Efficient Sink-Reachability Analysis via Graph Reduction

Jens Dietrich, Lijun Chang, Long Qian, Lyndon M. Henry, Catherine McCartin, and Bernhard Scholz

Abstract—The reachability problem on directed graphs, asking whether two vertices are connected via a directed path, is an elementary problem that has been well-studied. In this paper, we study a variation of the elementary reachability problem, called the *sink-reachability* problem, which can be found in many applications such as static program analysis, social network analysis, large scale web graph analysis, XML document link path analysis, and the study of gene regulation relationships. To scale sink-reachablity analysis to large graphs, we develop a highly scalable *sink-reachability preserving* graph reduction strategy for input sink graphs, by using a *composition* framework. That is, individual sink-reachability preserving condensation operators, each running in linear time, are pipelined together to produce graph reduction algorithms that result in close to maximum reduction, while keeping the computation efficient. Experiments on large real-world sink graphs demonstrate the efficiency and effectiveness of our compositional approach to sink-reachability preserving graph reduction with a reduction rate of up to 99.74 percent for vertices and a rate of up to 99.46 percent for edges.

Index Terms—Sink reachability, graph reduction, modular decomposition, dominator

1 Introduction

The basic reachability problem, asking whether two vertices u and v are connected via a directed path in a directed graph $G = (V, E)$, is a long-standing, well-studied problem [1]. One naïve approach is to pre-compute and store the full transitive closure of edges, which allows constant-time query processing. However, this approach requires quadratic worst-case space complexity, making it infeasible for large graphs. Another naïve approach is to online conduct a depth-first search (DFS) or breadth-first search (BFS), emanating from vertex u aiming to find vertex v, for verifying the reachability. However, the search approach exhibits a worst-case time complexity of $\mathcal{O}(|V| + |E|)$ for a single reachability query, which is inefficient for online query processing over large graphs. To overcome the complexity issues of the naïve approaches, indexing methods were introduced that perform a pre-processing step of the input graph. In the pre-processing step, a compact index structure is produced permitting fast access to the reachability information. Although many reachability indexing methods have been proposed, according to Jin *et al*. [2] most of the existing methods with index structures reach a scalability bottleneck for graphs with around one million vertices/edges. Graphs with millions of nodes and billions of edges have now become commonplace, e.g., social networks, web graphs, XML link graphs. To overcome the aforementioned scalability bottleneck, new methods have been developed that reduce the input graphs. By reducing/condensing the input graphs, indexing methods still continue to scale for larger graphs [1]. The aim of the reduction strategy is to produce a substantially smaller graph while preserving the reachability information of the input graph. We call such a strategy *reachability-preserving graph reduction*.

In this paper, we study the problem of sink reachability, a variation of the elementary graph reachability problem. The sink-reachability problem takes as an input a *sink graph* $G = (V, S, E)$. In G, the set of vertices is divided into sink vertices S and non-sink vertices V assuming that the two vertex sets are disjoint (i.e., $V \cap S = \emptyset$).[1] The sink-reachability problem seeks for the *sink-reachability function* $r : V \rightarrow 2^S$ that captures, which sinks are reachable from a non-sink vertex in the sink graph.

Our notion of sink reachability may be seen as a refinement of the reachability relation, to answer particular queries more efficiently than in the general construction. It can be applied to the analysis of ID/IDREF, XLink links and entity references in XML documents, social networks, and large scale web graphs. For example, resource access controls on social networks may be modelled via sink reachability [3]. By treating resources as sink nodes and access rules as edges, $r(u)$ gives all resources accessible by a user u, while a reversal of edges obtains all users which may access a resource. Similarly, friend relationships between users can be represented as nodes and edges respectively, with sinks being used to model social media "influencers". The analysis of influencers is of high value for brand management,

- *Jens Dietrich is with the Victoria University of Wellington, Wellington 6012, New Zealand. E-mail: jens.dietrich@vuw.ac.nz.*
- *Lijun Chang, Lyndon M. Henry, and Bernhard Scholz are with The University of Sydney, Sydney 2006, Australia. E-mail: {Lijun.Chang, bernhard.scholz}@sydney.edu.au, lyndonmhenry@gmail.com.*
- *Long Qian and Catherine McCartin are with the Massey University, Palmerston North 4474, New Zealand. E-mail: qianlong33@gmail.com, C.M.McCartin@massey.ac.nz.*

Manuscript received 6 July 2020; revised 30 November 2020; accepted 3 January 2021. Date of publication 18 January 2021; date of current version 6 October 2022.
(Corresponding author: Jens Dietrich.)
Recommended for acceptance by G. Chen.
Digital Object Identifier no. 10.1109/TKDE.2021.3052710

1. Note that, we assume that any successors/out-neighbors of sink vertices $s \in S$ are also contained in S, i.e., $N^+(s) \cap V = \emptyset$.

e-commerce, and social media marketing [3]. In the Semantic Web, graphs are found in the ID/IDREF links in XML documents or directly as RDF data. Efficiently performing various types of reachability query on such graph structured data is a topic of substantial investigation in the literature [4], [5], [6], [7]. Our notion of sink reachability is applicable to efficiently compute a range of queries in this domain.

A sink reachability problem arises when parsing structured data. Many expressive data formats have some notion of reference in order to avoid data redundancies. For instance, XML supports such references in the form of entity references. Those references can then be used by attackers to create malicious input that causes parsers to run out of time and/or memory as they have to evaluate an exponential number of paths, typically always resolving references to the same values via redundant paths. The classical vulnerability of this kind is CVE-2003-1564, better known as billion laughs. Similar attacks can be grafted for other data formats [8], [9]. The construction of the sink graph modelling relationships between references and values (with values being the sinks), and the use of effective algorithms to resolve references via sink reachability is a possible approach to avoid such attacks.

In biology, transcriptional regulatory networks model relationships between genes as directed graphs. Nodes that are connected by an edge model where one gene regulates the activity of another. Efficiently answering reachability queries of such networks gives key insights into transitive regulatory relationships [10]. Sink reachability problem may be readily applied here also, to speed up queries involving designated subsets of genes, and those that are regulated by or regulate them.

Besides asking $r(u)$ for a query vertex u, another typical sink-reachability query might ask whether two non-sink vertices u and v share at least one sink vertex, i.e., whether $r(u) \cap r(v) \neq \emptyset$. Reachability queries of this particular type find applications in static program analysis (in particular points-to analysis) [11], [12], [13], [14], in analysis of percolation patterns in large real-world networks [15], and in logistics applications [16]. For instance, consider the problem of points-to analysis [13] that uses a graph model to identify which program variables may point to which objects. In this graph model, the vertices represent program variables, sink vertices represent objects, and edges represent assignments. Sink-reachability is able to answer queries such as alias, i.e., two variables share at least one common sink vertex/object. This variation of the sink reachability problem has near-cubic running time,[2] although algorithms with a sub-cubic worst-case time complexity exists [17], [18], they are not very practical nor efficient in practice. We note that the sink-reachability query asking whether $r(u) \cap r(v) \neq \emptyset$ can be answered by existing elementary reachability query processing techniques as follows: Given a sink graph $G = (V, S, E)$, we create its reverse graph $\overleftarrow{G} = (V', S', \overleftarrow{E})$, where every vertex $v \in V$ has a copy $v' \in V'$ and every vertex $s \in S$ has a copy $s' \in S'$. Then, we concatenate G and $\overleftarrow{G}$, denoted $G \oplus \overleftarrow{G}$, by adding a directed edge (s, s') for every $s \in S$. It is easy to verify that $r(u) \cap r(v) \neq \emptyset$ if and only if u can reach v' in $G \oplus \overleftarrow{G}$. Thus, all existing techniques for the elementary reachability problem can be applied here. In particular, the reachability preserving reduction techniques proposed in [1] can be applied to reduce the graph. However, this is not necessarily effective as it ignores the special properties of the sink-reachability function.

2. We refer here to the computation of transitive closure for a directed graph, which is equivalent to matrix multiplication.

In this work, we introduce a highly scalable *sink-reachability preserving* graph reduction strategy to scale the sink-rechability problem to large graphs. The reduction strategy uses a *composition* framework consisting of condensation operators. We first design four sink-reachability preserving condensation operators: strongly connected component condensation (operator S), condensation via dominators (operator D), and condensation via two special cases of module (operators $\mathsf{M_i}$ and $\mathsf{M_c}$). We devise algorithms to perform each condensation operator in linear time. We also investigate the properties of these four condensation operators, and how to compose them by noting that each condensation operator maps sink graphs to sink graphs. Specifically, we prove the existence of a unique *fixpoint*.

Theorem 1. *Given any sink graph G, all maximal condensation sequences over $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c})$ reduce G to the same sink graph, where maximal means further applying any operator on the resulting sink graph has no effect.*

As each of our four condensation operators runs in linear time, the running time of a condensation sequence grows linearly to its length. Based on the fixpoint theorem, we propose maximal condensation sequences that are at most 3 times longer than the shortest condensation sequence, for any given sink graph G.

The main contributions of this work are as follows:

- As far as we know, we are the first to study the sink-reachability preserving graph reduction problem.
- We introduce four condensation operators, and present algorithms to conduct each operator in linear time. (Section 4.1)
- We state a fixpoint theorem for composing the condensation operators, showing that any sequence of operators leads to a unique fixpoint, i.e., applying a maximum sequence will result in the same reduced graph. (Section 4.2)

We conduct extensive experiments with large-scale graphs from static program analysis, social networks and web graphs from SNAP. Experiments on large real-world sink graphs demonstrate the efficiency and effectiveness of our composition framework. We can show compression rate of up to 99.74 percent for vertices and a compression rate of up to 99.46 percent for edges.

2 Related Work

A classic reachability problem asks, whether two vertices u and v are connected via a direct path in a graph $G = (V, E)$. Many reachability indexing methods have been proposed for a classic reachability problem. According to Jin *et al.* [2] most of the existing methods reach a scalability bottleneck around one million vertices/edges. In further support of this view, Yildirim *et al.* [19] have shown that very large graphs are not supported by the vast majority of indexing methods. Veloso *et al*. [20] claim to have found that the only indexing methods that can handle graphs with more than 100,000 edges are Nuutila's INTERVAL [21], [22], GRAIL [19], FERRARI [23], TF-Label [24] and their own FELINE [20] indexing method.

Reduction strategies, where the aim is to produce a substantially smaller graph while preserving the original graph reachability properties, have recently been proposed in two studies pertaining to the classic reachability problem. In [25], Fan *et al*. present a reachability preserving equivalence reduction, where two vertices u and v are equivalent in a DAG G if and only if they have the same set of ancestors and the same set of descendants. The result of this reachability preserving reduction over a graph G is a smaller graph G_r obtained by replacing each set of equivalent vertices in G with a representative vertex in G_r. On a set of ten real-world graphs with $|V|$ ranging between $6K$ and $2.4M$ and $|E|$ ranging between $21K$ and $5.0M$, an average compression factor of 95 percent is achieved. However, the algorithm in [25] has both high time complexity $\mathcal{O}(|V|(|V|+|E|))$ and high space complexity $\mathcal{O}(|V|^2)$. Zhou *et al.* [26] recently speed up the reduction process by first computing a transitive reduction followed by an equivalence reduction. The transitive reduction removes from G all redundant edges to get the unique smallest DAG G_t satisfying that G_t has the same transitive closure as that of G. Although this transitive reduction still has $\mathcal{O}(|V|^3)$ time complexity in the worst case, in [26] heuristics are employed that work well in practice to efficiently compute G_t on real-world examples. Given G_t as input, an equivalence reduction computing G_r, as in [25], can be achieved in time $\mathcal{O}(|V|+|E_t|)$ and space $\mathcal{O}(|V|)$. It is shown in [26] that this combined strategy can be scaled to large real-world graphs.

