example graph representing two universities and three employees. Vertices are represented by ellipses and edges are labeled arrows. Vertex labels are shown in rectangles. For example, the edges (pr837, xuni) and (st143, xuni) labeled with *worksAt* together with edges (pr837, "Bob"), (st143, "Alice") and (xuni, "X University") labeled with *name* and vertex labels *Professor* (pr837), *Student* (st143) and *Organization* (xuni) state that professor Bob and student Alice both work at the organization X University.

In a graph $G = (V, E, l_V, l_E)$ the *in-neighbors* of a vertex $v \in V$ are the set of vertices $N^-(v) = \{u \mid (u, v) \in E\}$ from which v receives an edge. Similarly, v's *out-neighbors* are the set $N^+(v) = \{w \mid (v, w) \in E\}$ to which it sends edges. For a set $S \subseteq V$, let $N^+(S) = \bigcup_{v \in S} N^+(v)$ be the set of out-neighbors of S.

3.2 Bisimulation

A bisimulation is an equivalence relation on the vertices of a directed graph [16, 27]. Informally, a bisimulation groups vertices with equivalent structural neighborhoods, i. e., the neighborhoods cannot be distinguished based on the vertices' sets of labels and/or edges' labels. *Forward bisimulation (fw)* considers outgoing edges; *backward bisimulation (bw)* uses incoming edges. Vertices u and v are forward-bisimilar if, every out-neighbor u' of u has a corresponding out-neighbor v' of v, and vice versa; furthermore, the two neighbors u' and v' must be bisimilar [16, 27]. Backward-bisimulation is defined similarly but using in-neighbors.

k	Partition blocks
0	{pr837}, {cs902}, {st143}, {uoy, xuni}, literals
1	{pr837}, {cs902}, {st143}, {uoy}, {xuni}, {"Alice"}, {"Charlie"}, {"Bob"}, {"X University", "Univ. of Y"}
2	{pr837}, {cs902}, {st143}, {uoy}, {xuni}, {"Alice"}, {"Charlie"}, {"Bob"}, {"X University"}, {"Univ. of Y"}

Table 1: Vertex-labeled backward 2-bisimulation partition of the example graph according to Definition 2.

This definition corresponds to a *complete* bisimulation. For the neighbors u' and v' to be bisimilar, their in-/out-neighbors must be bisimilar as well. A k-bisimulation is a bisimulation on G that considers features a distance at most k from a vertex when deciding whether it is equivalent to another.

The algorithms of Schätzle et al. [25] and Kaushik et al. [16] compute versions of forward and backward k-bisimulation on labeled graphs $G = (V, E, \Sigma_V, \Sigma_E)$.

Definition 1. *The* edge-labeled forward k-bisimulation $\approx^k_{\text{fw}} \subseteq V \times V$ *with* $k \in \mathbb{N}$ *of Schätzle et al. [25] is defined as follows:*

- $u \approx^0_{\text{fw}} v$ *for all* $u, v \in V$,
- $u \approx^{k+1}_{\text{fw}} v$ *iff* $u \approx^k_{\text{fw}} v$ *and, for every edge* (u, u'), *there is an edge* (v, v') *with* $l_E(u, u') = l_E(v, v')$ *and* $u' \approx^k_{\text{fw}} v'$, *and vice-versa.*

Reviewer #2 noticed the "vice-versa" was missing.

For the graph in Figure 1, $\approx^0_{\text{fw}}$ has the single block V (as for all graphs, all vertices are initially equivalent) and $\approx^1_{\text{fw}}$ has three blocks, i. e., sets of equivalent vertices: $\{\text{pr837}, \text{cs902}, \text{st143}\}$, $\{\text{uoy}, \text{xuni}\}$, and the literals. The vertices xuni and uoy are not 1-bisimilar to st143, cs902 and pr837, as they have no outgoing edge labeled *worksAt*. For $k \geq 2$, k-bisimulation in this case makes no more distinctions than 1-bisimulation. Note that Schätzle et al. compute full bisimulations. We have modified their algorithm to stop after k iterations.

Definition 2. *The* vertex-labeled backward k-bisimulation $\approx^k_{\text{bw}} \subseteq V \times V$ *with* $k \in \mathbb{N}$ *of Kaushik et al. [16] is defined as follows:*

- $u \approx^0_{\text{bw}} v$ *iff* $l_V(u) = l_V(v)$,
- $u \approx^{k+1}_{\text{bw}} v$ *iff* $v \approx^k_{\text{bw}} u$ *and, for every* $(u', u) \in E$, *there is* $(v', v) \in E$ *with* $u' \approx^k_{\text{bw}} v'$, *and vice versa.*

Table 1 shows the vertex-labeled backward 2-bisimulation partitions of the graph in Figure 1. 0-bisimulation partitions by label. Then, xuni, uoy and the literals are split by their parents' labels. No vertex is 2-bisimilar to any other: every block is a singleton.

3.3 Graph Summaries for Bisimulation

The BRS algorithm summarizes graphs with respect to a *graph summary model* (GSM), a mapping from graphs $G = (V, E)$ to equivalence relations $\sim \subseteq V \times V$.

The equivalence classes of $\sim$ partition G. A simple GSM is label equality, i. e., two vertices are equivalent iff they have the same label. Depending on the application, one might want to summarize a graph w.r.t. different GSMs. Therefore, the algorithm works with GSMs defined in our formal language FLUID [6]. To flexibly and quickly define GSMs, the language provides simple and complex schema elements, along with six parameterizations, of which we use two (for details we refer to [6]).

A *complex schema element* $CSE := (\sim_s, \sim_p, \sim_o)$ combines three equivalence relations [6]. Vertices v and v' are equivalent, iff $v \sim_s v'$; and for all $w \in N^+(v)$ there is a $w' \in N^+(v')$ with $l_E(v, w) \sim_p l_E(v', w')$ and $w \sim_o w'$, and vice versa.

The chaining parameterization enables computing k-bisimulations by increasing the neighborhood considered for determining vertex equivalence [6]. It is defined by nesting CSEs. Given a complex schema element $CSE := (\sim_s, \sim_p, \sim_o)$ and $k \in \mathbb{N}_{>0}$, the chaining parameterization CSE^k defines the equivalence relation that corresponds to recursively applying CSE to a distance of k hops. $CSE^1 := (\sim_s, \sim_p, \sim_o)$ and, inductively for $k > 1$, $CSE^k := (\sim_s, \sim_p, CSE^{k-1})$. This results in a summary graph that has one vertex for each equivalence class in each equivalence relation defined within the CSE. Let C_i and C_{i+1} be equivalence classes in the i- and $(i+1)$-bisimulation relations. There is an edge with label p between C_i and C_{i+1} if there is an edge with that label between some $u \in C_i$ and $v \in C_{i+1}$. For full details, see [6].

Response to Reviewer #1 explaining the quotient graph (=summary).

To model backward k-bisimulations, we need to work with incoming edges, whereas the schema elements consider only outgoing edges. In FLUID, this is done with the direction parameterization [6] but, here, we simplify notation. We write SE^{-1} for the schema element defined analogously to SE but using the relation $E^{-1} = \{(y, x) \mid (x, y) \in E\}$ in place of the graph's edge relation E, and the edge labeling $\ell_E^{-1}(y, x) = \ell_E(x, y)$.

Following Definitions 1 and 2 of Schätzle et al. and Kaushik et al., we define CSE_{Sch} and CSE_{Kau} as follows. Here, $id = \{(v, v) \mid v \in V\}$ and $T = V \times V$, and vertices are equivalent in OC_{type} iff they have the same labels [6].

$$CSE_{\text{Sch}} := (T, id, T)^k \tag{1}$$

$$CSE_{\text{Kau}} := \left((OC_{\text{type}}, T, OC_{\text{type}})^{-1}\right)^k . \tag{2}$$

4 Algorithms

We introduce the single-purpose algorithms of Schätzle et al. [25] in Section 4.1 and Kaushik et al. [16] in Section 4.2. Finally, we introduce the generic BRS algorithm in Section 4.3. The first two algorithms compute summaries in a fundamentally different way to the BRS algorithm. At the beginning of the execution, BRS considers every vertex to be in its own equivalence class. During execution, vertices with the same vertex summary are merged. Therefore, BRS can be seen as a *bottom-up* approach. In contrast, Schätzle et al. consider all vertices to be equivalent at the beginning, and Kaushik et al. initially consider

all vertices with the same label to be equivalent. The equivalence relation is then successively refined. These two algorithms can be seen as *top-down* approaches.

Here, it is convenient to consider set partitions. A partition of a set V is a set $\{B_1, \ldots, B_\ell\}$ such that: (i) $\emptyset \subsetneq B_i \subseteq V$ for each i, (ii) $\bigcup_i B_i = V$, and (iii) $B_i \cap B_j = \emptyset$ for each $i \neq j$. The sets B_i are known as *blocks*. The equivalence classes of an equivalence relation over V partition the graph's vertices. A key concept is *partition refinement*. A partition $P_i = \{B_{i1}, B_{i2}, \ldots\}$ *refines* $P_j = \{B_{j1}, B_{j2}, \ldots\}$ iff every block B_{ik} of P_i is contained in a block $B_{j\ell}$ of P_j.

4.1 Native Schätzle et al. Algorithm

This algorithm [25] is a distributed MapReduce approach for reducing labeled transition systems. Two fundamental concepts are the *signature* and *ID* of a vertex v with respect to the current iteration's partition P_i.

The *signature* of a vertex v w.r.t. a partition $P_i = \{B_{i1}, B_{i2}, \ldots\}$ of V is given by $\text{sig}_{P_i}(v) = \{(\ell, B_{ij}) \mid (v, w) \in E \text{ with } l_E(v, w) = \ell \text{ and } w \in B_{ij}\}$. That is, v's signature w.r.t. the current iteration's partition P_i is the set of outgoing edge labels to blocks of P_i. By Definition 1, $u \approx_{\text{fw}}^{k+1} v$, iff $\text{sig}_{P_k}(u) = \text{sig}_{P_k}(v)$. Therefore signatures identify the block of a vertex and represent the current bisimulation partition, and the signature of v w.r.t. P_i can be represented as $\text{sig}_{P_{i+1}}(v) = \{(l_E(v, w), \text{sig}_{P_i}(w)) \mid (v, w) \in E\}$. The nested structure of vertex signatures means they can become very large. Thus, we compute a recursively defined hash value proposed by Hellings et al. [13], which is also used by Schätzle et al. [25]. We use this hash function to assign $\text{sig}_{P_i}(v)$ an integer value which we denote by $\text{ID}_{P_i}(v)$. Now the signature of a vertex v w.r.t. the current partition P_i can be represented as $\text{sig}_{P_{i+1}}(v) = \{(l_E(v, w), \text{ID}_{P_i}(w)) \mid (v, w) \in E\}$.