The sink reachability problem that we formally define in this paper has mainly, so far, found applications in static program analysis, in particular points-to analysis [11], [13], [14] expressed as logic programs. The use of dominators as an adhoc reduction technique for points-to analysis was proposed in [12]. Dominators have also been employed as a reduction strategy in the context of instrumentation of code coverage testing in [27]. However, they were not stated as a generic graph reduction operator as we do in this paper, and our algorithm for dominator-based condensation is different.

Computing the modular decomposition of a directed graph is an active area of research. The current best time bound, $O(|V|+|E|)$ is achieved by the algorithm of McConnell and Montgolfier [28]. However, the existing algorithms are of theoretical interests only, and no implementation exists. In addition, the notion of module that we formally define in this paper, which considers only successors, rather than all neighbours, doesn't appear in the literature. We opt to utilise a simple, fast algorithm that requires repeated application to achieve a modular decomposition of an input graph G into maximum successor modules, allowing for trade-off between maximum reduction and computational efficiency.

3 Preliminary

We use $G=(V,E)$ to denote a directed graph consisting of a set V of vertices and a set E of edges. A directed edge from $u \in V$ to $v \in V$ is denoted by (u,v). For a vertex u, its set of out-neighbors is denoted by $N^+(u)=\{v \in V \mid (u,v) \in E\}$, and its set of in-neighbors is denoted by $N^-(u)=\{v \in V \mid (v,u) \in E\}$. The out-degree and in-degree of u are denoted by $d^+(u)=|N^+(u)|$ and $d^-(u)=|N^-(u)|$, respectively. A *path* in G from vertex $u \in V$ to vertex $v \in V$ is a sequence of vertices $(v_{i_1}, v_{i_2}, \ldots, v_{i_l})$ such that $u=v_{i_1}, v=v_{i_l}$, and $(v_{i_j}, v_{i_{j+1}}) \in E$ for all $1 \leq j < l$. We say that u can *reach* v in G, denoted $u \rightsquigarrow_G v$, if there is a path in G from u to v.

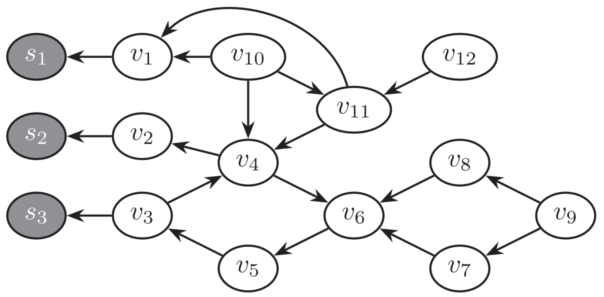


Fig. 1. A toy sink graph.

In this paper, we study sink graph, a special type of directed graph, and focus on the sink-reachability.

Definition 1. *A* sink graph, *denoted (V,S,E), is a directed graph $(V \cup S, E)$ where the set of vertices is partitioned into two disjoint subsets V (normal vertices) and S (sink vertices) such that there is no edge in G from S to V, i.e., $N^+(s) \cap V = \emptyset, \forall s \in S$.*

Definition 2. *Given a sink graph $G=(V,S,E)$, the* sink-reachability *of a vertex $u \in V$ is the set of sink vertices that u can reach, denoted $r_G(u)=\{s \in S \mid u \rightsquigarrow_G s\}$.*

For example, Fig. 1 shows a sink graph with $V=\{v_1,\ldots,v_{12}\}$ and $S=\{s_1,s_2,s_3\}$. The sink-reachability of v_1 is $r_G(v_1)=\{s_1\}$, and the sink-reachability of v_3 is $r_G(v_3)=\{s_2,s_3\}$.

The relation $\simeq_r \subseteq V \times V$, where $u \simeq_r v$ iff (if and only if) $r_G(u)=r_G(v)$, is an equivalence relation that is reflexive, symmetric, and transitive. Thus, if we contract each equivalence class of $\simeq_r$ into a super-vertex, the sink-reachability for all vertices is still preserved in the resulting graph, where the *equivalence class* of vertex $u \in V$ is $[u]_{\simeq_r}=\{v \in V \mid u \simeq_r v\}$. We define condensation for *any* equivalence relation as follows.

Definition 3. *Given a sink graph $G=(V,S,E)$ and an equivalence relation $\simeq \subseteq V \times V$, the* condensation *of G induced by $\simeq$ is the triple $G_\simeq=(V_\simeq,S_\simeq,E_\simeq)$ such that*

- *$V_\simeq=\{[u]_\simeq \mid u \in V\}$ represents classes of vertices,*
- *$S_\simeq=S$, and*
- *$E_\simeq=\{([u]_\simeq,[v]_\simeq) \mid (u,v) \in E \wedge [u]_\simeq \neq [v]_\simeq\}$*

where self-loops and parallel edges are removed, and the equivalence relation $\simeq$ is extended to include S by defining $[s]_\simeq=\{s\}$ for each $s \in S$.

For example, Fig. 2 shows the condensation of G induced by the equivalence relation that has two non-trivial equivalence classes: $\{v_{10},v_{11},v_{12}\}$ and $\{v_3,v_4,\ldots,v_9\}$. Note that $G_\simeq$ is also a sink graph. That is, *condensation maps sink graphs to sink graphs*.

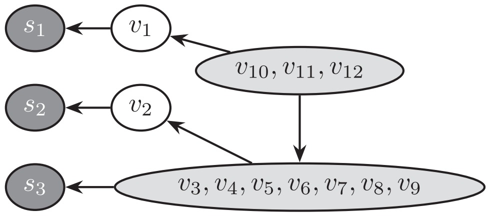


Fig. 2. Kernel condensation of the graph in Fig. 1.

Definition 4. *Given a sink graph $G = (V, S, E)$ and an equivalence relation $\simeq \subseteq V \times V$, the condensation $G_\simeq$ is* sink-reachability preserving *if it preserves the sink-reachability for all vertices of V, i.e., $r_G(u) = r_{G_\simeq}([u]_\simeq), \forall u \in V$.*

As a condensation reduces the graph size (i.e., $|V_\simeq| \leq |V|$ and $|E_\simeq| \leq |E|$), we refer to a sink-reachability preserving condensation as a *sink-reachability preserving graph reduction*. The condensation $G_{\simeq_r}$ is the smallest (in terms of vertex number) sink-reachability preserving reduction we can obtain. Thus, we call the relation $\simeq_r$ the *kernel equivalence relation* and the condensation $G_{\simeq_r}$ the *kernel condensation*. For example, the condensation in Fig. 2 is the kernel condensation of the graph in Fig. 1. However, directly computing the kernel condensation requires at least quadratic time in the worst case which is prohibitive for large graphs with millions of vertices. Thus, we resort to approximating the kernel condensation. We say that $G_\simeq$ *under-approximates* the kernel condensation $G_{\simeq_r}$ if $\simeq \subseteq \simeq_r$. It is easy to see that a condensation is sink-reachability preserving iff it under-approximates the kernel condensation.

Problem Statement. Given a sink graph $G = (V, S, E)$, in this paper we study the problem of sink-reachability preserving graph reduction, i.e., compute a condensation $G_\simeq$ that is small and under-approximates $G_{\simeq_r}$.

Without loss of generality, we assume that the input sink graph G satisfies $r_G(u) \neq \emptyset$ for every $u \in V$; otherwise, all such vertices with $r_G(u) = \emptyset$ can be removed from G in a pre-processing step in linear time. Frequently used notations are summarized in Table 1.

4 A Compositional Approach

In this section, we propose a compositional approach to sink-reachability preserving graph reduction. We first investigate four linear-time condensation operators in Section 4.1, and then compose them in Section 4.2.

TABLE 1
Frequently Used Notation

Notation	Description
(V, S, E)	Sink graph where $V \cap S = \emptyset$ and $N^+(s) \cap V = \emptyset, \forall s \in S$
$u \rightsquigarrow_G v$	u reach v in G, i.e., there is a path in G from u to v
$r_G(u)$	Sink reachability of u: $r_G(u) = \{s \in S \mid u \rightsquigarrow_G s\}$
$t(u)$	Topological number of vertex u in a DAG
$\simeq$	An equivalence relation $\simeq \subseteq V \times V$
$\simeq_r$	The kernel equivalence relation, where $u \simeq_r v$ iff $r_G(u) = r_G(v)$
$[u]_\simeq$	The equivalence class of u that is defined by $\simeq$
$G_\simeq$	sink reachability reduction w.r.t. $\simeq$
$\mathbb{C}$	A composition sequence of condensation operators

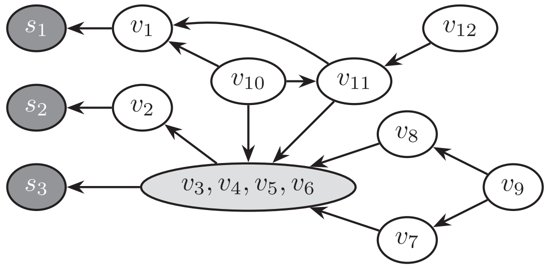


Fig. 3. SCC-Condensation of the graph in Fig. 1.

4.1 Linear-Time Condensation Operators

We study the condensations induced by SCC-, DOM-, IMOD-, and CMOD-equivalence relations in the following three subsections. We show that each of the four condensations under-approximates the kernel condensation, and can be conducted in linear time.

4.1.1 SCC-Condensation

Our first condensation is based on the concept of strongly connected component (SCC). An SCC of a directed graph is a maximal set of vertices such that every pair of its vertices can reach each other [29].

Definition 5. *Given a sink graph $G = (V, S, E)$, two vertices $u, v \in V$ are said to be* SCC-equivalent*, denoted $u \simeq_{\text{SCC}} v$, if they belong to the same SCC in G.*

The relation induced by SCC-equivalence is reflexive, symmetric, and transitive. Moreover, SCC-equivalent vertices are also equivalent in the kernel equivalence relation, i.e., $\simeq_{\text{SCC}} \subseteq \simeq_r$. Hence, the condensation induced by the SCC-equivalence relation, called SCC-condensation, is sink-reachability preserving. For the sink graph in Fig. 1, the only non-trivial SCC is $\{v_3, v_4, v_5, v_6\}$, and its SCC-condensation is shown in Fig. 3.

Algorithm 1. Algorithm for SccCondense (Operator S)

```
Require: A sink graph $G = (V, S, E)$
Ensure: SCC-condensation of $G$
1: Compute the set of all SCCs in $G$
2: Condense $G$ based on the equivalence classes defined by the SCCs
```

Each SCC defines an equivalence class under SCC-equivalence. To conduct SCC-condensation, we first compute all the SCCs and then contract each SCC into a super-vertex. The pseudocode is shown in Algorithm 1. It is well-known that the set of all SCCs in a directed graph can be computed in linear time, e.g., by Tarjan's algorithm [30]. Consequently, SCC-condensation can be conducted in linear time. In the following, we call Algorithm 1 the condensation operator S.

4.1.2 DOM-Condensation

Our second condensation is based on the concept of dominance. In order to define dominance on a sink graph $G = (V, S, E)$, we need to introduce an exit vertex $\perp$ to G, and add an edge from every sink vertex $s \in S$ to $\perp$, e.g., see Fig. 4. In the following, we assume that such an exit vertex $\perp$ always exists in G. Then, vertex $u \in V$ is said to *dominate* vertex $v \in V$ (or v is dominated by u), if every path from v to $\perp$ goes through u.

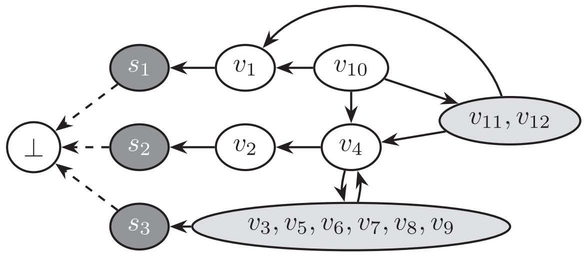


Fig. 4. DOM-Condensation of the graph in Fig. 1.

Definition 6. *Given a sink graph $G = (V, S, E)$, two vertices $u, v \in V$ are said to be* DOM-equivalent, *denoted $u \simeq_{\text{DOM}} v$, if either u dominates v or v dominates u in G.*

The relation induced by DOM-equivalence directly is not transitive. For example, it is possible that both u and w are dominated by another vertex v, but there is no dominance between u and w. We manually enforce transitivity, i.e., we also add (u, w) to the relation $\simeq_{\text{DOM}}$ if this happens. We call the resulting relation $\simeq_{\text{DOM}}$ the DOM-equivalence relation which is reflexive, symmetric, and transitive. It can be verified that $\simeq_{\text{DOM}} \subseteq \simeq_r$. Thus, the condensation induced the DOM-equivalence relation, called DOM-condensation, is sink-reachability preserving. For the sink graph in Fig. 1, v_{11} dominates v_{12}, and v_3 dominates $\{v_5, v_6, v_7, v_8, v_9\}$; its DOM-condensation is shown in Fig. 4.

It is known in [31], [32] that the dominance relationship among all vertices in a directed graph can be compactly represented by a dominator tree rooted at $\perp$, and the dominator tree can be constructed in linear time. Consequential, we can obtain the DOM-equivalence classes from the dominator tree in linear time. However, the linear-time dominator tree construction algorithms [32] are complicated. In this paper, we propose a simpler and practical algorithm to conduct DOM-condensation by

- directly obtaining the DOM-equivalence classes without constructing the dominator tree, and
- assuming that the sink graph is *acyclic* (which will be made possible in Section 4.2).

Recall that, for any directed acyclic graph (DAG) with vertices V and edges E, we can assign a unique *topological number* $t(u)$ to each vertex $u \in V$ such that $t(u) < t(v)$ holds for every $(u, v) \in E$, and then a *topological ordering* of V is the increasing ordering of V regarding their topological numbers [29]. Thus, in a DAG, a necessary condition for u to dominate v is $t(u) > t(v)$. Then, we have the following two lemmas.

Lemma 4.1. *Given any sink graph G, vertex $v \in V$ is dominated by vertex $u \in V$ iff all out-neighbors of v are dominated by u.*

Proof. The proof of the "if" part is trivial. We prove the "only if" part by contradiction. Suppose there exists a vertex $v \in V$ that is dominated by a vertex $u \in V$, but one of its out-neighbors $w \in N^+(v)$ is not dominated by u. Recall that we assumed in Section 3 that $r_G(x) \neq \emptyset$ for every $x \in V$. Then, there must be at least one path from w to $\perp$ that does not go through u. By prepending v to the beginning of this path, we obtain a path from v to $\perp$ that does not go through u; this contradicts that u dominates v. Thus, the lemma holds. □

Lemma 4.2. *Each* DOM*-equivalence class M is a maximal subset of V such that all vertices of M are dominated by the vertex in M with the largest topological number.*

Algorithm 2. Algorithm for DomCondense (Operator D)

```
Require: An acyclic sink graph $G = (V, S, E)$
Ensure: DOM-condensation of $G$
1: Initialize a disjoint-set data structure $\mathcal{D}$ for $V$
2: Compute a topological ordering of $V$
3: for vertex $u \in V$ in reverse topological ordering do
4:     if $N^+(u) \cap S = \emptyset$ and $N^+(u)$ belong to the same set in $\mathcal{D}$
       then
5:         Union $u$ and $N^+(u)$ in $\mathcal{D}$
6:     end if
7: end for
8: Condense $G$ based on the equivalence classes defined by $\mathcal{D}$
```

Proof. First, following from the definition of DOM-equivalence relation, all such vertices that are dominated by the same vertex must be in the same DOM-equivalence class. Second, we prove by contradiction that all vertices in a DOM-equivalence class M must be dominated by the vertex $u^* \in M$ that has the largest topological number. Suppose there are vertices in M not dominated by u^*. Let v be the vertex in M with the largest topological number that is not dominated by u^*, and M_v be the *maximal* subset of M (including v itself) that are dominated by v; note that $M \backslash M_v \neq \emptyset$. Then, vertices of $M \backslash M_v$ cannot be dominated by any vertex of M_v, as the dominance relationship is transitive. In addition, vertices of M_v cannot be dominated by any vertex of $M \backslash M_v$, as otherwise v would also be dominated by this vertex from $M \backslash M_v$ which in turn is dominated by u^* (note that, all vertices dominating v have larger topological numbers than v). Thus, M_v and $M \backslash M_v$ cannot be in the same DOM-equivalence class, and the lemma holds. □

Following from the above two lemmas, we propose to incrementally grow the equivalence classes by processing vertices according to the reverse of the topological ordering. When processing vertex u, all its out-neighbors must have already been processed; if all its out-neighbors are in the same equivalence class (indicating that u is dominated by a vertex in the equivalence class), then u itself should also belong to this equivalence class. The pseudocode of our DOM-condensation algorithm is shown in Algorithm 2, where we use a disjoint-set data structure $\mathcal{D}$ to incrementally grow the equivalence classes. We call Algorithm 2 the condensation operator D. Its correctness and time complexity are proved in the theorem below.

Theorem 2. *Given any sink graph G, Algorithm 2 correctly computes the* DOM*-condensation of G in $\mathcal{O}(|E|)$ time.*

Proof. The correctness directly follows from Lemmas 4.1 and 4.2. Regarding the time complexity, Line 2 can be computed in linear time [29], and the total time for conducting Lines 4–5 for all vertices is $\mathcal{O}(|E|\alpha(|E|))$ [29]. Here, $\alpha(\cdot)$ is a functional inverse of Ackermann's function [29], and is at most 4 for all practical input values; we consider $\alpha(\cdot)$ to be constant in this paper. Thus, the theorem holds. □

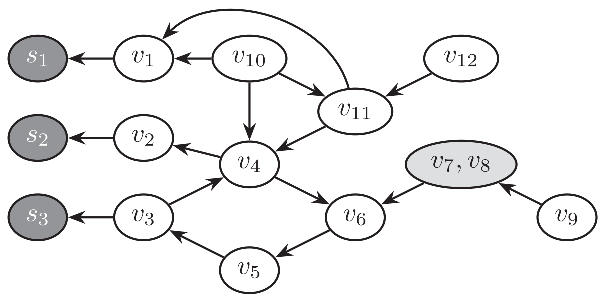


Fig. 5. IMOD-condensation of the graph in Fig. 1.

4.1.3 IMOD-Condensation and CMOD-Condensation

Our next two condensations are based on the concept of module [28], [33], which we adapt to sink graph as follows. Given a sink graph $G = (V, S, E)$, a vertex subset $M \subseteq V$ is a *module* if all vertices of M have the same external out-neighbors, i.e., $N^+(u) \backslash M = N^+(v) \backslash M, \forall u, v \in M$. It is easy to see that all vertices in the same module are equivalent in the kernel equivalence relation $\simeq_r$. However, there are two challenges to compute modules.

- The existing studies on modular decomposition require all vertices in a module to have *the same external in-neighbors* as well as the same external out-neighbors, while we only require the same external out-neighbors.
- The existing algorithms designed for modular decomposition on directed graphs are of theoretical interest only, where no implementation exists.

Thus, we define the following two special cases of module that we can compute efficiently.

Algorithm 3. Algorithm for iModCondense (Operator $\mathsf{M_i}$)

```
Require: A sink graph $G = (V, S, E)$
Ensure: IMOD-condensation of $G$
1: Initialize a partitioning $\mathcal{P} = \{V\}$
2: for vertex $u \in V \cup S$ do
3:     Refine $\mathcal{P}$ based on $N^-(u)$ (i.e., each partition $P \in \mathcal{P}$ is split
       into $P \cap N^-(u)$ and $P \backslash N^-(u)$)
4: end for
5: Condense $G$ based on the equivalence classes defined by $\mathcal{P}$
```

Definition 7. *Given a sink graph $G = (V, S, E)$, two vertices $u, v \in V$ are said to be* IMOD-equivalent, *denoted $u \simeq_{\text{IMOD}} v$, if $N^+(u) = N^+(v)$.*

Definition 8. *Given a sink graph $G = (V, S, E)$, two vertices $u, v \in V$ are said to be* CMOD-equivalent, *denoted $u \simeq_{\text{CMOD}} v$, if $(u, v) \in E$ and $N^+(u) \backslash \{v\} \subseteq N^+(v)$.*

The relation induced by IMOD-equivalence is reflexive, symmetric, and transitive. For CMOD-equivalence, we additionally enforce reflexivity, symmetry, and transitivity, similar to Section 4.1.2, to obtain a CMOD-equivalence relation. It is easy to see that $\simeq_{\text{IMOD}} \subseteq \simeq_r$ and $\simeq_{\text{CMOD}} \subseteq \simeq_r$. Thus, the condensation induced by the IMOD-equivalence relation, called IMOD-condensation, and the condensation induced by the CMOD-equivalence relation, called CMOD-condensation, are both sink-reachability preserving. For the sink graph in Fig. 1, its IMOD-condensation is shown in Fig. 5 and its CMOD-condensation is shown in Fig. 6. Note that, alternatively we can change the condition in Definition 8 to be "$(u, v) \in E$ and $N^+(u) \backslash \{v\} = N^+(v)$". However, it can be verified that Definition 8 defines a larger equivalence relation and thus is better for condensation.