Response to Reviewer #1 comment on citing Hellings et al., SIGMOD 2012, to introduce the idea of working with hash values. As well how the ID function is defined.

With $\text{sig}_{P_i}(v)$ and $\text{ID}_{P_i}(v)$, the procedure for computing an edge-labeled forward k-bisimulation partition is outlined in Algorithm 1. The initial partition is just V (line 2), as every vertex is 0-bisimilar to every other vertex. Next, the algorithm performs k iterations (lines 3–8). In the ith iteration, the information needed to construct a vertex's signature $\text{sig}_{P_i}(v)$ is sent to every vertex v (line 5). This information is the edge label $l_E(v, w) \in \Sigma_E$ and the block identifier $\text{ID}_{P_{i-1}}(w)$ for every $w \in N^+(v)$. The signature $\text{sig}_{P_i}(v)$ is then constructed using the received information, and the identifiers $\text{ID}_{P_i}(v)$ are updated for all v (line 6). At the end of each iteration, the algorithm checks if any vertex ID was updated, by comparing the number of distinct values in ID_{P_i} and $\text{ID}_{P_{i-1}}$ (line 7). If no vertex ID was updated, we have reached full bisimulation [3, 25] and hence can stop execution early. At the end, the resulting k-bisimulation partition P_k is constructed by putting vertices v in one block if they share the same identifier value $\text{ID}_{P_i}(v)$ (line 9).

4.2 Native Kaushik et al. Algorithm

This algorithm [16] sequentially computes vertex-labeled backward k-bisimulations. The following definitions are from [23], modified for backward bisimulation.

Algorithm 1: Bisimulation Algorithm by Schätzle et al. [25]

```
1  function BISIMSCHÄTZLE(G = (V, E, l_V, l_E), k ∈ ℕ)
       /* Initially, all v ∈ V in same block with ID_{P_0}(v) = 0 */
2      P_0 ← {V};
3      for i ← 1 to k do
           /* Map Job */
4          for (v, w) ∈ E do
5              Send (l_E(v, w), ID_{P_{i−1}}(w))
           /* Reduce Job */
6          Construct sig_{P_i}(v) and update ID_{P_i}(v);
           /* Check if full bisimulation is reached */
7          if |ID_{P_i}| = |ID_{P_{i−1}}| then
8              break;
9      Construct P_k from ID_{P_i};
10     return P_k;
```

A subset $B \subseteq V$ is *stable* with respect to another subset $S \subseteq V$ if either $B \subseteq N^+(S)$ or $B \cap N^+(S) = \emptyset$. That is, vertices in a stable set B are indistinguishable by their relation to S: either all vertices in B get at least one edge from S, or none do. If B is not stable w.r.t. S, we call S a *splitter* of B.

Building on this, partition P_i of V is *stable with respect to a subset* $S \subseteq V$ if every block $B_{ij} \in P_i$ is stable w.r.t. S. P_i is *stable* if it is stable w.r.t. each of its blocks B_{ij}. Thus, a partition P_i is stable if none of its blocks B_{ij} can be split into a set of vertices that receive edges from some B_{ik} and a set of vertices that do not. A stable partition corresponds to the endpoint of a bisimulation computation: no further distinctions can be made.

This gives an algorithm for bisimulation, due to Paige and Tarjan [23] who refer to it as the "naïve algorithm". The initial partition is repeatedly refined by using its own blocks or unions of them as splitters: if S splits a block B, we replace B in the partition with the two new blocks $B \cap N^+(S)$ and $B - N^+(S)$. When no more splitters exist, the partition is stable [23] and equivalent to the full backward bisimulation of the initial partition P_0 [15].

The algorithm of Kaushik et al., Algorithm 2, modifies this naïve approach. The first difference is that in each iteration $i \in \{1, \ldots, k\}$ the partition is stabilized with respect to each of its own blocks (lines 7–16). This ensures that, after iteration i, the algorithm has computed the i-bisimulation [16], which is not the case in the naïve algorithm. Second, blocks are split as defined above, (lines 9–15). As a result, Algorithm 2 computes the k-backward bisimulation. To check if full bisimulation has been reached, the algorithm uses the Boolean variable wasSplit (lines 6 and 15). If this is false at the end of an iteration i, no block was split, so the algorithm stops early (line 17). Moreover, Algorithm 2 tracks which sets have been used as splitters (line 3), to avoid checking for stability against sets w.r.t. which the partition is already known to be stable. The

algorithm provided by Kaushik et al. does not include this. If a partition P is stable w.r.t. a block B, each refinement of P is also stable w.r.t. B [23]. So after the partition P is stabilized w.r.t. a block copy B^{copy} (lines 7–16), we can add B^{copy} to the usedSplitters set and not consider it in subsequent iterations.

Algorithm 2: Bisimulation Algorithm by Kaushik et al. [16]

```
1  function BISIMKAUSHIK(G = (V, E, l_V, l_E), k ∈ ℕ)
       /* P := {B_1, B_2, ..., B_t} */
2      P ← partition V by label;
3      usedSplitters ← ∅;
4      for i ← 1, ..., k do
5          P^copy ← P;
6          wasSplit ← false;
7          for B^copy ∈ P^copy − usedSplitters do
               /* Use blocks of copy partition to stabilize blocks of
                  original partition */
8              for B ∈ P do
9                  succ ← B ∩ N⁺(B^copy);
10                 nonSucc ← B − N⁺(B^copy);
                   /* Split non-stable blocks */
11                 if succ ≠ ∅ and nonSucc ≠ ∅ then
12                     P.add(succ);
13                     P.add(nonSucc);
14                     P.delete(B);
15                     wasSplit ← true;
16             usedSplitters.add(B^copy);
17         if ¬wasSplit then
18             break;
19     return P;
```

4.3 Generic BRS Algorithm

The parallel BRS algorithm is not specifically an implementation of k-bisimulation. Rather, one can define a graph summary model in a formal language FLUID (see Section 3.3). This model is denoted by $\sim$ and input to the BRS algorithm, which then summarizes a graph w.r.t. $\sim$. In particular, the k-bisimulation models of Schätzle et al. and Kaushik et al. can be expressed in FLUID, as shown in Section 3.3. Thus, the BRS algorithm can compute k-bisimulation partitions. In other words, k-bisimulation can be incorporated into any graph summary model defined in FLUID. The BRS algorithm summarizes the graph w.r.t. $\sim$ in parallel and uses the Signal/Collect paradigm. In Signal/Collect [26], vertices collect information from their neighbors, sent over the edges as signals. Details of the use of the Signal/Collect paradigm in our algorithm can be found in Blume et al. [5].

Response to Reviewer #1 with some more details about the message passing. See also comment further below.

Briefly, the algorithm builds equivalence classes by starting with every vertex in its own singleton set and forming unions of equivalent vertices. Before outlining the algorithm, we give a necessary definition.

Definition 3. *Suppose we have a graph summary GS of $G = (V, E, l_V, l_E)$ w.r.t. some GSM $\sim$. For each $v \in V$, the* vertex summary *vs is the subgraph of GS that defines v's equivalence class w.r.t. $\sim$.*

We give the pseudocode of our version of the BRS algorithm in Algorithm 3 and briefly describe it below. [4] gives a step-by-step example run.

Initialization (lines 3–5). For each vertex $v \in V$, VertexSchema computes the local schema information w.r.t. $\sim_s$ and $\sim_o$ of the graph summary model $(\sim_s,\sim_p,\sim_o)^k$. The method also takes into account $\sim_p$, i. e., the equivalence relation defined over the edge labels. Order-invariant hashes of this schema information are stored as the identifiers $id_{\sim_s}$ and $id_{\sim_o}$ (lines 4–5).

At the end of the initialization, every vertex has identifiers $id_{\sim_s}$ and $id_{\sim_o}$. Two vertices v and v' are equivalent w.r.t. $(\sim_s, \sim_p, \sim_o)$, iff $v.id_{\sim_s} = v'.id_{\sim_s}$. Thus, this initialization step can be seen as iteration $k = 0$ of bisimulation.

Case of $k = 1$ bisimulation (lines 6–11). Every vertex v sends, to each in-neighbor w, its $id_{\sim_o}$ value and the label set $\ell(w, v)$ of the edge (w, v) (line 9).

Reviewer #1: "What is L?"

Every vertex v sends, to each in-neighbor w, its $id_{\sim_o}$ value and the label set $L = \ell(w, v)$ of the edge (w, v) (line 9). Thus, each vertex receives a set of schema $\langle L, id_{\sim_o}\rangle$ pairs from its out-neighbors, which are collated into the set M_o (line 10) and merged with an order-invariant hash to give v's new $id_{\sim_s}$ (line 11). Here, the MergeAndHash(M_o) function first merges the elements of the schema message M_o received from vertex o and hashes it with an order-independent hash function. This hash is then combined with the existing hash value $v.id_{\sim_s}$ using the xor ($\oplus$) operator.

Response to Reviewer #1 explaining MergeAndHash and $\oplus$.

Case of $k > 1$ bisimulation (lines 13–32). In the first iteration (lines 13–19), every vertex v sends a message to each of its out-neighbors w. The message contains v's $id_{\sim_s}$ and $id_{\sim_o}$ values, and the edge label set $\ell(w, v)$ (line 15). Subsequently, the incoming messages of the vertex are merged into a set of tuples with the received information $\langle \ell(w, v), id_{\sim_s}\rangle$ and $\langle \ell(w, v), id_{\sim_o}\rangle$ (lines 16 and 17). Finally, the identifiers $id_{\sim_s}$ and $id_{\sim_o}$ of v are updated by hashing the corresponding set (lines 18 and 19). Note that, whenever an update of an identifier value $v.id_{\sim_s}$ of vertex v is performed, the algorithm combines the old $v.id_{\sim_s}$ with the new hash value, indicated by $\oplus$.

In the remaining iterations, the algorithm performs the same steps (lines 20–27), but excludes the edge label set $\ell(w, v)$ when merging messages for $id_{\sim_o}$ (line 25). When merging the messages in $id_{\sim_o}$, it is not necessary to consider $\ell(w, v)$, as in the iterations 2 to $k - 1$ it is only needed to update the $id_{\sim_o}$ values using the hash function as described above. This is possible as $id_{\sim_o}$ by definition already contains the edge label information $\ell(w, v)$, computed in the first iteration.