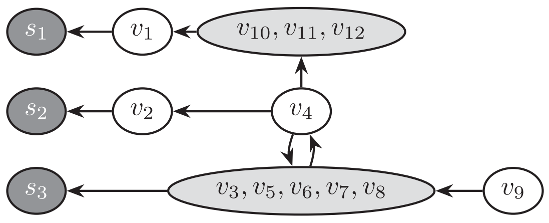


Fig. 6. CMOD-condensation of the graph in Fig. 1.

Algorithm 4. Algorithm for cModCondense (Operator $\mathsf{M_c}$)

```
Require: An acyclic sink graph $G = (V, S, E)$
Ensure: CMOD-condensation of $G$
1: Initialize a disjoint-set data structure $\mathcal{D}$ for $V$
2: Assign a topological number $t(u)$ to each vertex $u \in V \cup S$,
   and obtain a topological ordering of $V \cup S$
3: for vertex $u \in V$ do
4: $t^{min}(u) \leftarrow \min_{v \in N^+(u)} t(v)$
5: end for
6: for vertex $v \in V$ in reverse topological ordering do
7:     Mark $v$ and $N^+(v)$
8:     for in-neighbor $u \in N^-(v)$ of $v$ do
9:         if $t^{min}(u) = t(v)$ and $d^+(u) - 1 \leq d^+(v)$ then
10:            if all out-neighbors of $u$ are marked then
11:                Union $u$ and $v$ in $\mathcal{D}$
12:            end if
13:        end if
14:    end for
15:    Unmark $v$ and $N^+(v)$
16: end for
17: Condense $G$ based on the equivalence classes defined by $\mathcal{D}$
```

The pseudocode of conducting IMOD-condensation is shown in Algorithm 3, which is self-explanatory. We call Algorithm 3 the condensation operator $\mathsf{M_i}$. Its correctness and time complexity is proved in the theorem below.

Theorem 3. *Given any sink graph G, Algorithm 3 correctly computes the* IMOD*-condensation of G in $\mathcal{O}(|E|)$ time.*

Proof. First, it is easy to see that for any two IMOD-equivalent vertices v and w, they will never be split into different partitions at Line 3. Second, consider any two vertices v and w that are not IMOD-equivalent. Without loss of generality, assume there is a vertex $u \in N^+(v) \backslash N^+(w)$. Then, v and w must be in two different partitions after processing u for Line 3. Thus, Algorithm 3 correctly computes the IMOD-condensation. The linear time complexity of Algorithm 3 follows from the fact that each refinement at Line 3 can be conducted in $\mathcal{O}(d^-(u))$ time [1]. □

The pseudocode of conducting CMOD-condensation is shown in Algorithm 4, like Algorithm 2 it assumes that the sink graph is acyclic. The algorithm processes vertices in reverse topological ordering (Line 6). When processing vertex v, it checks for every in-neighbor $u \in N^-(v)$ whether $N^+(u) \backslash \{v\} \subseteq N^+(v)$: if the condition holds (Line 10), then u

and v are CMOD-equivalent (see Definition 8). Here, for time efficiency consideration, we first conduct a constant-time filtering at Line 9: $t^{min}(u) = t(v)$ and $d^+(u) - 1 \leq d^+(v)$ which are necessary conditions for $N^+(u)\backslash\{v\} \subseteq N^+(v)$ to hold. We call Algorithm 4 the condensation operator $\mathsf{M_c}$.

To prove the correctness of Algorithm 4, we first define the concept of subsume and subsume graph.

Definition 9. *Given an acyclic sink graph $G = (V, S, E)$, $v \in V$* subsumes *$u \in V$ if $(u, v) \in E$ and $N^+(u)\backslash\{v\} \subseteq N^+(v)$. The* subsume graph *of G consists of vertices V, and it has a directed edge from u to v if v subsumes u.*

Then, we show the connection between CMOD-equivalence classes and (weakly) connected components of the subsume graph in the lemma below.

Lemma 4.3. *The subsume graph consists of a set of rooted trees, where in each rooted tree, the edges point towards the root. Each CMOD-equivalence class consists of all vertices in a (weakly) connected component of the subsume graph.*

Proof. First, it is easy to verify that each vertex is subsumed by at most one of its out-neighbors. In particular, a vertex can only be possibly subsumed by the out-neighbor that has the smallest topological number. Thus, in the subsume graph, each vertex has at most one outgoing edge, and the subsume graph consists of a set of rooted trees where edges point towards root. □

Second, according to the definition of CMOD-equivalence relation, each CMOD-equivalence class consists of all vertices in a (weakly) connected component of the subsume graph.

Now, we are ready to prove the correctness and time complexity of Algorithm 4.

Theorem 4. *Given any sink graph G, Algorithm 4 correctly computes the CMOD-condensation of G in $\mathcal{O}(|E|)$ time.*

Proof. The correctness directly follows from Lemma 4.3. Regarding the time complexity, we know that first Lines 1–5 run in $\mathcal{O}(|E|)$ time. Second, running Line 9 for all vertices and their in-neighbors takes $\mathcal{O}(|E|)$ time in total. Third, Line 10 is tested for each vertex u at most once (due to the testing of $t^{min}(u) = t(v)$), and each testing takes time $\mathcal{O}(d^+(u))$. Thus, the total time complexity of $\mathcal{O}(|E|)$ follows. □

4.2 Composing Condensation Operators

In this subsection, we investigate how to compose the four condensation operators $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ proposed in Section 4.1. Note that, as each condensation operator maps sink graphs to sink graphs, condensation operators can be composed.

Definition 10. *Given two condensation operators c_1 and c_2, we define $c_1 \circ c_2(G)$, for any sink graph G, as $c_2(c_1(G))$ where $c_1(G)$ denotes the result of condensation of G by c_1.*

By definition, vertices of $c_1 \circ c_2(G)$ are nested sets of vertices from G, i.e., classes of classes. It is practical to flatten this structure by recursively aggregating the elements of the nested classes. Let $\simeq_1 \subseteq V \times V$ and $\simeq_2 \subseteq V_{\simeq_1} \times V_{\simeq_1}$ be the underlying equivalence relations of operator c_1 on G and operator c_2 on $c_1(G)$, respectively. We define the equivalence relation $\simeq_1\simeq_2 \subseteq V \times V$ to be: $u \simeq_1\simeq_2 v$ if and only if $[u]_{\simeq_1} \simeq_2 [v]_{\simeq_1}$. Then, the result of flattening $c_1 \circ c_2(G)$ is the same as the condensation of G induced by $\simeq_1\simeq_2$. That is, *a composition of condensation operators also has an underlying equivalence relation*. In the following, we consider only flat condensation, and use $c_1 \circ c_2$ to denote the flat condensation of composing c_1 and c_2.

We refer to a composition sequence of any positive number of condensation operators simply as a *condensation sequence*, and use $\mathbb{C}$ to denote a condensation sequence. Given any two condensation sequences $\mathbb{C}_1$ and $\mathbb{C}_2$, we say $\mathbb{C}_1(G)$ is the same as $\mathbb{C}_2(G)$ for a sink graph G, denoted $\mathbb{C}_1(G) = \mathbb{C}_2(G)$, if the underlying equivalence relation of $\mathbb{C}_1(G)$ is the same as that of $\mathbb{C}_2(G)$. We say $\mathbb{C}_1$ is the same as $\mathbb{C}_2$, denoted $\mathbb{C}_1 = \mathbb{C}_2$, if $\mathbb{C}_1(G) = \mathbb{C}_2(G)$ holds for every sink graph G. We define the maximality of condensation sequences as follows.

Definition 11. *Given a sink graph G and a set $\mathcal{C}$ of distinct condensation operators, a condensation sequence $\mathbb{C} \equiv c_1 \circ \cdots \circ c_n$, where $c_i \in \mathcal{C}$ for $1 \leq i \leq n$, is said to be* maximal *if $\mathbb{C} \circ c(G) = \mathbb{C}(G), \forall c \in \mathcal{C}$.*

Our main results of composing condensation operators are summarized in the theorem below.

Theorem 5. *Let $\mathcal{C}$ be $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.*

1) *Given any sink graph G, all maximal condensation sequences over $\mathcal{C}$ reduce G to the same sink graph.*
2) *Given any sink graph G, the shortest maximal condensation sequence over $\mathcal{C}$ can be approximated within a factor of 3.*
3) *There exist sink graphs G such that the reduced graph of G obtained by maximal condensation sequences over $\mathcal{C}$ is larger than the kernel condensation $G_{\simeq_r}$.*

In the following, we prove Theorems 5 (1), (2) and (3) in Sections 4.2.2, 4.2.3, and 4.2.4, respectively. Before that, we first make some observations on the properties of composing condensation operators $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ in Section 4.2.1.

4.2.1 Properties of Composing $\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}$

We first prove the following three lemmas which map edges in the condensation of G by $\{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ to paths of G.

Lemma 4.4. *Given any acyclic sink graph G and consider any edge $([u]_{\text{DOM}}, [v]_{\text{DOM}})$ in the DOM-condensation $\mathsf{D}(G)$ and any $u' \in [u]_{\text{DOM}}$, there exists a path in G from u' to a vertex of $[v]_{\text{DOM}}$ that goes through only vertices of $[u]_{\text{DOM}}$.*

Proof. Let u^* be the vertex with the largest topological number in $[u]_{\text{DOM}}$. Following Lemmas 4.1 and 4.2, we know that there exists a path in G from u' to u^* that goes through only vertices of $[u]_{\text{DOM}}$. Moreover, from the definition of condensation and the definition of dominance and Lemma 4.2, we know that there must be an edge (u^*, v') in G where $v' \in [v]_{\text{DOM}}$. Thus, the lemma holds. □

Lemma 4.5. *Given any acyclic sink graph G and consider any edge $([u]_{\text{IMOD}}, [v]_{\text{IMOD}})$ in the IMOD-condensation $\mathsf{M_i}(G)$ and any $u' \in [u]_{\text{IMOD}}$, there exists an edge in G from u' to a vertex of $[v]_{\text{IMOD}}$.*

Proof. Following the definition of IMOD-equivalence, we know that $N^+(u_1) = N^+(u_2)$ for any two vertices $u_1, u_2 \in$

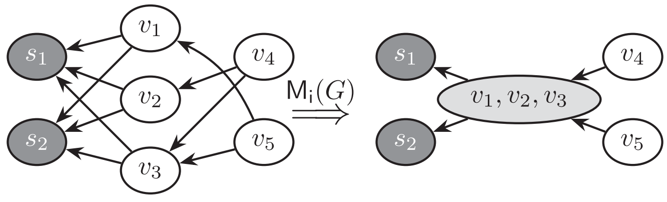


Fig. 7. $\mathsf{M_i}$ is not idempotent.

$[u]_{\text{IMOD}}$. Then, following the definition of condensation, we know that for any edge $([u]_{\text{IMOD}}, [v]_{\text{IMOD}})$ in $\mathsf{M_i}(G)$, there must exist a vertex $v' \in [v]_{\text{IMOD}}$, such that (u'', v') is an edge in G for all $u'' \in [u]_{\text{IMOD}}$. Thus, the lemma holds. □

Lemma 4.6. *Given any acyclic sink graph G and consider any edge $([u]_{\text{CMOD}}, [v]_{\text{CMOD}})$ in the* CMOD*-condensation $\mathsf{M_c}(G)$ and any $u' \in [u]_{\text{CMOD}}$, there exists a path in G from u' to a vertex of $[v]_{\text{CMOD}}$ that goes through only verties of $[u]_{\text{CMOD}}$.*

Proof. Consider any CMOD-equivalence class M, let u^* be the vertex with the largest topological number in M. Then, for each vertex $u \in M \backslash \{u^*\}$, following Lemma 4.3 we know that (1) there is a path from u to u^* that goes through only vertices of M, and (2) $N^+(u) \backslash M \subseteq N^+(u^*)$. Thus, the lemma can be proved by following the same argument as in proving Lemma 4.4. □

The properties of composing condensation operators $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ are summarized in the theorem below.

Theorem 6. *We note the following properties of composing condensation operators $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.*

1) S *is idempotent, i.e.,* $\mathsf{S} \circ \mathsf{S} = \mathsf{S}$.
2) D *is idempotent, i.e.,* $\mathsf{D} \circ \mathsf{D} = \mathsf{D}$.
3) $\mathsf{M_i}$ *is not idempotent, i.e., there exist sink graphs G such that* $\mathsf{M_i} \circ \mathsf{M_i}(G) \neq \mathsf{M_i}(G)$.
4) $\mathsf{M_c}$ *is not idempotent, i.e., there exist sink graphs G such that* $\mathsf{M_c} \circ \mathsf{M_c}(G) \neq \mathsf{M_c}(G)$.
5) *Given any acyclic sink graph G,* $\mathsf{D}(G)$, $\mathsf{M_i}(G)$ *and* $\mathsf{M_c}(G)$ *are all acyclic.*

Proof. The five properties are proved one-by-one in the following. □

Proof of Property (1). According to the definition, an SCC is a maximal set of vertices such that every pair of its vertices can reach each other. Thus, after contracting each SCC into a super-vertex, the resulting graph is acyclic. As a result, $\mathsf{S} \circ \mathsf{S}(G) = \mathsf{S}(G)$ holds for every sink graph G.

Proof of Property (2). Recall that operator D takes an acyclic sink graph as input (see Algorithm 2). We prove this property by contradiction. Suppose there exists an acyclic sink graph G such that $\mathsf{D} \circ \mathsf{D}(G) \neq \mathsf{D}(G)$; that is, in $\mathsf{D}(G)$, there exists a vertex $[u]_{\text{DOM}}$ that dominates another vertex $[v]_{\text{DOM}} \neq [u]_{\text{DOM}}$. Without loss of generality, assume that there is no other vertex $[w]_{\text{DOM}}$ such that $[w]_{\text{DOM}}$ also dominates $[v]_{\text{DOM}}$ and is dominated by $[u]_{\text{DOM}}$. Then, $[v]_{\text{DOM}}$ has only one out-neighbor in $\mathsf{D}(G)$, which is $[u]_{\text{DOM}}$. Following Lemma 4.2 and the definition of condensation, all vertices of $[v]_{\text{DOM}}$ must be dominated in G by the vertex u^* with the largest topological number in $[u]_{\text{DOM}}$. This contradicts that $[v]_{\text{DOM}} \neq [u]_{\text{DOM}}$.

Proof of Property (3). This can be easily seen by using the example depicted in Fig. 7. Applying operator $\mathsf{M_i}$ yields only one non-trivial IMOD-equivalence class $\{v_1, v_2, v_3\}$. Once this equivalence class is contracted, another application of $\mathsf{M_i}$ on the resulting sink graph finds another IMOD-equivalence class $\{v_4, v_5\}$.

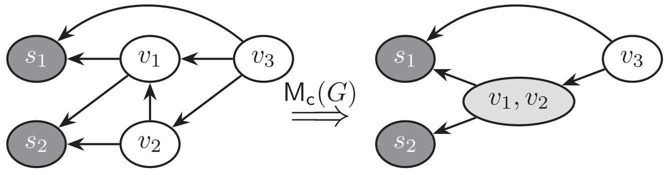


Fig. 8. $\mathsf{M_c}$ is not idempotent.

Proof of Property (4). This can also be easily seen by using the example depicted in Fig. 8. Applying operator $\mathsf{M_c}$ yields only one non-trivial CMOD-equivalence class $\{v_1, v_2\}$. Once this equivalence class is contracted, another application of $\mathsf{M_c}$ on the resulting sink graph finds another CMOD-equivalence class $\{\{v_1, v_2\}, v_3\}$.

Proof of Property (5). This can be proved by contradiction and using the results of Lemmas 4.4, 4.5 and 4.6. That is, suppose there is a cycle in $\mathsf{D}(G)$, $\mathsf{M_i}(G)$ or $\mathsf{M_c}(G)$, then we can also find a cycle in G which contradicts that G is acyclic.

It is worth mentioning that Figs. 7 and 8 also demonstrate that the operators D, $\mathsf{M_i}$, $\mathsf{M_c}$ complement each other. For example, there are no non-trivial DOM-equivalence classes in these two graphs, there are no non-trivial CMOD-equivalence classes in Fig. 7, and there are no non-trivial IMOD-equivalence classes in Fig. 8.

4.2.2 Fixpoint of Maximal Condensation Sequences

We now consider the question of whether, for any sink graph G, the applications of different condensation sequences reach the same fixpoint. Following Theorem 6, we do not need to consider all sequences in $(\mathsf{S}|\mathsf{D}|\mathsf{M_i}|\mathsf{M_c})^*$ since some sequences are known to yield the same result for an arbitrary sink graph. In particular, we first can restrict our study to sequences where S is always followed by D, or $\mathsf{M_i}$, or $\mathsf{M_c}$, as $\mathsf{S} \circ \mathsf{S} = \mathsf{S}$ (see Theorem 6 (1)). Moreover, once operator S has been applied, no further application of S will be needed any more; this is because Theorem 6 (5) shows that operators $\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}$ preserve the acyclicity of sink graphs. As a result, *we assume an acyclic input sink graph and consider only operators* $\mathsf{D}, \mathsf{M_i}$ *and* $\mathsf{M_c}$ *in the following.*

Obviously, two non-maximal condensation sequences may reduce a sink graph into different sizes. On the other hand, since condensation reduces the graph size and graphs have a trivial lower bound size of 0, every condensation sequence can be extended to a maximal one. Then, the question arises as to whether all maximal condensation sequences produce the same result for an arbitrary sink graph, i.e., whether the applications of all maximal condensation sequences reach a unique fixpoint. To answer this question, we define aggregation between two condensation sequences, and define the monotonicity of a condensation operator as follows.

Definition 12. *Given a sink graph $G = (V, S, E)$ and two condensation sequences $\mathbb{C}_1$ and $\mathbb{C}_2$, let $\simeq_1 \subseteq V \times V$ and $\simeq_2 \subseteq V \times V$ be their underlying equivalence relations, then $\mathbb{C}_2(G)$ is said to be an aggregation of $\mathbb{C}_1(G)$, denoted $\mathbb{C}_1(G) \succeq \mathbb{C}_2(G)$, if $[u]_{\simeq_1} \subseteq [u]_{\simeq_2}, \forall u \in V$. That is, $\mathbb{C}_2(G)$ can be considered as a further condensation of $\mathbb{C}_1(G)$.*

Definition 13. *A condensation operator c is said to be monotone regarding a set $\mathcal{C}$ of condensation operators if for any acyclic sink graph G and any two condensation sequences $\mathbb{C}_1$ and $\mathbb{C}_2$ over $\mathcal{C}$, $\mathbb{C}_1(G) \succeq \mathbb{C}_2(G)$ implies that $\mathbb{C}_1 \circ c(G) \succeq \mathbb{C}_2 \circ c(G)$.*

We prove in the theorem below that, the applications of all maximal condensation sequences over a set $\mathcal{C}$ of monotone condensation operators reach a unique fixpoint.

Theorem 7. *Given any set $\mathcal{C}$ of monotone condensation operators and any sink graph G, all maximal condensation sequences over $\mathcal{C}$ reduce G to the same sink graph.*

Proof. Let I be the identity condensation operator, i.e., $I(G) = G$ holds for every sink graph G. For a given sink graph G, let $\mathbb{C}_1 \equiv c_1^1 \circ \cdots \circ c_n^1$ and $\mathbb{C}_2 \equiv c_1^2 \circ \cdots \circ c_m^2$ be any two maximal condensation sequences over $\mathcal{C}$. Then, trivially we have $I(G) \succeq \mathbb{C}_1(G)$. By appending $\mathbb{C}_2$ to both sides, we have $\mathbb{C}_2(G) = I \circ \mathbb{C}_2(G) \succeq \mathbb{C}_1 \circ \mathbb{C}_2(G) = \mathbb{C}_1(G)$, as all condensation operators of $\mathcal{C}$ are monotone; the second equality follows from the fact that $\mathbb{C}_1$ is maximal for G. Similarly, we also have $\mathbb{C}_1(G) = I \circ \mathbb{C}_1(G) \succeq \mathbb{C}_2 \circ \mathbb{C}_1(G) = \mathbb{C}_2(G)$. Thus, $\mathbb{C}_1(G) = \mathbb{C}_2(G)$ according to Definition 12, and the theorem holds. □

In the following, we prove that all our operators $\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}$ are monotone regarding $\mathcal{C} = \{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.

Operator D *is Monotone*. We prove this by contradiction. Suppose there is an acyclic sink graph $G = (V, S, E)$ and two condensation sequences $\mathbb{C}_1$ and $\mathbb{C}_2$ such that $\mathbb{C}_1(G) \succeq \mathbb{C}_2(G)$ and $\mathbb{C}_1 \circ \mathsf{D}(G) \not\succeq \mathbb{C}_2 \circ \mathsf{D}(G)$. Let $\simeq_1, \simeq_2, \simeq_3, \simeq_4$ be the underlying equivalence relations of $\mathbb{C}_1(G), \mathbb{C}_2(G), \mathbb{C}_1 \circ \mathsf{D}(G), \mathbb{C}_2 \circ \mathsf{D}(G)$, respectively. Then, there must exist two vertices $u, v \in V$ such that $[u]_{\simeq_1} \neq [v]_{\simeq_1}$, $u \simeq_3 v$ (which is the result of $[u]_{\simeq_1}$ dominating $[v]_{\simeq_1}$ in $\mathbb{C}_1(G)$) and $u \not\simeq_4 v$. Consequently, $u \not\simeq_2 v$ and $u \not\simeq_1 v$. As $u \not\simeq_4 v$, $[u]_{\simeq_2}$ does not dominate $[v]_{\simeq_2}$ in $\mathbb{C}_2(G)$; that is, there is a path from $[v]_{\simeq_2}$ to $\bot$ in $\mathbb{C}_2(G)$ that does not go through $[u]_{\simeq_2}$. Following Lemmas 4.4, 4.5 and 4.6, there is a path from v to $\bot$ in G without going through any vertices of $[u]_{\simeq_2}$. As $\mathbb{C}_1(G) \succeq \mathbb{C}_2(G)$, we have $[u]_{\simeq_1} \subseteq [u]_{\simeq_2}$. Thus, there is a path from $[v]_{\simeq_1}$ to $\bot$ in $\mathbb{C}_1(G)$ without going through $[u]_{\simeq_1}$. This contradicts that $[u]_{\simeq_1}$ dominates $[v]_{\simeq_1}$ in $\mathbb{C}_1(G)$. Thus, operator D is monotone regarding $\{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.