In the final iteration, the identifiers are updated w.r.t. the final messages (lines 28–32). The final messages received are the values stored in the out-neighbors' $id_{\sim_o}$ values. Each vertex signals its $id_{\sim_o}$ value to its in-neighbors

Algorithm 3: Parallel BRS algorithm

```
1  function ParallelSummarize(G, (∼_s, ∼_p, ∼_o)^k)
2      returns graph summary SG
       /* Initialization */
3      for all v ∈ V do in parallel
4          v.id_{∼_s} ← hash(VertexSchema(v, G, ∼_s, ∼_p));
5          v.id_{∼_o} ← hash(VertexSchema(v, G, ∼_o, ∼_p));
       /* If k = 1, only signal edge labels and v.id_{∼_o} */
6      if k = 1 then
7          for all v ∈ V do in parallel
8              for all w ∈ N⁻(v) do
9                  SendMsgs(w, ⟨ℓ(w, v), 0, v.id_{∼_o}⟩);
10             M_o ← {⟨L, id_{∼_o}⟩ | ⟨L, id_{∼_s}, id_{∼_o}⟩ was received};
11             v.id_{∼_s} ← v.id_{∼_s} ⊕ MergeAndHash(M_o);
12     else
           /* Signal initial messages. Update v.id_{∼_s} and v.id_{∼_o} */
13         for all v ∈ V do in parallel
               /* Message each in-neighbor */
14             for all w ∈ N⁻(v) do
15                 SendMsg(w, ⟨ℓ(w, v), v.id_{∼_s}, v.id_{∼_o}⟩);
               /* Collect all incoming messages of v */
16             M_s ← {⟨L, id_{∼_s}⟩ | ⟨L, id_{∼_s}, id_{∼_o}⟩ was received};
17             M_o ← {⟨L, id_{∼_o}⟩ | ⟨L, id_{∼_s}, id_{∼_o}⟩ was received};
               /* Update identifiers by hashing the messages */
18             v.id_{∼_s} ← v.id_{∼_s} ⊕ MergeAndHash(M_s);
19             v.id_{∼_o} ← v.id_{∼_o} ⊕ MergeAndHash(M_o);
           /* Signal messages k − 2 times. As above, but we do not
              include L when updating v.id_{∼_o}. (See text.) */
20         for i ← 2 to k − 1 do
21             for all v ∈ V do in parallel
22                 for all w ∈ N⁻(v) do
23                     SendMsg(w, ⟨ℓ(w, v), v.id_{∼_s}, v.id_{∼_o}⟩);
24                 M_s ← {⟨L, id_{∼_s}⟩ | ⟨L, id_{∼_s}, id_{∼_o}⟩ received};
25                 M_o ← {⟨id_{∼_o}⟩ | ⟨L, id_{∼_s}, id_{∼_o}⟩ received};
26                 v.id_{∼_s} ← v.id_{∼_s} ⊕ MergeAndHash(M_s);
27                 v.id_{∼_o} ← v.id_{∼_o} ⊕ MergeAndHash(M_o);
           /* Signal final messages. Update v.id_{∼_s} */
28         for all v ∈ V do in parallel
29             for all w ∈ N⁻(v) do
30                 SendMsg(w, ⟨∅, 0, v.id_{∼_o}⟩);
31             M_o ← {⟨id_{∼_o}⟩ | ⟨L, id_{∼_s}, id_{∼_o}⟩ was received};
32             v.id_{∼_s} ← v.id_{∼_s} ⊕ MergeAndHash(M_o);
33     SG ← FindAndMerge(SG, V);
34     return SG;
```

Graph	$\|V\|$	$\|E\|$	$\|\Sigma_V\|$	$r(l_V)$	$\mu(\|l_V(v)\|)$	$\|\Sigma_E\|$
Laundromat100M	30 M	88 M	33,431	7,373	0.93 ± 44	5,630
BTC150M	5 M	145 M	69	137	1.04 ± 0.26	10,750
BTC2B	80 M	1.92 B	113,365	576,265	0.95 ± 1.82	38,136
BSBM100M	18 M	90 M	1,289	2,274	1.02 ± 0.13	39
BSBM1B	172 M	941 M	6,153	27,306	1.03 ± 0.18	39

Table 2: Statistics of the datasets.

(line 30). The messages a vertex receives are merged (line 31) and hashed to update the final $id_{\sim_s}$ value (line 32). Equivalence between any two vertices v and v' can now be defined. Vertices with the same $id_{\sim_s}$ value are merged (line 33), ending the computation.

Response to Reviewer #2 on complexity of the algorithm.

We show in [4] that Algorithm 3 computes k-bisimulation of a graph with m edges in time $O(km)$. As a modification of the algorithm in [6], the algorithm is correct, as long as hash collisions are avoided.

5 Experimental Apparatus

5.1 Datasets

We experiment with smaller and larger as well as real-world and synthetic graphs. Table 2 lists statistics of these datasets, where $r(l_V) = |\{l_V(v) \mid v \in V\}|$ is the number of different label sets (range) and $\mu(|l_V(v)|)$ is the average number of labels of a vertex $v \in V$.

Three real-world datasets were chosen. The *Laundromat100M* dataset contains 100 M edges of the LOD Laundromat service [1]. This service automatically cleaned existing linked datasets and provided the cleaned version on a publicly accessible website. The *BTC150M* and *BTC2B* datasets contain, respectively, around 150 million and 1.9 billion edges of the Billion Triple Challenge 2019 (BTC2019) dataset [14]. 93% of the total edges originate from Wikidata [29]. BTC150M is the first chunk of the 1.9 billion edges. For synthetic datasets, two versions of the Berlin SPARQL Benchmark (BSBM) [2] were used. The BSBM data generator produces RDF datasets that simulate an e-commerce use case. *BSBM100M* was generated with 284,826 products and has about 17.77 million vertices and 89.54 million edges. *BSBM1B* was generated with 2,850,000 products and has about 172 million vertices and 941 million edges.

5.2 Procedure

An experiment consists of the algorithm to run, the dataset to summarize, and the bisimulation degree k. In case of the BRS algorithm, it additionally consists of the graph summary model to use. We have two different graph summary models defined by Schätzle et al. [25] and Kaushik et al. [16]. Each model comes in two implementations, one native implementation as defined by the original authors

and a generic implementation through our hash-based version of the BRS algorithm. We chose the bisimulation algorithms of Kaushik et al. and Schätzle et al. as they represent two typical variants of backward and forward k-bisimulation models as found in the literature (cf. Section 2). This choice of algorithms allows us to demonstrate that our algorithm can be applied to different settings. We use the terms *BRS-Schätzle* and *BRS-Kaushik* to refer to our implementations of the two GSMs BRS algorithm; we refer to our single-purpose implementations of these two GSMs as *native Schätzle* and *native Kaushik*.

Response to Review #1 on the motivation to choose Kaushik et al. and Schätzle et al.

The four algorithms are applied on the five datasets, giving 20 experiments. Each experiment is executed with a bisimulation degree of $k = 1, \ldots, 10$, using the following procedure. We run the algorithms six times with the specific configuration. We use the first run as a warm-up and do not account it for our measurements. The next five runs are used to measure the variables.

5.3 Implementation

All algorithms, i. e., the native algorithms of Schätzle et al. and Kaushik et al. and their generic BRS-variants are implemented using the same underlying framework and paralellization approach, i. e., are implemented in Scala upon the Apache Spark Framework. This API offers flexible support for parallel computation and message passing, which enables implementation of Map-Reduce and Signal-Collect routines. We use an Ubuntu 20 system with 32 cores and 2 TB RAM. The Apache Spark contexts were given the full resources. Time and memory measurements were taken using the Apache Spark Monitoring API.

Response to Reviewer #1 with some more details about the message passing. See also comment above.

5.4 Measures

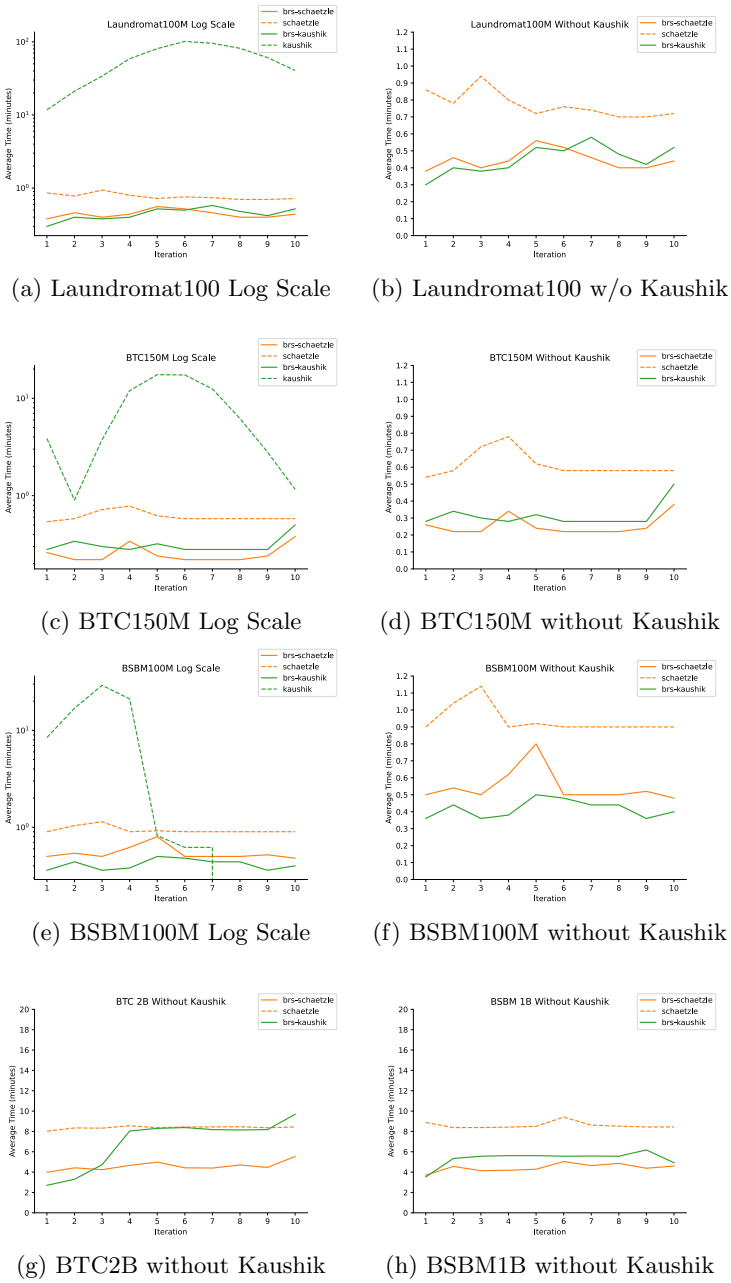
We evaluate the algorithms' running time and memory consumption. For every run of an experiment, we report the total run time, the run time of each of the k iterations, and the maximum JVM on-heap memory consumption.