Operator $\mathsf{M_i}$ *is Monotone*. We prove this by contradiction. Suppose there is an acyclic sink graph $G = (V, S, E)$ and two condensation sequences $\mathbb{C}_1$ and $\mathbb{C}_2$ such that $\mathbb{C}_1(G) \succeq \mathbb{C}_2(G)$ and $\mathbb{C}_1 \circ \mathsf{M_i}(G) \not\succeq \mathbb{C}_2 \circ \mathsf{M_i}(G)$. Let $\simeq_1, \simeq_2, \simeq_3, \simeq_4$ be the underlying equivalence relations of $\mathbb{C}_1(G), \mathbb{C}_2(G), \mathbb{C}_1 \circ \mathsf{M_i}(G), \mathbb{C}_2 \circ \mathsf{M_i}(G)$, respectively. Then, there must exist two vertices $u, v \in V$ such that $u \simeq_3 v$ and $u \not\simeq_4 v$. Consequently, $u \not\simeq_2 v$ and $u \not\simeq_1 v$, and $N^+_{\mathbb{C}_1(G)}([u]_{\simeq_1}) = N^+_{\mathbb{C}_1(G)}([v]_{\simeq_1})$. Here, $N^+_{\mathbb{C}_1(G)}([u]_{\simeq_1})$ denotes the set of out-neighbors of $[u]_{\simeq_1}$ in $\mathbb{C}_1(G)$. As $u \not\simeq_4 v$, we have $N^+_{\mathbb{C}_2(G)}([u]_{\simeq_2}) \neq N^+_{\mathbb{C}_2(G)}([v]_{\simeq_2})$; without loss of generality, we can safely assume that $([v]_{\simeq_2}, [u]_{\simeq_2})$ is not an edge of $\mathbb{C}_2(G)$ and there is a $[w]_{\simeq_2} \in N^+_{\mathbb{C}_2(G)}([u]_{\simeq_2}) \backslash N^+_{\mathbb{C}_2(G)}([v]_{\simeq_2})$ (note that, $[w]_{\simeq_2}$ can be the same as $[v]_{\simeq_2}$). Following Lemmas 4.4, 4.5 and 4.6, for any $u' \in [u]_{\simeq_1} \subseteq [u]_{\simeq_2}$, there is a vertex $w' \in [w]_{\simeq_2}$ such that there is a path from u' to w' that does not go though any vertices outside $[u]_{\simeq_2}$. Thus, there must be a $[x]_{\simeq_1} \subset [u]_{\simeq_2} \cup [w]_{\simeq_2}$ such that $([u]_{\simeq_1}, [x]_{\simeq_1})$ is an edge of $\mathbb{C}_1(G)$ (i.e., there is an edge in G from a vertex of $[u]_{\simeq_1}$ to a vertex of $[x]_{\simeq_1}$) and $([v]_{\simeq_1}, [x]_{\simeq_1})$ is not an edge of $\mathbb{C}_1(G)$ (i.e., there is no edge in G from any vertex of $[v]_{\simeq_1}$ to any vertex of $[x]_{\simeq_1}$). This contradicts that $N^+_{\mathbb{C}_1(G)}([u]_{\simeq_1}) = N^+_{\mathbb{C}_1(G)}([v]_{\simeq_1})$. Thus, operator $\mathsf{M_i}$ is monotone regarding $\{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.

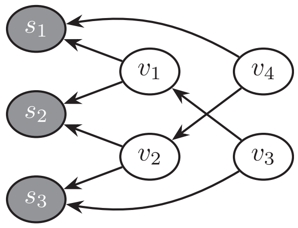


Fig. 9. An example sink graph G where $G \neq G_{\simeq_r}$ but G cannot be reduced by $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.

Operator $\mathsf{M_c}$ *is Monotone*. We prove this by contradiction. Suppose there is an acyclic sink graph $G = (V, S, E)$ and two condensation sequences $\mathbb{C}_1$ and $\mathbb{C}_2$ such that $\mathbb{C}_1(G) \succeq \mathbb{C}_2(G)$ and $\mathbb{C}_1 \circ \mathsf{M_c}(G) \not\succeq \mathbb{C}_2 \circ \mathsf{M_c}(G)$. Let $\simeq_1, \simeq_2, \simeq_3, \simeq_4$ be the underlying equivalence relations of $\mathbb{C}_1(G), \mathbb{C}_2(G), \mathbb{C}_1 \circ \mathsf{M_c}(G), \mathbb{C}_2 \circ \mathsf{M_c}(G)$, respectively. Then, there must exist two vertices $u, v \in V$ such that $[u]_{\simeq_1} \neq [v]_{\simeq_1}$, $u \simeq_3 v$ (which is the result of $[u]_{\simeq_1}$ subsuming $[v]_{\simeq_1}$ in $\mathbb{C}_1(G)$), and $u \not\simeq_4 v$. Consequently, $u \not\simeq_2 v$ and $u \not\simeq_1 v$. As $u \not\simeq_4 v$, $[u]_{\simeq_2}$ does not subsume $[v]_{\simeq_2}$ in $\mathbb{C}_2(G)$; that is, there is a vertex $[w]_{\simeq_2} \in N^+_{\mathbb{C}_2(G)}([v]_{\simeq_2}) \backslash (N^+_{\mathbb{C}_2(G)}([u]_{\simeq_2}) \cup \{[u]_{\simeq_2}\})$ (note that, $([v]_{\simeq_2}, [u]_{\simeq_2})$ must be an edge in $\mathbb{C}_2(G)$). Following Lemmas 4.4, 4.5 and 4.6, for any $v' \in [v]_{\simeq_2}$, there is a vertex $w' \in [w]_{\simeq_2}$ such that there is path from v' to w' that does not go though any vertex outside $[v]_{\simeq_2}$. Thus, there must be a $[x]_{\simeq_1} \subset [v]_{\simeq_2} \cup [w]_{\simeq_2}$ such that $([v]_{\simeq_1}, [x]_{\simeq_1})$ is an edge of $\mathbb{C}_1(G)$ and $([u]_{\simeq_1}, [x]_{\simeq_1})$ is not an edge of $\mathbb{C}_1(G)$. This contradicts that $N^+_{\mathbb{C}_1(G)}([v]_{\simeq_1}) \backslash (N^+_{\mathbb{C}_1(G)}([u]_{\simeq_1}) \cup [u]_{\simeq_1}) = \emptyset$. Consequently, operator $\mathsf{M_c}$ is monotone regarding $\{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.

4.2.3 Approximate Shortest Reduction Sequence

From Section 4.2.2, we call maximal condensation sequences *reduction sequences*. As each condensation operator runs in linear time, the running time of a reduction sequence grows linearly with its length (i.e., the number of condensation operators in the sequence). Thus, for any given sink graph G, it is ideal to have a shortest reduction sequence. In this paper, we propose a simple reduction sequence in the theorem below which is a 3-approximation to the shortest reduction sequence, and leave finding shorter reduction sequences as future work.

Theorem 8. *Given any acyclic sink graph G, let C be the composition of any permutation of $\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}$, and let C^* be the sequence of repeatedly applying C until convergence, then the length of C^* is at most three times the length of the shortest reduction sequence.*

Proof. Without loss of generality, assume $\mathsf{C} \equiv \mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c}$. Consider a shortest reduction sequence $\mathbb{C}$ of G, let n be the length of $\mathbb{C}$ (i.e., $n = |\mathbb{C}|$) and $\mathbb{C}^\Delta$ be the sequence obtained by augmenting $\mathbb{C}$ as follows: for each condensation operator $c \in \mathbb{C}$, replace it with C. We prove by induction that $\mathbb{C}^\Delta$ is maximal; thus $|C^*| \leq |\mathbb{C}^\Delta| = 3n$. Denote $\mathbb{C}_i$ the prefix of $\mathbb{C}$ with exactly i condensation operators, and $\mathbb{C}_i^\Delta$ the result of augmenting $\mathbb{C}_i$. Initially, $\mathbb{C}_0(G) = \mathbb{C}_0^\Delta(G)$ trivially holds. Assume that $\mathbb{C}_{i-1}(G) \succeq \mathbb{C}_{i-1}^\Delta(G)$ holds.

TABLE 2
Statistics of Sink Graphs $G = (V, S, E)$, Where $|V_r|$ Represents the Vertex Count in the Kernel Condensation, $|V^c|$ and $|E^c|$ Represent the Vertex Count and Edge count in the Core Graph (i.e., After Removing All Vertices That Cannot Reach Any Sink Vertices)

Graphs	$\|V\|$	$\|S\|$	$\|E\|$	DAG?	$\|V^c\|$	$\|E^c\|$	$\|V_r\|$
avrora-s	562615	68971	708614	no	248066	483220	10340
batik-s	684010	78921	864264	no	290420	571600	12560
eclipse-s	324226	41516	414638	no	146513	289207	6183
fop-s	765724	95327	989696	no	339797	674585	13524
h2-s	590845	70293	758421	no	259960	514197	10402
jenkins-s	3196349	404514	4103165	no	1398361	2750669	41580
jython-s	867730	88565	1155836	no	352899	675566	12356
luindex-s	317197	41454	401044	no	143613	282322	5996
lusearch-s	317510	41503	401416	no	143639	282425	5997
openjdk8-s	1573653	215902	1978755	no	694939	1346830	22324
pmd-s	568521	67999	725670	no	247054	482285	10258
sunflow-s	522667	64464	664208	no	230685	450004	9880
tomcat-s	303510	40426	390167	no	141100	277496	5938
tradebean-s	272083	36272	346088	no	125810	246038	5356
xalan-s	673681	99952	841457	no	311060	601623	10908
avrora-l	562615	68971	2725835	no	443283	2369919	11794
batik-l	684010	78921	3336522	no	533147	2910704	14339
eclipse-l	324226	41516	1421235	no	248581	1233657	7160
fop-l	765724	95327	4063315	no	603459	3573151	15449
h2-l	590845	70293	2865741	no	460667	2480215	11964
jenkins-l	3196349	404514	59821383	no	2466802	53310172	48198
jython-l	867730	88565	26716793	no	701498	25943050	13676
luindex-l	317197	41454	1401880	no	245739	1222298	7178
lusearch-l	317510	41503	1402425	no	245938	1222868	7180
openjdk8-l	1573653	215902	12387187	no	1257860	11035312	26755
pmd-l	568521	67999	3068722	no	441828	2657647	11726
sunflow-l	522667	64464	2482443	no	409761	2152612	11171
tomcat-l	303510	40426	1327010	no	233842	1152127	6739
tradebean-l	272083	36272	1139807	no	209582	993454	6224
xalan-l	673681	99952	2997836	no	520315	2593130	12530
LiveJournal1-8	893533	77699	1017575	yes	366304	522267	7390
soc-Pokec-8	299821	26071	377788	yes	112609	168423	2825
web-BerkStan-8	100654	8752	574228	yes	81157	553039	209
web-Google-8	342023	29741	508426	yes	156756	263057	3536

We have $\mathbb{C}_i(G) = \mathbb{C}_{i-1} \circ \{\mathsf{D}|\mathsf{M_i}|\mathsf{M_c}\}(G) \succeq \mathbb{C}^{\Delta}_{i-1} \circ \{\mathsf{D}|\mathsf{M_i}|\mathsf{M_c}\}(G) \succeq \mathbb{C}^{\Delta}_{i-1} \circ \mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c}(G) = \mathbb{C}^{\Delta}_{i}(G)$, as operators $\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}$ are monotone regarding $\{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$. Thus, $\mathbb{C}_n(G) \succeq \mathbb{C}^{\Delta}_n(G)$. Moreover, from Section 4.2.2, we know that $\mathbb{C}^{\Delta}_n(G) \succeq \mathbb{C}_n(G)$. Thus, the theorem holds. □

4.2.4 Gap to the Kernel Condensation

We know from Section 4.2.2 that, (1) given any sink graph G, the result of $\mathsf{S}(G)$ is acyclic, and (2) given any acyclic sink graph G', all maximal condensation sequences over $\{\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ reduce G' to the same sink graph. It is natural to ask whether the reduced graph at the fixpoint is the same as the kernel condensation $G_{\simeq_r}$. We show by the following theorem that the answer is "no".