6 Results

We present full results for each algorithm, iteratively calculating k-bisimulation for every value of $k = 1, \ldots, 10$ and every dataset, in Figure 2. Table 3 summarizes the average total run time (minutes) for each experiment, for the computation of 10-bisimulation. The BRS algorithm takes an additional initialization step (see Algorithm 3, lines 13–19). Table 4 reports the maximum JVM on-heap memory in GB for each experiment. The BRS-Schätzle algorithm computes the 10-bisimulation the fastest on all datasets, except for BSBM100M, where BRS-Kaushik is fastest. Native Schätzle consumes the least memory on all smaller datasets. Native Schätzle consumes slightly more memory on BSBM1B than BRS, whereas on BTC2B, the memory consumption is about the same.

Made clearer that Figure 2 contains the main results as it reports the runtimes per iteration (each value of k).

Smaller Datasets (100M+ Edges). Figure 2 shows the average run time (minutes) for each of the ten iterations on the smaller datasets. The native



(a) Laundromat100 Log Scale (b) Laundromat100 w/o Kaushik

(c) BTC150M Log Scale (d) BTC150M without Kaushik

(e) BSBM100M Log Scale (f) BSBM100M without Kaushik

(g) BTC2B without Kaushik (h) BSBM1B without Kaushik

Fig. 2: Average iteration times (minutes) on the smaller datasets in (a) to (f) and larger datasets in (g) and (h).

	Schätzle et al.		Kaushik et al.	
	BRS	Native	BRS	Native
Laundromat100M	5.56 (0.50)	7.72 (0.07)	5.60 (0.15)	586.66 (21.09)
BTC150M	4.08 (1.55)	6.14 (0.38)	4.54 (1.34)	78.02 (3.71)
BTC2B	61.96 (11.38)	83.74 (1.96)	85.92 (13.6)	out of time
BSBM100M	6.46 (0.08)	9.40 (0.06)	5.20 (0.50)	77.84 (2.41)
BSBM1B	54.44 (4.35)	85.98 (3.83)	64.00 (3.22)	out of memory

Table 3: Total run time (in minutes) needed for computing the $k = 1, \ldots, 10$ bisimulation (average and standard deviation over 5 runs).

	Schätzle et al.		Kaushik et al.	
	BRS	Native	BRS	Native
Laundromat100M	211.5	147.9	210.5	335.1
BTC150M	140.6	107.7	130.1	181.7
BTC2B	1,249.1	1,249.7	1,249.3	out of time
BSBM100M	248.6	113.1	172.0	327.0
BSBM1B	1,248.2	1,335.4	1,249.2	out of memory

Table 4: Maximal JVM on-heap memory (in GB) used for $k = 1, \ldots, 10$ bisimulation. We provide the maximal memory usage determined over five runs, instead of the average and standard deviation, in order to demonstrate what memory is needed to perform the bisimulations without running out of memory.

Kaushik experiments take much longer than the others (Figures 2a, 2c, and 2e), so we provide plots without native Kaushik (Figures 2b, 2d, and 2f), to allow easier comparison between the other algorithms.

BRS-Schätzle and native Schätzle have relatively constant iteration time on all smaller datasets. For example, on Laundromat100M (Figure 2b), native Schätzle computes an iteration in about 0.7 to 0.95 minutes. BRS is slightly faster on all smaller datasets, but uses more memory (Table 4). BRS-Kaushik shows similar results to BRS-Schätzle. Again iteration time is relatively constant on all datasets. As before, it is fastest on BTC150M.

Native Kaushik shows a different behavior across the datasets. It does not have constant iteration time on any dataset: the iteration time increases in early iterations until it reaches a maximum value, and then decreases. The only exception to this behavior occurs on BTC150M, where the iteration time decreases from iteration one to two before following the described pattern for the remaining iterations. For example, on Laundromat100M (Figure 2a) an iteration takes about 10 to 100 minutes. The maximum iteration time is reached in iteration six. Native Kaushik runs fastest on BSBM100M, taking about 77.84 minutes on average. BRS-Kaushik is much faster on all smaller datasets (Table 3) and uses slightly less memory (Table 4).

Larger Datasets (1B+ Edges). BRS-Schätzle computes the $k = 1, \ldots, 10$ bisimulation on BTC2B and BSBM1B with a relatively constant iteration time. The total run time (Table 3), is lowest on BSBM1B: 54.44 minutes on average.

Native Schätzle also has nearly constant running time across iterations. The average running time for all ten iterations is lowest on BTC2B: 83.74 minutes on average, compared to 85.98 minutes on BSBM1B. Comparing the run times of the two implementations, BRS is 37% on the BSBM1B dataset and 26% faster on BTC2B. Both algorithms require about the same memory during execution on BTC2B. On BSBM1B, native Schätzle uses slightly more memory than BRS. Native Kaushik did not complete one iteration on BTC2B in 24 hours, so execution was canceled. The algorithm ran out of memory on BSBM1B (Table 4).

Finally, BRS-Kaushik also has similar iteration times. On BSBM1B (Figure 2h), computing a bisimulation iteration ranges from about 4 to 6 minutes. On BTC2B (Figure 2g), the execution times for iterations one to three range from 2 to 4 minutes. Iterations four to nine take about 8 minutes each. For the final iteration ten, the running time slightly increases to about 9.5 minutes.

7 Discussion

Rewrote main results to make the contribution clearer (response to Reviewer #1 and #2).

Main Results. Our results show that, on all datasets, the generic BRS algorithm outperforms the native bisimulation algorithms of Schätzle et al. and Kaushik et al. for $k = 10$ (see Table 3). Since the BRS algorithm has an initialization phase, which is not present in the two native algorithms, we examine the data more closely to see at which value of k BRS begins to outperform the native implementations. We provide the exact numbers per iteration together with the standard deviation and averaged over five executions in Appendix D. Our generic BRS-Kaushik outperforms the native Kaushik et al. for all k: this is unsurprising, as BRS parallelizes the original sequential algorithm. The time taken to compute k-bisimulation is the total time for iterations up to, and including, k. The comparison of BRS-Schätzle with the native Schätzle et al. algorithm shows that our generic algorithm begins to outperform the native one at $k = 3$ for the Laundromat100M, BSBM100M, and BSBM1B; at $k = 4$ for BTC150M; and at $k = 5$ for BTC2B. Again, we emphasize that BRS is a generic algorithm supporting all graph summary models definable in FLUID (see introduction), whereas native algorithms compute only one version of bisimulation each.

Additional response to Reviewer #2 to reflect on the choice of k, reaching full bisimulation, etc.

Note that, for each dataset and each algorithm, we iteratively compute in total ten bisimulation-based graph summaries using every value of k from 1 to 10. As can be seen in Figure 2, the execution times per bisimulation iteration are fairly constant over all iterations for all datasets for BRS and Schätzle et al. The native Kaushik et al. algorithm sequentially computes the refinement of the partitions, so execution time directly relates to the number of splits per k-bisimulation iteration. This can be seen by the curve in the plots of the smaller datasets (see Figures 2a, 2c, and 2e). In particular, Kaushik et al. detect that full bisimulation has been reached and do not perform further computations: this happens after computing the 7th iteration (7-bisimulation, see Figure 2e) on the BSBM100M dataset but not on the other datasets on which the execution

of the Kaushik et al. algorithm completed. This kind of termination check could also be added to the BRS algorithm but this is nontrivial, as we discuss below.

This consistent per-k iteration runtime is explained by each iteration of the BRS algorithm processing every vertex, considering all its neighbors. Thus, the running time of the iteration depends primarily on the size of the graph, and not on how much the k-bisimulation that is being computed differs from the $(k-1)$-bisimulation that was computed at the previous iteration. The native Schätzle et al. algorithm behaves in the same way. There is some variation in per-iteration execution time but this may be because, as each vertex is processed, duplicate incoming messages must be removed. The time taken for this will depend on the distribution of the incoming messages, which will vary between iterations.

We consider experimenting with values between $k = 1$ and 10 to be appropriate. The native Kaushik et al. algorithm terminates when full bisimulation is reached, which shows that full bisimulation is reached at $k = 7$ on the BSBM100M dataset, but has not been reached up to $k = 10$ for Laundromat100 and BTC150M. The rapidly decreasing per-iteration running times for native Kaushik et al. for BTC150M (Figure 2c) suggests that full bisimulation will be reached in a few more iterations. A similar curve can be observed for Laundromat100 (Figure 2a), but the running time for the $k = 10$ iteration is higher, suggesting that full bisimulation will not be reached for several iterations. Thus, on these datasets, computing 10-bisimulations is a reasonable thing to do, as full bisimulation has not yet been reached. Note that, for the other algorithms that we consider (native Schätzle, and the two instances of BRS), per-iteration running time is largely independent of k and of whether full bisimulation as been reached (see the per-iteration measurements on BSBM100M in Table 9).

Conversely, our primary motivation is graph summarization. When two vertices in a graph are 10-bisimilar, this means they have equivalent neighborhoods out to distance 10. This means that they are already "largely similar" for that value of k. Thus, we feel that, having computed k-bisimulation for a relatively large value of k, there is little advantage in going to $k+1$. In other words, adding another iteration of $k+1$ still leaves us with vertices that are "largely similar".

Scalability to Graph Size. From the execution time of computing k-bisimulation for $k = 1, \ldots, 10$, we observe that BRS-Schätzle, BRS-Kaushik, and native Schätzle scale linearly. Here, we consider the scalability of the algorithms with respect to graph size. To this end, we fix on iterations $k = 1, \ldots, 10$ and compare the total time taken to compute these bisimulations for input graphs of different (large) sizes. BTC150M contains approximately 5M vertices and 145M edges. BTC2B has around 80M vertices and 2B edges, which is a factor of about 18 and 14 larger, respectively. BRS-Schätzle takes 4.08 minutes, BRS-Kaushik 4.54 minutes, and native Schätzle 6.14 minutes on BTC150M. On BTC2B, they take 15 times, 19 times, and 14 times longer. BSBM1B contains about 10 times as many vertices and edges as BSBM100M. The experiments on BSBM1B took about 10 times longer for BRS-Schätzle, 12 times for BRS-Kaushik, and 9 times for native Schätzle. Thus, the scaling factor of the execution times is approximately equal to that of the graph's size. Finally, the total runtimes of native

Kaushik on the different graphs indicate that the algorithm does not scale linearly with the input graph's size. The initial partition P_0 has one block per label set and Laundromat100M has many different label sets (Definition 2). The algorithm checks stability of every block against every other, leading to a run time that is quadratic in the number of blocks.

Generalization and Threats to Validity. We use synthetic and real-world graphs, which is important to analyze the practical application of an algorithm [5, 17]. Two GSMs were used for evaluation of our hash-based BRS algorithm. The GSM of Schätzle et al. computes a forward k-bisimulation, based on edge labels (Definition 1). The GSM of Kaushik et al. computes a backward k-bisimulation based on vertex labels (Definition 2). Hence, the two GSMs consider different structural features for determining vertex equivalence. Regardless, for both GSMs, the BRS algorithm scales linearly with the number of bisimulation iterations and the input graph's size and computes the aggregated $k = 1, \ldots, 10$-bisimulations the fastest on every dataset.

All algorithms are implemented in Scala, in the same framework. Each algorithm was executed using the same procedure on the same machine with exclusive access during the experiments. Each experimental configuration was run six times. The first run is discarded in the evaluations to address side effects.

8 Conclusion and Future Work

We focus on the performance (runtime and memory use) of our generic, parallel BRS algorithm for computing different bisimulation variants, and how this performance compares to specific algorithms for those bisimulations, due to Schätzle et al. and Kaushik et al. Our experiments comparing $k = 1, \ldots, 10$ bisimulations on large synthetic and real-world graphs show that our generic, hash-based BRS algorithm outperforms the respective native bisimulation algorithms on all datasets for all $k \geq 5$ and for smaller k in some cases. The experimental results indicate that the parallel BRS algorithm and native Schätzle et al. scale linearly with the number of bisimulation iterations and the input graph's size. Our experiments also show that our generic BRS-Kaushik algorithm effectively parallelizes the original sequential algorithm of Kaushik et al., showing runtime performance similar to BRS-Schätzle. Overall, we recommend using our generic BRS algorithm over implementations of specific algorithms of k-bisimulations. Due to the support of a formal language for defining graph summaries [6], the BRS approach is flexible and easily adaptable, e. g., to changes in the desired features and user requirements, without sacrificing performance.

Future work includes incorporating a check (similar to Kaushik et al.) for whether full bisimulation has been reached. This is nontrivial as the algorithm of Kaushik et al. is based on splitting partitions, whereas BRS is based on describing the equivalence class in which each vertex lives. It is easy to check that no classes have been split, but harder to check that every pair of vertices that had the same description at the previous iteration have the same description at the current iteration, since the description of every vertex changes at each iteration.

References

1. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: LOD laundromat: A uniform way of publishing other people's dirty data. In: ISWC. vol. 8796, pp. 213–228. Springer (2014). https://doi.org/10.1007/978-3-319-11964-9_14
2. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semantic Web Inf. Syst. **5**(2), 1–24 (2009). https://doi.org/10.4018/jswis.2009040101
3. Blom, S., Orzan, S.: A distributed algorithm for strong bisimulation reduction of state spaces. Electron. Notes Theor. Comput. Sci. **68**(4), 523–538 (2002). https://doi.org/10.1016/S1571-0661(05)80390-1
4. Blume, T., Rau, J., Richerby, D., Scherp, A.: Time and memory efficient parallel algorithm for structural graph summaries and two extensions to incremental summarization and k-bisimulation for long k-chaining. CoRR **abs/2111.12493** (2021), https://arxiv.org/abs/2111.12493
5. Blume, T., Richerby, D., Scherp, A.: Incremental and parallel computation of structural graph summaries for evolving graphs. In: CIKM. pp. 75–84. ACM (2020). https://doi.org/10.1145/3340531.3411878
6. Blume, T., Richerby, D., Scherp, A.: FLUID: A common model for semantic structural graph summaries based on equivalence relations. Theor. Comput. Sci. **854**, 136–158 (2021). https://doi.org/10.1016/j.tcs.2020.12.019
7. Bonifati, A., Dumbrava, S., Kondylakis, H.: Graph summarization. CoRR **abs/2004.14794** (2020), https://arxiv.org/abs/2004.14794
8. Buneman, P., Staworko, S.: RDF graph alignment with bisimulation. Proc. VLDB Endow. **9**(12), 1149–1160 (2016). https://doi.org/10.14778/2994509.2994531
9. Čebirić, Š., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., Zneika, M.: Summarizing semantic graphs: a survey. VLDB J. **28**(3), 295–327 (2019). https://doi.org/10.1007/s00778-018-0528-3
10. Ciglan, M., Nørvåg, K., Hluchý, L.: The semsets model for ad-hoc semantic list search. In: WWW. pp. 131–140. ACM (2012). https://doi.org/10.1145/2187836.2187855
11. Consens, M.P., Fionda, V., Khatchadourian, S., Pirrò, G.: S+EPPs: Construct and explore bisimulation summaries, plus optimize navigational queries; all on existing SPARQL systems. Proc. VLDB Endow. **8**(12), 2028–2031 (2015). https://doi.org/10.14778/2824032.2824128
12. Goasdoué, F., Guzewicz, P., Manolescu, I.: RDF graph summarization for first-sight structure discovery. VLDB J. **29**(5), 1191–1218 (2020). https://doi.org/10.1007/s00778-020-00611-y
13. Hellings, J., Fletcher, G.H.L., Haverkort, H.J.: Efficient external-memory bisimulation on dags. In: SIGMOD. pp. 553–564. ACM (2012), https://doi.org/10.1145/2213836.2213899
14. Herrera, J., Hogan, A., Käfer, T.: BTC-2019: the 2019 billion triple challenge dataset. In: ISWC. pp. 163–180. Springer (2019). https://doi.org/10.1007/978-3-030-30796-7_11
15. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. Inf. Comput. **86**(1), 43–68 (1990). https://doi.org/10.1016/0890-5401(90)90025-D
16. Kaushik, R., Shenoy, P., Bohannon, P., Gudes, E.: Exploiting local similarity for indexing paths in graph-structured data. In: ICDE. pp. 129–140. IEEE (2002). https://doi.org/10.1109/ICDE.2002.994703

17. Luo, Y., Fletcher, G.H.L., Hidders, J., Bra, P.D., Wu, Y.: Regularities and dynamics in bisimulation reductions of big graphs. In: Workshop on Graph Data Management Experiences and Systems. p. 13. CWI/ACM (2013). https://doi.org/10.1145/2484425.2484438
18. Luo, Y., Fletcher, G.H.L., Hidders, J., Wu, Y., Bra, P.D.: External memory k-bisimulation reduction of big graphs. In: CIKM. pp. 919–928. ACM (2013). https://doi.org/10.1145/2505515.2505752
19. Martens, J., Groote, J.F., van den Haak, L.B., Hijma, P., Wijs, A.: A linear parallel algorithm to compute bisimulation and relational coarsest partitions. In: Formal Aspects of Component Software (FACS). LNCS, vol. 13077, pp. 115–133. Springer (2021), https://doi.org/10.1007/978-3-030-90636-8_7
20. Milo, T., Suciu, D.: Index structures for path expressions. In: ICDT. pp. 277–295. Springer (1999). https://doi.org/10.1007/3-540-49257-7_18
21. Nestorov, S., Ullman, J.D., Wiener, J.L., Chawathe, S.S.: Representative objects: Concise representations of semistructured, hierarchical data. In: ICDE. pp. 79–90. IEEE Computer Society (1997), https://doi.org/10.1109/ICDE.1997.581741
22. Neumann, T., Moerkotte, G.: Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In: ICDE. pp. 984–994. IEEE (2011). https://doi.org/10.1109/ICDE.2011.5767868
23. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987). https://doi.org/10.1137/0216062
24. Rau, J., Richerby, D., Scherp, A.: Single-purpose algorithms vs. a generic graph summarizer for computing k-bisimulations on large graphs. CoRR **abs/2204.05821** (2022), https://doi.org/10.48550/arXiv.2204.05821
25. Schätzle, A., Neu, A., Lausen, G., Przyjaciel-Zablocki, M.: Large-scale bisimulation of RDF graphs. In: Workshop on Semantic Web Information Management. pp. 1:1–1:8. ACM (2013). https://doi.org/10.1145/2484712.2484713
26. Stutz, P., Strebel, D., Bernstein, A.: Signal/Collect12. Semantic Web **7**(2), 139–166 (2016), https://doi.org/10.3233/SW-150176
27. Tran, T., Ladwig, G., Rudolph, S.: Managing structured and semistructured RDF data using structure indexes. IEEE Trans. Knowl. Data Eng. **25**(9), 2076–2089 (2013). https://doi.org/10.1109/TKDE.2012.134
28. Vaigh, C.B.E., Goasdoué, F.: A well-founded graph-based summarization framework for description logics. In: Description Logics. vol. 2954. CEUR-WS.org (2021), http://ceur-ws.org/Vol-2954/paper-8.pdf
29. Wikimedia Foundation: Wikidata. https://www.wikidata.org/ (2022)

Appendix

A Additional Dataset Statistics

Table 5 lists degree statistics of the datasets, where $d(G)$ is the average degree of vertices in G, $d_G(v)$ the degree of a vertex $v \in V$, and $\Delta(G)$ the maximum degree in G.

Graph	$\mu(d_G(v))$	$\Delta(G)$	$\mu(d_G^-(v))$	$\Delta^-(G)$	$\mu(d_G^+(v))$	$\Delta^+(G)$
L'mat100M	5.89 ± 569	1,570,748	2.95 ± 559	1,570,748	2.95 ± 108	545,688
BSBM100M	$9.76 \pm 1,144$	2,273,014	5.04 ± 1144	2,273,014	5.04 ± 6	76
BTC150M	$58.43 \pm 5,655$	5,629,275	29.22 ± 504	283,686	$29.22 \pm 5,629$	5,628,254
BSBM1B	$10.58 \pm 3,862$	23,924,441	5.46 ± 3862	23,924,441	5.46 ± 6	85
BTC2B	$48.40 \pm 17,119$	65,879,409	24.20 ± 1556	3,856,778	$24.20 \pm 17,038$	65,878,298

Table 5: Degree Statistics of the Datasets.

Laundromat100M has the smallest average and maximum values for both degree and in-degree. BTC150M has the highest average and maximum degrees and the highest maximum in-degree. BSBM100M has the largest maximum in-degree and smallest maximum out-degree of our datasets.

We note that standard deviations in Table 5 are surprisingly large. This is due to some vertices having degrees orders of magnitude larger than the average, as shown by the maximum degrees.

We note that, in the real-world datasets, average total degree is average in-degree plus average out-degree, to the precision quoted. This is expected, because every edge adds one to the total degree of each of its endpoints. This is not the case for the two versions of the synthetic BSBM dataset. This dataset contains relatively many self-loops and, by convention, a self-loop adds 1 to a vertex's total degree, while also adding 1 to its in- and out-degrees.

B Results of the BRS Algorithm

As the BRS algorithm takes an additional initialization step (see Algorithm 3, lines 13–19), Table 6 shows the breakdown of the running time of the algorithms in initialization and computing the actual k bisimulation iterations. It is separated into average run time for initialization and computation of all ten iterations.