Theorem 9. *There are sink graphs G such that all condensation sequences over condensation operators $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ yield a reduced sink graph that is larger than $G_{\simeq_r}$.*

Proof. This can be easily seen by using the example sink graph G depicted in Fig. 9. It can be verified that G cannot be reduced by any of the condensation operators $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$. However, $r_G(v_3) = r_G(v_4) = \{s_1, s_2, s_3\}$, i.e., $v_3 \simeq_r v_4$. □

We leave the study of efficiently computing the kernel condensation $G_{\simeq_r}$ to future work. Nevertheless, in practice maximal condensation sequences over $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ compress sink graphs close to their kernel condensations, as we will show empirically in Section 5.

5 Experiments

In this section, we conduct extensive empirical studies on real sink graphs aiming to answer the following questions.

1) *How effective are the individual condensation operators* S, D, $\mathsf{M_i}$, $\mathsf{M_c}$ *for sink-reachability preserving reduction?* (Eval-I in Section 5.1)
2) *How effective is our compositional approach to sink-reachability preserving reduction?* (Eval-II)
3) *How long are our compression sequences and how far are they from the shortest one?* (Eval-III)
4) *Which reduction sequence to use in practice?* (Eval-IV)
5) *Is our reduction useful for computing reachability indexes?* (Eval-V)

Datasets. As discussed in Section 1, static program analysis naturally models programs to be analysed as sink graphs. We downloaded 15 programs for our testings: 13 mid-sized Java programs from the Dacapo benchmark [34],[3] openjdk8 from the Java standard class library, and jenkins from a popular open-source web application for continuous integration. For each program, we use the doop processing pipeline [35] to produce datalog facts, and then build sink graphs from the records that represent assignments and allocations. This resulted in 15 sink graphs shown in Table 2

3. The Dacapo benchmark consists of 14 programs. We removed one program, *tradesoap*, as it is almost identical to another program in the benchmark, *tradebean*.

TABLE 3
Percentage ($\%$) for Vertex ($|V|$) and Edge ($|E|$) Counts of the Compressed Graphs to the Corresponding Core Graphs

Graphs	$\frac{\lvert V_{\mathsf{S}}\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_{\mathsf{S}}\rvert}{\lvert E^c\rvert}$	$\frac{\lvert V_{\mathsf{S}\circ\mathsf{D}}\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_{\mathsf{S}\circ\mathsf{D}}\rvert}{\lvert E^c\rvert}$	$\frac{\lvert V_{\mathsf{S}\circ\mathsf{M_i}}\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_{\mathsf{S}\circ\mathsf{M_i}}\rvert}{\lvert E^c\rvert}$	$\frac{\lvert V_{\mathsf{S}\circ\mathsf{M_c}}\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_{\mathsf{S}\circ\mathsf{M_c}}\rvert}{\lvert E^c\rvert}$	$\frac{\lvert V_r\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_r\rvert}{\lvert E^c\rvert}$	$\frac{\lvert V_{fp}\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_{fp}\rvert}{\lvert E^c\rvert}$	$\frac{\lvert V_{ter}\rvert}{\lvert V^c\rvert}$	$\frac{\lvert E_{ter}\rvert}{\lvert E^c\rvert}$
avrora-s	94.15	93.70	6.41	39.10	61.94	53.83	7.47	25.56	4.17	21.44	4.40	21.75	73.01	59.32
batik-s	94.64	94.01	6.91	39.74	61.92	53.82	7.96	26.40	4.32	21.79	4.56	22.09	73.59	59.63
eclipse-s	93.92	93.32	6.81	39.56	61.50	53.30	7.96	25.92	4.22	21.41	4.44	21.66	72.81	58.61
fop-s	94.04	93.67	6.15	39.86	60.75	53.60	7.17	26.42	3.98	22.54	4.16	22.78	71.55	58.89
h2-s	93.30	92.75	6.54	38.45	59.70	51.62	7.67	25.21	4.00	20.67	4.24	20.97	70.92	57.22
jenkins-s	93.68	93.17	4.91	37.16	55.95	48.83	5.71	22.93	2.97	19.13	3.11	19.31	66.80	53.53
jython-s	94.64	94.08	5.65	36.36	57.08	50.41	6.61	23.88	3.50	19.87	3.69	20.10	67.73	55.86
luindex-s	93.51	92.93	6.57	39.47	62.17	53.53	7.64	25.45	4.18	21.23	4.39	21.47	72.99	58.82
lusearch-s	93.50	92.92	6.56	39.49	62.17	53.52	7.63	25.44	4.18	21.24	4.39	21.48	72.98	58.81
openjdk8-s	95.09	94.61	5.33	38.30	58.26	49.73	6.27	22.94	3.21	19.00	3.37	19.20	68.24	53.86
pmd-s	93.86	93.36	6.52	38.96	61.40	53.40	7.58	25.52	4.15	21.28	4.39	21.58	72.59	58.94
sunflow-s	94.41	93.92	6.69	39.38	61.66	53.60	7.76	25.75	4.28	21.48	4.53	21.78	72.80	59.08
tomcat-s	93.76	93.17	6.69	39.47	61.67	53.29	7.78	25.56	4.21	21.24	4.44	21.50	72.87	58.76
tradebean-s	93.99	93.36	6.80	39.62	62.12	53.54	7.85	25.48	4.26	21.06	4.49	21.33	73.23	59.00
xalan-s	94.34	93.86	5.64	38.84	60.25	50.67	6.54	22.82	3.51	19.02	3.71	19.26	70.00	55.56
avrora-l	87.54	66.31	12.76	34.91	50.82	19.99	11.75	15.77	2.66	6.28	3.07	6.88	66.29	21.90
batik-l	87.15	64.90	13.86	35.58	49.88	20.00	12.81	16.71	2.69	6.35	3.13	7.03	65.46	21.10
eclipse-l	87.11	69.13	13.66	37.88	51.00	22.08	12.68	17.93	2.88	7.13	3.34	7.77	66.15	25.83
fop-l	86.87	62.92	13.07	34.42	49.72	19.13	12.14	16.10	2.56	6.38	2.98	6.96	64.89	20.24
h2-l	86.64	66.09	13.34	34.58	48.97	19.78	12.37	16.32	2.60	6.34	3.03	6.93	64.34	21.63
jenkins-l	85.12	50.55	11.11	31.39	46.27	6.89	10.00	6.77	1.95	1.81	2.25	2.11	60.63	5.45
jython-l	74.80	10.56	10.56	5.84	40.68	2.47	9.55	2.17	1.95	0.76	2.25	0.84	53.32	2.72
luindex-l	87.45	70.72	14.18	39.31	51.55	22.32	13.16	18.84	2.92	7.28	3.40	7.98	66.72	26.56
lusearch-l	87.46	70.74	14.19	39.32	51.56	22.33	13.17	18.86	2.92	7.28	3.40	7.98	66.74	26.59
openjdk8-l	87.51	56.15	12.06	30.81	48.23	13.04	11.47	11.56	2.13	3.90	2.47	4.38	63.08	13.11
pmd-l	85.83	63.63	13.31	34.81	49.50	18.12	12.09	15.03	2.65	5.80	3.06	6.34	64.57	19.84
sunflow-l	87.04	65.68	13.28	35.32	50.07	20.43	12.20	16.74	2.73	6.57	3.16	7.18	65.20	22.37
tomcat-l	86.68	67.35	13.51	36.97	51.34	21.89	12.04	16.74	2.88	7.10	3.33	7.69	66.08	24.20
tradebean-l	87.37	70.46	14.17	38.79	51.68	22.89	13.11	18.60	2.97	7.26	3.45	7.94	66.69	27.23
xalan-l	87.66	66.83	12.38	34.90	50.19	20.95	11.33	16.71	2.41	6.45	2.81	7.06	64.20	22.56
LiveJournal1-8	100.00	100.00	2.39	30.59	5.14	18.49	3.12	15.95	2.02	15.13	2.02	15.13	7.16	18.68
soc-Pokec-8	100.00	100.00	3.35	33.36	6.09	21.29	4.68	18.31	2.51	16.73	2.52	16.74	8.25	20.46
web-BerkStan-8	100.00	100.00	2.42	2.99	6.54	3.10	25.97	7.33	0.26	0.54	0.26	0.54	11.77	3.14
web-Google-8	100.00	100.00	3.34	21.96	15.36	25.93	16.55	19.88	2.26	9.13	2.27	9.15	25.31	25.70

$|V_r|$ and $|E_r|$ represent the size of the kernel condensation. $|V_{fp}|$ and $|E_{fp}|$ represent the size of the fixpoint condensation by $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$. $|V_{ter}|$ and $|E_{ter}|$ represent the size of the graph obtained by the techniques in [1]. Note that for the vertex count, only non-sink vertices are taken into account.

with suffix “-s” in the name. These graphs provide an unsound model for program analysis as methods are not devirtualised and fields are not modelled; unsound here means that the model does not represent the complete runtime behaviour of the program being analysed.

To build a sound model, we then added additional edges to model flow through fields for matching store-load records by following the process described in [14], and added additional assign edges modelling inter-procedural flow created by devirtualisation. The result of adding those two sets of additional edges is the 15 denser sink graphs shown in Table 2 with suffix “-l” in the name.

In addition, we also downloaded four social network and web graphs from SNAP:[4] LiveJournal1, soc-Pokec, web-BerkStan, and web-Google. As these graphs are not sink graphs, we convert them into sink graphs as follows: we first contract each SCC into a super-vertex, then compute a topological ordering for the resulting DAG, and finally assign the last $x\%$ percent of vertices as sink vertices. We choose x from $\{2, 4, 6, 8, 10, 20, 40\}$, and set $x = 8$ by default which are shown in Table 2 with suffix “-8” in the name.