On all smaller datasets, i. e., Laundromat100M, BSBM100M, and BTC150M and the initialization step of BRS-Schätzle takes about 1 minute on average (Table 6). Regarding BRS-Kaushik, the initialization also takes around 1 minute on all smaller datasets (Table 6). Regarding the larger dataset BSBM1B, the BRS-Schätzle algorithm takes 10.20 minutes for initialization and 44.24 minutes for the 10 iterations. On BTC2B the algorithm needs 16.22 minutes to initialize

	BRS-Schätzle et al.		BRS-Kaushik et al.	
	Init.	Iterations	Init.	Iterations
Laundromat100M	1.10	4.46	1.10	4.50
BSBM100M	1.00	5.46	1.05	4.15
BTC150M	1.47	2.61	1.44	3.10
BSBM1B	10.20	44.24	10.43	53.57
BTC2B	16.22	45.74	16.29	69.63

Table 6: Initialization and bisimulation iteration (minutes) of average total run time for BRS.

the graph and 45.74 minutes to compute the 10 iterations. The BRS-Kaushik takes on BSBM1B 10.43 minutes for initialization and 53.57 minutes for the 10 iterations. On BTC2B, the BRS-Kaushik algorithm needs 16.29 minutes for initialization and 69.93 minutes for the 10 iterations.

Native Kaushik also has an initialization step, where the vertices are partitioned based on their label. This partitioning is done in parallel and took less than a second, so we do not account for it separately.

C Discussion of the Results of the Kaushik Algorithm

Experimental results for native Kaushik indicate that the algorithm does not scale linearly with the input graph's size. Its worst-case complexity is $\mathcal{O}(k \cdot m)$ [16], where m is the number of edges in the input graph.

A reason for the long running time of native Kaushik on Laundromat100M could be the size of the initial partition P_0. It depends on the number of different label sets present in the graph's vertices (Definition 2). Laundromat100M contains $33,431$ different label sets, which is much higher than for BTC150M with 69 and BSBM100M with $1,289$ (Table 2). As a consequence, the algorithm has to perform stability checks and (potential) splits on blocks more often than on the other datasets. This does not contradict the similar run times on BTC150M and BSBM100M (69 label sets vs. $1,289$ label sets). First, BSBM100M reaches full bisimulation w.r.t. Definition 2 in iteration seven and hence execution is stopped early. Second, as can be seen in Figure 2e, the iteration time starts to decrease rapidly from iteration four to five on BSBM100M, going down from about 20 minutes to about 1 minute. This indicates that the partition hardly changes any further and therefore only a small number of stability checks and splits are performed in iterations five to seven.

Native Kaushik operates on the blocks of the graph's current partition. In each iteration i, the algorithm produces a partition P_i, which is stable w.r.t. all the blocks in P_{i-1} [16]. Consequently, if a block is split into two new blocks, these must also be checked for stability in that iteration (see also discussion in Section 5.3). Hence, the more blocks are split in an iteration, the more steps must be performed by the algorithm. In addition, bisimulation relationships between the vertices change far more often in early iterations [25]. This explains

the specific shape of the run time curve (Figures 2a, 2c and 2e) for the native Kaushik et al. algorithm. The exception is BTC150M (Figure 2c), where the iteration time first decreases from iteration one to two, indicating that the 1- and 2-bisimulations of this graph are very similar.

D Detailed Results of Experiments

We provide the detailed experimental results. The results are split into results for the smaller datasets (Laundromat100M, BTC150M, BSBM100M) shown in Tables 7 to 9. The results for the larger datasets (BTC2B, BSBM1B) are shown in Tables 10 and 11. The tables for BRS include an additional iteration 0, which corresponds to the initialization routine of the algorithm.

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.1	0.4	0.5	0.5	0.4	0.4	0.6	0.6	0.5	0.5	0.5	6.0	0.49	0.07
2	1.1	0.3	0.4	0.3	0.3	0.3	0.6	0.4	0.3	0.3	0.3	4.6	0.35	0.09
3	1.1	0.4	0.5	0.4	0.7	0.6	0.5	0.4	0.4	0.4	0.4	5.8	0.47	0.10
4	1.1	0.4	0.5	0.4	0.4	0.7	0.5	0.5	0.4	0.4	0.5	5.8	0.47	0.09
5	1.1	0.4	0.4	0.4	0.4	0.8	0.4	0.4	0.4	0.4	0.5	5.6	0.45	0.12
μ	1.10	0.38	0.46	0.40	0.44	0.56	0.52	0.46	0.40	0.40	0.44	**5.56**		
σ	0.00	0.04	0.05	0.06	0.14	0.19	0.07	0.08	0.06	0.06	0.08	**0.50**		

(a) BRS algorithm executed with GSM Schätzle

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.1	0.8	0.7	0.8	0.8	0.7	0.7	0.7	0.7	0.8	7.8	0.78	0.12
2	1.1	0.8	0.7	0.8	0.7	0.7	0.7	0.7	0.7	0.7	7.6	0.76	0.12
3	0.7	0.8	1.1	0.8	0.7	0.8	0.8	0.7	0.7	0.7	7.8	0.78	0.12
4	0.7	0.8	1.1	0.8	0.7	0.8	0.7	0.7	0.7	0.7	7.7	0.77	0.12
5	0.7	0.7	1.1	0.8	0.7	0.8	0.8	0.7	0.7	0.7	7.7	0.77	0.12
μ	0.86	0.78	0.94	0.80	0.72	0.76	0.74	0.70	0.70	0.72	**7.72**		
σ	0.00	0.00	0.00	0.08	0.04	0.07	0.00	0.00	0.05	0.04	**0.07**		

(b) Schätzle algorithm

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.1	0.3	0.4	0.4	0.4	0.4	0.4	0.6	0.7	0.5	0.5	5.7	0.46	0.11
2	1.1	0.3	0.4	0.3	0.3	0.4	0.5	0.7	0.4	0.4	0.5	5.3	0.42	0.12
3	1.1	0.3	0.4	0.4	0.5	0.7	0.5	0.5	0.4	0.4	0.5	5.7	0.46	0.10
4	1.1	0.3	0.4	0.4	0.4	0.4	0.7	0.6	0.5	0.4	0.5	5.7	0.46	0.11
5	1.1	0.3	0.4	0.4	0.4	0.7	0.4	0.5	0.4	0.4	0.6	5.6	0.45	0.11
μ	1.10	0.30	0.40	0.38	0.40	0.52	0.50	0.58	0.48	0.42	0.52	**5.60**		
σ	0.00	0.00	0.00	0.04	0.06	0.15	0.11	0.07	0.12	0.04	0.04	**0.15**		

(c) BRS algorithm executed with GSM Kaushik

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	11.4	21.2	35.1	60.0	85.5	106.7	98.8	81.8	59.3	44.1	603.9	60.39	30.9
2	13.6	22.7	35.8	66.8	76.8	91.9	85.6	69.6	52.1	35.0	549.9	54.99	25.81
3	10.9	19.3	31.9	53.7	81.3	100.1	98.7	79.9	62.5	42.5	580.8	58.08	30.21
4	11.2	21.1	33.3	58.2	78.6	100.5	98.1	85.1	61.6	41.2	588.9	58.89	30.04
5	11.8	21.3	34.1	55.1	81.8	107.7	97.0	91.9	69.0	40.1	609.8	60.98	31.81
μ	11.78	21.12	34.04	58.76	80.80	101.38	95.64	81.66	60.90	40.58	**586.66**		
σ	0.96	1.08	1.37	4.59	2.97	5.67	5.06	7.28	5.45	3.09	**21.09**		

(d) Kaushik algorithm

Table 7: Detailed Results (minutes) on Laundromat100M for 10-bisimulation.

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.4	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	2.4	0.10	0.00
2	1.4	0.4	0.4	0.4	0.8	0.5	0.4	0.4	0.4	0.5	0.4	6.0	0.46	0.12
3	1.7	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.6	3.3	0.16	0.15
4	1.4	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.4	2.8	0.14	0.09
5	1.7	0.4	0.4	0.4	0.6	0.4	0.4	0.4	0.4	0.4	0.4	5.9	0.42	0.06
μ	1.47	0.26	0.22	0.22	0.34	0.24	0.22	0.22	0.22	0.24	0.38	**4.08**		
σ	0.14	0.12	0.15	0.15	0.3	0.17	0.15	0.15	0.15	0.17	0.16	**1.55**		

(a) BRS algorithm executed with GSM Schätzle

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	0.6	0.6	0.7	0.8	0.7	0.6	0.6	0.6	0.6	0.6	6.4	0.64	0.07
2	0.5	0.6	0.7	0.8	0.6	0.6	0.6	0.6	0.6	0.6	6.2	0.62	0.07
3	0.6	0.6	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	6.4	0.64	0.08
4	0.5	0.6	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	6.3	0.63	0.09
5	0.5	0.5	0.6	0.7	0.6	0.5	0.5	0.5	0.5	0.5	5.4	0.54	0.07
μ	0.54	0.58	0.72	0.78	0.62	0.58	0.58	0.58	0.58	0.58	**6.14**		
σ	0.05	0.04	0.07	0.04	0.04	0.04	0.04	0.04	0.04	0.04	**0.38**		

(b) Schätzle algorithm

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.4	0.4	0.4	0.3	0.4	0.6	0.4	0.4	0.4	0.4	0.4	5.5	0.41	0.07
2	1.4	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.7	3.0	0.16	0.18
3	1.4	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.5	2.8	0.14	0.12
4	1.4	0.4	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	5.6	0.42	0.04
5	1.4	0.4	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.5	5.8	0.44	0.07
μ	1.44	0.28	0.34	0.30	0.28	0.32	0.28	0.28	0.28	0.28	0.50	**4.54**		
σ	0.14	0.15	0.21	0.18	0.15	0.19	0.15	0.15	0.15	0.15	0.11	**1.34**		

(c) BRS algorithm executed with GSM Kaushik

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	3.9	0.9	3.7	12.2	17.0	17.8	12.7	6.6	2.8	1.0	78.6	7.86	6.17
2	3.8	0.9	3.7	11.2	16.8	17.0	12.9	6.0	2.9	1.5	76.7	7.67	5.92
3	4.1	0.9	3.9	11.7	17.6	17.6	12.6	6.1	3.1	1.1	78.7	7.87	6.14
4	3.9	0.9	4.0	14.1	20.5	18.5	12.2	6.0	2.5	1.2	83.8	8.38	6.95
5	3.6	0.9	3.5	10.8	15.5	16.0	12.0	6.3	2.7	1.0	72.3	7.23	5.55
μ	3.86	0.90	3.76	12.00	17.48	17.38	12.48	6.20	2.80	1.16	**78.02**		
σ	0.16	0.00	0.17	1.15	1.66	0.84	0.33	0.23	0.20	0.19	**3.71**		

(d) Kaushik algorithm

Table 8: Detailed Results (minutes) on BTC150M for 10-bisimulation.

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.0	0.5	0.5	0.5	0.6	0.8	0.5	0.5	0.5	0.5	0.4	6.3	0.53	0.10
2	1.0	0.5	0.6	0.5	0.5	0.9	0.5	0.5	0.5	0.5	0.5	6.5	0.55	0.12
3	1.0	0.5	0.6	0.5	0.5	0.9	0.5	0.5	0.5	0.5	0.5	6.5	0.55	0.12
4	1.0	0.5	0.5	0.5	0.6	0.9	0.5	0.5	0.5	0.5	0.5	6.5	0.55	0.12
5	1.0	0.5	0.5	0.5	0.9	0.5	0.5	0.5	0.5	0.6	0.5	6.5	0.55	0.12
μ	1.00	0.50	0.54	0.50	0.62	0.80	0.50	0.50	0.50	0.52	0.48	**6.46**		
σ	0.00	0.00	0.05	0.00	0.15	0.15	0.00	0.00	0.00	0.04	0.04	**0.08**		

(a) BRS algorithm executed with GSM Schätzle

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	0.9	1.3	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	9.4	0.94	0.12
2	0.9	1.0	1.2	0.9	1.0	0.9	0.9	0.9	0.9	0.9	9.5	0.95	0.09
3	0.9	0.9	1.2	0.9	0.9	0.9	0.9	0.9	0.9	0.9	9.3	0.93	0.09
4	0.9	1.0	1.2	0.9	0.9	0.9	0.9	0.9	0.9	0.9	9.4	0.94	0.09
5	0.9	1.0	1.2	0.9	0.9	0.9	0.9	0.9	0.9	0.9	9.4	0.94	0.09
μ	0.90	1.04	1.14	0.90	0.92	0.90	0.90	0.90	0.90	0.90	**9.40**		
σ	0.00	0.14	0.12	0.00	0.04	0.00	0.00	0.00	0.00	0.00	**0.06**		

(b) Schätzle algorithm

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	1.0	0.4	0.5	0.4	0.4	0.6	0.7	0.4	0.4	0.4	0.4	5.6	0.46	0.10
2	1.0	0.3	0.4	0.3	0.3	0.3	0.3	0.5	0.5	0.3	0.3	4.5	0.35	0.08
3	1.1	0.4	0.4	0.4	0.5	0.7	0.5	0.4	0.4	0.4	0.5	5.7	0.46	0.09
4	1.0	0.4	0.5	0.4	0.4	0.6	0.6	0.4	0.4	0.4	0.4	5.5	0.45	0.08
5	1.1	0.3	0.4	0.3	0.3	0.3	0.3	0.5	0.5	0.3	0.4	4.7	0.36	0.08
μ	1.05	0.36	0.44	0.36	0.38	0.50	0.48	0.44	0.44	0.36	0.40	**5.20**		
σ	0.05	0.05	0.05	0.05	0.07	0.17	0.16	0.05	0.05	0.05	0.06	**0.50**		

(c) BRS algorithm executed with GSM Kaushik

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	8.0	16.5	33.9	19.0	0.8	0.6	0.6	0.0	0.0	0.0	79.4	7.94	11.03
2	8.8	17.2	29.3	22.1	0.9	0.6	0.6	0.0	0.0	0.0	79.5	7.95	10.44
3	8.2	16.2	30.7	22.6	0.8	0.7	0.7	0.0	0.0	0.0	79.9	7.99	10.71
4	8.0	19.0	26.4	21.5	0.8	0.6	0.6	0.0	0.0	0.0	76.9	7.69	9.97
5	9.2	15.9	25.8	20.6	0.8	0.6	0.6	0.0	0.0	0.0	73.5	7.35	9.43
μ	8.44	16.96	29.22	21.16	0.82	0.62	0.62	0.00	0.00	0.00	**77.84**		
σ	0.48	1.11	2.96	1.27	0.04	0.04	0.04	0.00	0.00	0.00	**2.41**		

(d) Kaushik algorithm

Table 9: Detailed Results (minutes) on BSBM100M for 10-bisimulation.

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	16.0	4.7	5.2	4.9	5.0	5.6	5.1	5.1	5.1	5.1	5.9	67.7	5.17	0.33
2	16.0	4.5	5.1	4.9	5.0	5.9	5.0	5.0	5.8	5.1	5.3	67.6	5.16	0.40
3	16.4	1.6	1.8	1.7	1.9	2.2	1.9	1.9	1.9	1.9	6.0	39.2	2.28	1.25
4	16.0	4.5	5.1	4.9	5.2	5.5	5.1	5.0	5.8	5.2	5.4	67.7	5.17	0.33
5	16.4	4.6	4.9	4.8	6.2	5.7	5.0	5.0	4.9	5.0	5.1	67.6	5.12	0.45
μ	16.22	3.98	4.42	4.24	4.66	4.98	4.42	4.40	4.70	4.46	5.54	**61.96**		
σ	0.16	1.19	1.31	1.27	1.45	1.4	1.26	1.25	1.45	1.28	0.35	**11.38**		

(a) BRS algorithm executed with GSM Schätzle

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	8.3	8.7	8.7	9.2	8.7	8.7	8.7	8.7	8.7	8.9	87.3	8.73	0.21
2	8.1	8.3	8.3	8.8	8.3	8.3	8.3	8.3	8.3	8.3	83.3	8.33	0.17
3	7.9	8.2	8.2	8.2	8.1	8.2	8.1	8.6	8.1	8.2	81.8	8.18	0.17
4	7.9	8.3	8.3	8.4	8.4	8.7	8.8	8.4	8.4	8.5	84.1	8.41	0.23
5	7.9	8.2	8.1	8.2	8.3	8.3	8.3	8.3	8.3	8.3	82.2	8.22	0.12
μ	8.02	8.34	8.32	8.56	8.36	8.44	8.44	8.46	8.36	8.44	**83.74**		
σ	0.16	0.19	0.20	0.39	0.20	0.22	0.27	0.16	0.20	0.25	**1.96**		

(b) Schätzle algorithm

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	16.1	1.9	2.5	3.8	7.0	7.0	7.5	7.2	7.4	7.2	7.3	74.9	5.88	2.11
2	16.1	2.1	2.7	4.2	7.0	8.3	7.5	8.0	7.6	8.0	11.8	83.3	6.72	2.77
3	16.6	2.4	2.8	4.3	7.9	7.8	8.0	7.8	7.8	7.8	10.9	84.1	6.75	2.55
4	16.1	1.9	2.6	3.9	7.1	7.3	7.7	7.2	7.3	7.1	7.1	75.3	5.92	2.10
5	16.6	5.2	5.9	7.4	11.2	11.1	11.2	10.7	10.6	10.8	11.3	112.0	9.54	2.28
μ	16.29	2.70	3.30	4.72	8.04	8.30	8.38	8.18	8.14	8.18	9.68	**85.92**		
σ	0.20	1.26	1.30	1.35	1.62	1.47	1.42	1.30	1.24	1.35	2.05	**13.60**		

(c) BRS algorithm executed with GSM Kaushik

Table 10: Detailed Results (minutes) on BTC2B for 10-bisimulation.

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	10.1	3.9	4.7	4.3	4.3	4.5	5.3	4.7	5.1	4.7	5.7	57.3	4.72	0.50
2	10.1	2.9	3.7	3.3	3.4	3.3	4.1	4.1	4.1	3.8	3.1	45.9	3.58	0.42
3	10.0	3.9	4.8	4.3	4.4	4.5	5.2	4.7	4.9	4.5	4.2	55.4	4.54	0.36
4	10.1	3.9	4.8	4.4	4.4	4.5	5.2	4.9	5.2	4.5	5.8	57.7	4.76	0.51
5	10.0	3.9	4.8	4.4	4.4	4.6	5.4	4.8	5.0	4.4	4.2	55.9	4.59	0.41
μ	10.20	3.70	4.56	4.14	4.18	4.28	5.04	4.64	4.86	4.38	4.60	**54.44**		
σ	0.62	0.40	0.43	0.42	0.39	0.49	0.48	0.28	0.39	0.31	1.02	**4.35**		

(a) BRS algorithm executed with GSM Schätzle

Run	**Iteration**										**Aggregates**		
	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	9.0	8.1	8.1	8.2	8.3	8.2	8.2	8.5	8.2	8.2	83.0	8.30	0.26
2	10.0	8.3	8.2	8.3	8.4	8.5	8.4	8.4	8.3	8.3	85.1	8.51	0.50
3	8.8	8.9	8.9	9.0	9.0	12.5	9.3	9.0	9.0	9.0	93.4	9.34	1.06
4	8.1	8.2	8.4	8.2	8.3	8.6	8.7	8.2	8.2	8.2	83.1	8.31	0.19
5	8.5	8.4	8.3	8.4	8.5	9.2	8.5	8.5	8.5	8.5	85.3	8.53	0.23
μ	8.88	8.38	8.38	8.42	8.50	9.40	8.62	8.52	8.44	8.44	**85.98**		
σ	0.64	0.28	0.28	0.30	0.26	1.58	0.38	0.26	0.30	0.30	**3.83**		

(b) Schätzle algorithm

Run	**Iteration**											**Aggregates**		
	0	1	2	3	4	5	6	7	8	9	10	Σ	μ	σ
1	10.7	4.0	5.6	5.8	5.9	6.0	5.8	5.8	5.8	6.3	5.0	66.7	5.60	0.62
2	10.7	3.2	5.2	5.5	5.4	5.5	5.5	5.5	5.4	6.3	5.6	63.8	5.31	0.75
3	10.3	3.1	5.0	5.2	5.2	5.1	5.1	5.2	5.2	6.0	4.0	59.4	4.91	0.76
4	10.7	3.2	5.1	5.2	5.5	5.4	5.3	5.3	5.3	5.7	5.1	61.8	5.11	0.66
5	10.3	4.1	5.8	6.1	6.1	6.1	6.1	6.1	6.1	6.6	4.9	68.3	5.80	0.70
μ	10.43	3.52	5.34	5.56	5.62	5.62	5.56	5.58	5.56	6.18	4.92	**64.00**		
σ	0.27	0.44	0.31	0.35	0.33	0.38	0.36	0.33	0.34	0.31	0.52	**3.22**		

(c) BRS algorithm executed with GSM Kaushik

Table 11: Detailed Results (minutes) on BSBM1B for 10-bisimulation.

Subject: ICGT 2023 *conditional* acceptance notification for paper 1
From: "ICGT 2023" <icgt2023@easychair.org>
Date: 22.04.23, 09:17
To: Ansgar Scherp <mail@ansgarscherp.net>

Dear Ansgar,

Congratulations!

We are delighted to inform you that your paper, "Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis", has been **conditionally accepted** for presentation at ICGT'23 and inclusion in the conference proceedings.

You will find your reviews at the end of this email. The specific conditions for your paper's acceptance are summarised in the meta-review.

Instructions on submitting your revised paper will be emailed to you soon. In the meantime, please note that: (1) we request your final version by **4th May** so that referees have time to check it before the camera-ready deadline; and (2) you will be allowed **one additional page** so that you can sufficiently address the requirements of your referees and meta-review.

Please note that if your revised paper is accepted, then at least one author of the paper **must attend** ICGT (taking place in Leicester, UK) for your paper to be included in the proceedings. We will inform you once registration details are posted to the STAF website (https://conf.researchr.org/home/staf-2023).

Finally, if you have other relevant published work (in a journal, book, or other conference) that you would like to present at ICGT'23, please consider submitting to our so-called "journal-first" track (https://conf.researchr.org/track/icgt-2023/icgt-2023-journal-first). The submission process is very straightforward – we just ask for your proposal by 25th April to help our programme planning.

Best wishes,
Chris and Maribel

SUBMISSION: 1
TITLE: Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis

------------------------ METAREVIEW ------------------------
For acceptance of this paper, we require the following four issues to be addressed:

* (Review #1) A more elaborate comparison to related work is needed to establish the novelty of the contribution. (Please see the review for some specific pointers.)

* (Review #1) The algorithms, in particular the parallel BRS algorithm has to be defined more carefully. In particular it uses functions such as MergeAndHash and an oplus operation. What exactly are these operators? Are there any requirements on the hash function? This is all left quite open and unspecified. Also: I would have been interested in more details of the parallelization, which seems to be the main contribution. In particular: how does the Signal/Collect paradigm work and why is it suitable for what you are doing?

* (Review #1) You are comparing the so-called Schätzle algorithm, which computes an edge-labeled forward k-bisimulation with the Kaushik algorithm determining a vertex-labeled backward k-bisimulation. First I wonder why these two variants: why is forward combined with edge labels and backward with vertex labels? Second: in my opinion it does not make too much sense to compare the runtimes of these two algorithm, simply because they compute different relations. It would have made more sense to compare algorithms which both compute edge-labeled/forward or vertex-labeled/forward or ...

* (Review #2) for k-bisimulation, the authors choose the k=10 case only. Without justification or analysis, this looks like an arbitrary choice, and one may wonder if 10 isn't too small (i.e., bisimulation doesn't summarize the graph well) and if 10 isn't too large (i.e., the same result could have been obtained with a smaller k). A proper evaluation should analyze the change of the size of the summary and the execution time, both with respect to k. In particular, it is important to see if k=10 is just a rough summary or close to full bisimulation.

Note that the fourth point implies re-running the program with some different values of 'k'
and recording statistics. Please let us know why if this would not be possible to do.

---------- REVIEW 1 ----------
SUBMISSION: 1
TITLE: Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis
AUTHORS: Jannik Rau, David Richerby and Ansgar Scherp

---------- Overall evaluation ----------
SCORE: 0 (borderline paper)
----- TEXT:
This paper is concerned with the parallelization of an algorithm for
summarizing graphs, in particular the computation of k-bisimulations
(bisimulation up to k steps) respectively generalizations thereof
(based on complex schema elements, CSEs).

In principle the evaluation of such algorithms is a welcome topic for
ICGT, however I find the contents a bit thin and I see a few other
shortcomings of the paper:

- It is true that computing bisimilarity efficiently requires some care
  and there is a fairly long history of efficient bisimilarity algorithms,
  starting with Kannelakis/Smolka and Paige/Tarjan. Hence, I would not say
  that the algorithms are particular novel, although parallelization
  has maybe been studied in less detail.

  Hence, a more elaborate comparison to related work is needed
  to establish the novelty of the contribution. For instance

  Jan Martens, Jan Friso Groote, Lars B. van den Haak, Pieter Hijma,
  Anton Wijs: A Linear Parallel Algorithm to Compute Bisimulation and
  Relational Coarsest Partitions. FACS 2021: 115-133

  also deals with parallelization, whereas

  Jelle Hellings, George H. L. Fletcher, Herman J. Haverkort:
  Efficient external-memory bisimulation on DAGs. SIGMOD Conference
  2012: 553-564

  introduces the idea of working with hash values.

  I am not saying that these papers have to be cited (although they
  both look relevant), but to indicate that maybe there are not so many
  novel ideas in the present paper.

- The algorithms, in particular the parallel BRS algorithm has
  to be defined more carefully. In particular it uses functions
  such as MergeAndHash and an oplus operation. What exactly are
  these operators? Are there any requirements on the hash function?
  This is all left quite open and unspecified.
  Also: I would have been interested in more details of the
  parallelization, which seems to be the main contribution. In particular:
  how does the Signal/Collect paradigm work and why is it suitable
  for what you are doing?

- Although the algorithms are fairly straightforward, I would have
  appreciated a correctness argument. This is not entirely obvious
  for Algorithm 3 (parallel BRS algorithm) due to the involvement of
  the hash function.

- You are comparing the so-called Schätzle algorithm, which computes an
  edge-labeled forward k-bisimulation with the Kaushik algorithm
  determining a vertex-labeled backward k-bisimulation.
  First I wonder why these two variants: why is forward combined with
  edge labels and backward with vertex labels?
  Second: in my opinion it does not make too much sense to compare the
  runtimes of these two algorithm, simply because they compute different
  relations. It would have made more sense to compare algorithms

which both compute edge-labeled/forward or vertex-labeled/forward or ...

- It is not such a big surprise that a parallel version of the
  algorithm is faster, although it is based on a more general
  specification language (FLUID). I believe that it would have been
  interesting to say more about FLUID and how it can be handled by
  your algorithm. But the authors here mainly refer to another paper.

Hence I am not sure what I can learn from this paper. It is certainly
possible to rewrite it into a more interesting contribution, but in
its present form I can not really recommend acceptance.

More comments:

- p.5: Please explain how you quotient a graph once the equivalence
  is computed. Is there an edge between two equivalence classes iff
  there is an edge between any two nodes in these classes?

- p.7, Algorithm 1: How do you initialize ID_{P_i}?

- p.9, l.-8: What is L?

- p.11 ff.: Say more clearly that the native algorithms are sequential.
  (I hope that I understood this correctly.)

- Sct.7 (Discussion): The discussion mainly summarizes the runtime
  results, but does not really explain why you get these results and
  whether there is a potential for optimization. There are a few
  more details in Appendix C, but I would have expected the more
  interesting conclusions in the main body of the paper.

Minor comments:

- p.5, l.-2: "combines of three equivalence relations"
  -> "consists of three equivalence relations"

- p.6, l.-6: Use double subscripts: B_{i1} -> B_{i_1}

- p.7: ID occurs in two types of font (roman and italics)

- p.12, l.-6 of Sct. 5.2: "We run ..." -> "We ran ..."

---------- REVIEW 2 ----------
SUBMISSION: 1
TITLE: Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis
AUTHORS: Jannik Rau, David Richerby and Ansgar Scherp

---------- Overall evaluation ----------
SCORE: 1 (weak accept)
----- TEXT:
This paper is concerned with summarizing large graphs by means of bounded-depth bisimulation.
The key idea is use BRS as the core algorithm that encompasses existing relations and can be
executed in parallel. The presentation is detailed and easy to read (though some technical
concerns are listed below).

The experiments are detailed, but some key issues remain. Firstly, the authors say that an
advantage of the proposed algorithm is that it is parallelizable, but they do not show
parallel speedup or scalability (with respect to the number of cores), which is very important
for papers on parallel programs/algorithms for real data.

Secondly, for k-bisimulation, the authors choose the k=10 case only. Without justification or
analysis, this looks like an arbitrary choice, and one may wonder if 10 isn't too small (i.e.,
bisimulation doesn't summarize the graph well) and if 10 isn't too large (i.e., the same
result could have been obtained with a smaller k). A proper evaluation should analyze the
change of the size of the summary and the execution time, both with respect to k. In
particular, it is important to see if k=10 is just a rough summary or close to full
bisimulation.

To conclude, I do believe that the experiments should provide missing important information.

(Technical comments)

(Section 3.2) This definition of bisimulation is not a standard/original one (which should be a relation between two labelled transition systems). Some remark is appropriate to avoid confusion.

(Section 3.2) The definition of "forward-bisimilar" is split into two sentences, which is not appropriate for a "definition" (even if it's an informal one).

(Section 3.2) Similarly, the description of complete bisimulation ends with "and so on", which is not good because the crux of bisimulation lies in how this "and so on" is formulated!!

(Section 3.2) Definition 2 ends with "and vice versa" but Definition 1 does not.

(Section 3.3) The definition of id and T above (1) seems incorrect. They are not used as equivalence over vertices.

(Algorithm 3) When an algorithm is presented, it is standard to discuss its correctness, termination, and complexity. Any results or comments on these?

(Table 2) I don't understand why (in some graphs) r(l_V) is larger than |\Sigma_V| when l_V(v) is taken from \Sigma_V.

(Table 2) In some graphs, the standard deviation is so large and subtracting it from the mean would result in a negative value. Some comments would be appropriate.

(Other comments)

(page 1) What does BRS stands for? Please spell out.
(page 2, line 2) Can you give examples of arbitrary (but motivated)user-defined equivalence relations that can't be handed by k-bisimulation?
(page 4) "rounded rectangles": in Fig.1 they are not rounded!
(page 5, line -5) What are the "two kinds of operators"?
(page 5, line -2) "combines of" --> "consists of" or "combines"
(page 5, line -2) Please explain what the suffixes (s, p, o) stand for.
(page 5, line -1) The use of the comma and the semicolon is not appropriate.
(page 6, end of Section 3.3) Examples or some explanation of (1) and(2) would be appropriate.

---------- REVIEW 3 ----------
SUBMISSION: 1
TITLE: Computing k-Bisimulations for Large Graphs: A Comparison and Efficiency Analysis
AUTHORS: Jannik Rau, David Richerby and Ansgar Scherp

---------- Overall evaluation ----------
SCORE: 2 (accept)
----- TEXT:
The authors present a new algorithm to compute graph similarity, based on the notion of k-bisimulation. There are many applications for algorithms capable of computing graph similarity and, while the paper does not directly focus on graph ***transformations***, its topic is very relevant to the ICGT community and well in scope of the conference.

The proposed algorithm (named BRS) is generic and allows users to plug in their own approach on how to compute local vertex similarity. The authors demonstrate how two well-known algorithms for computing k-bisimulations can be reformulated using the BRS generic framework. A major selling point of this approach is that BRS is a parallel algorithm and can therefore be computed more efficiently than sequential algorithms. The authors report on experimental results comparing their own BRS-type implementations of two well known algorithms with their native implementations. Large scale synthetic and real-world data sets are used in these experiments. The results demonstrate that the BRS-type implementations are faster when compared to their native implementation. As a result, the authors recommend using BRS implementations of these algorithms.

Comments:

+ The topic is relevant and important to the ICGT community

+ The paper is easy to read and well organized
+ The evaluation uses synthetic as well as real-world data
+ The authors provide source code of their algorithm implementation

Minor comment:

There is a grammatical issue in the second paragraph of section 3.3: "... combines of three" should either be "... combines three" or "... consists of three"