Statistics of these sink graphs are summarized in Table 2, where $|V|$ represents the number of non-sink vertices, $|S|$ represents the number of sink vertices, and $|E|$ represents the total number of edges. It is as expected that the graphs extracted with the second method (named -d) are much denser than the graphs extracted with the first method (named -s). The sink graphs obtained from programs contain cycles, while the sink graphs constructed from social networks and web graphs are acyclic, as illustrated in the fifth column. For each sink graph G, we also obtain its *core graph*, denoted G^c, which is the result of removing all non-sink vertices that cannot reach any sink vertices. We apply our reduction techniques on the core graph, as all non-sink vertices that cannot reach any sink vertices can be easily identified and removed in linear time in a preprocessing step using a simple traversal starting from the sink vertices. We denote the number of non-sink vertices and the number of edges in the core graph by $|V^c|$ and $|E^c|$, respectively, those are shown in the sixth and seventh columns of Table 2; note that, the number of sink vertices still remains $|S|$. In Table 2, we also show the number $|V_r|$ of non-sink vertices in the kernel condensation in the last column, which will be used later to assess the quality of different reduction techniques.

4. http://snap.stanford.edu/data/index.html.

TABLE 4
Lengths of Compression Sequences ($\mathbb{C}_{opt}$ is the Shortest Compression Sequence, $\mathbb{C}_{(\mathsf{DM_iM_c})^*}$ is the Compression Sequence $\mathsf{S} \circ (\mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c})^*$, $\mathbb{C}_{\mathsf{D}(\mathsf{M_iM_c})^*}$ is the Compression Sequence $\mathsf{S} \circ \mathsf{D} \circ (\mathsf{M_i} \circ \mathsf{M_c})^*$, and $\mathbb{C}_{(\mathsf{M_iM_c})^*}$ is the Compression Sequence $\mathsf{S} \circ (\mathsf{M_i} \circ \mathsf{M_c})^*$)

Graphs	$\lvert\mathbb{C}_{opt}\rvert$	$\lvert\mathbb{C}_{(\mathsf{DM_iM_c})^*}\rvert$	$\lvert\mathbb{C}_{(\mathsf{DM_cM_i})^*}\rvert$	$\lvert\mathbb{C}_{(\mathsf{M_iDM_c})^*}\rvert$	$\lvert\mathbb{C}_{(\mathsf{M_iM_cD})^*}\rvert$	$\lvert\mathbb{C}_{(\mathsf{M_cDM_i})^*}\rvert$	$\lvert\mathbb{C}_{(\mathsf{M_cM_iD})^*}\rvert$	$\lvert\mathbb{C}_{\mathsf{D}(\mathsf{M_iM_c})^*}\rvert$	$\lvert\mathbb{C}_{(\mathsf{M_iM_c})^*}\rvert$
avrora-s	11	16	19	16	16	19	19	13	13
batik-s	12	16	19	16	16	19	19	13	13
eclipse-s	11	16	19	16	16	19	19	13	13
fop-s	10	16	19	16	16	19	19	13	13
h2-s	10	13	16	13	16	16	16	11	13
jenkins-s	12	16	16	16	19	19	19	11	13
jython-s	9	13	13	13	16	16	16	9	13
luindex-s	10	16	19	16	16	19	19	13	13
lusearch-s	10	16	19	16	16	19	19	13	13
openjdk8-s	9	13	13	13	16	16	16	9	13
pmd-s	11	16	19	16	16	19	19	13	13
sunflow-s	11	16	19	16	16	19	19	13	13
tomcat-s	11	16	19	16	16	19	19	13	13
tradebean-s	11	16	19	16	16	19	19	13	13
xalan-s	11	16	19	16	16	19	19	13	13
avrora-l	12	16	16	16	19	19	19	15	15
batik-l	13	19	19	19	19	19	19	15	17
eclipse-l	12	13	16	13	16	16	16	11	15
fop-l	13	16	19	16	19	19	19	15	17
h2-l	13	16	16	16	16	16	16	15	15
jenkins-l	13	19	19	19	19	19	19	13	19
jython-l	13	16	19	16	19	19	19	15	15
luindex-l	11	16	13	16	16	16	16	11	15
lusearch-l	11	16	13	16	16	16	16	11	15
openjdk8-l	15	22	22	22	22	22	22	15	19
pmd-l	12	16	16	16	16	16	16	15	15
sunflow-l	13	16	16	16	19	16	16	15	15
tomcat-l	11	13	16	13	16	16	16	11	15
tradebean-l	12	13	16	13	16	16	16	11	13
xalan-l	12	16	16	16	19	16	16	15	15
LiveJournal1-8	6	13	13	13	13	13	13	9	9
soc-Pokec-8	6	7	10	7	7	10	10	5	7
web-BerkStan-8	7	7	10	10	10	10	10	5	11
web-Google-8	7	13	13	13	16	16	16	9	11

5.1 Experimental Results

Eval-I: Effectiveness of Individual Condensation Operators. In the first experiment, we assess how effective each condensation operator is for reducing the sink graphs; note that, here (and also in the remainder of this section) we actually consider core graphs. We measure the percentage of the resulting size, obtained by the condensation operators, of the sink graph with respect to the size of the core graph. The results are shown in columns 2 – 9 of Table 3, e.g., columns 2 – 3 show $\frac{|V_{\mathsf{S}}|}{|V^c|} \times 100$ and $\frac{|E_{\mathsf{S}}|}{|E^c|} \times 100$ for operator S. Note that, we use the sequences $\mathsf{S} \circ \mathsf{D}$, $\mathsf{S} \circ \mathsf{M_i}$, and $\mathsf{S} \circ \mathsf{M_c}$ for the operators D, $\mathsf{M_i}$, and $\mathsf{M_c}$, respectively, as they assume acyclic input sink graph. From Table 3, we can see that cycle removal (i.e., operator S) has almost no effect for sparse graphs (i.e., named -s), yields only small gains for the denser graphs (i.e., named -d), and has no effect for the other four graphs that are acyclic. Operators D and $\mathsf{M_c}$ are the most effective: they result in the smallest reduced graph in most cases. Operator $\mathsf{M_i}$ is the middle ground: it achieves a reasonable reduction, but performs worse than D and $\mathsf{M_c}$. It can also be observed that operators D, $\mathsf{M_i}$, and $\mathsf{M_c}$ complement each other: they have different reduction effectiveness.

Eval-II: Effectiveness of Composed Condensations. In this experiment, we evaluate the effectiveness of our compositional approach to sink-reachability preserving reduction by comparing our fixpoint condensation (i.e., maximal condensation sequences over $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$) with the kernel condensation G_r. In addition, we also include the comparison with the state-of-the-art reachability preserving reduction techniques proposed in [1]; note that, the reduction techniques in [1] preserve all reachability information, and thus also the sink-reachability information. The results are shown in the last six columns of Table 3. We can see that our fixpoint condensation obtains a reduction that is very close to the kernel condensation, although in many cases it is slightly larger than the kernel condensation which conforms our analysis in Section 4.2.2. On the other hand, the compressed graph obtained by the techniques in [1] is much larger than our fixpoint condensation. This demonstrates the effectiveness of our compositional approach for sink-reachability preserving compression.

Eval-III: Lengths of Reduction Sequences $\mathsf{S} \circ \mathsf{C}^*$. Now, we measure the lengths of the six reduction sequences of the form $\mathsf{S} \circ \mathsf{C}^*$ on the sink graphs; here C is the composition of a permutation of the three condensation operators $\mathsf{D}, \mathsf{M_i}, \mathsf{M_c}$. Recall from Section 4.2.3 that a reduction sequence is a maximal condensation sequence, and the shorter the reduction sequence, the better the running time. The results are shown in columns 3 – 8 of Table 4, where the shortest ones among these six reduction sequences are highlighted by bold font. We can see that the permutation $\mathsf{C} \equiv \mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c}$ (column 3 in Table 4) results in a no longer length than all other five permutations across all the tested sink graphs, except on luindex-d and lusearch-d.

TABLE 5
Size, Construction Time, and Query Time (in milliseconds) of 2-Hop Indexes on G^c, on the Graph G_{fp} Obtained by Our Fixpoint Compression, and on the Graph G_{ter} Obtained by the Compression Technique in [1]

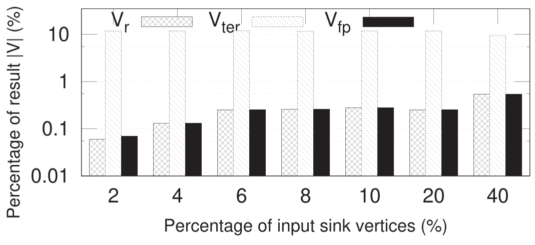
	Index on G^c			Index on G_{fp}			Index on G_{ter}		
Graph	Size	Construction Time	Query Time	Size	Construction Time	Query Time	Size	Construction Time	Query Time
avrora-s	654296	3209	19	302515	2950	3	519361	3389	16
batik-s	827578	3973	23	382063	3615	3	656467	4485	19
eclipse-s	379821	1591	12	173130	1547	1	296213	1637	9
fop-s	922000	4539	26	446893	4147	4	735426	5001	21
h2-s	718434	3399	20	306507	2992	3	547584	3544	16
jenkins-s	3287452	24851	107	1493020	22122	18	2450823	26793	81
jython-s	885707	5310	27	386727	4595	4	672469	5440	21
luindex-s	353750	1543	11	160980	1500	1	278685	1525	8
lusearch-s	353031	1620	11	160991	1501	1	278260	1555	9
openjdk8-s	1594631	10642	52	703867	9162	9	1181854	11312	42
pmd-s	667741	3192	19	300372	2960	3	528636	3325	16
sunflow-s	627732	2877	18	285669	2702	3	496825	3030	15
tomcat-s	351841	1506	11	158171	1417	1	276133	1504	9
tradebean-s	313248	1332	10	140248	1245	1	246105	1297	8
xalan-s	736309	3925	24	333058	3626	4	566122	4110	19
avrora-l	2259006	6893	33	379476	5467	3	1274178	6637	34
batik-l	2851914	8458	43	436597	6833	4	1556867	8247	38
eclipse-l	1707429	3209	19	222169	2499	1	1011163	3025	17
fop-l	3366294	10172	42	520894	8111	5	1829424	9928	43
h2-l	2472182	7164	34	383598	5518	3	1366696	6714	31
jenkins-l	20022159	168382	181	1612649	144179	19	6733811	158756	177
jython-l	3477781	65458	44	465887	62039	4	1683750	63601	40
luindex-l	1874032	3199	17	216792	2518	1	1135982	2973	15
lusearch-l	1878680	3142	17	216911	2523	1	1138942	2970	15
openjdk8-l	8499383	32312	87	881799	26299	10	3593158	30702	73
pmd-l	2404183	7071	32	378025	5594	3	1360305	6778	33
sunflow-l	2231046	5962	30	350367	4712	3	1241859	5749	28
tomcat-l	1209228	2922	16	197365	2299	1	695283	2741	15
tradebean-l	1486614	2545	15	179476	1991	1	880781	2402	15
xalan-l	2646944	7818	36	413414	6140	4	1419177	7796	34
LiveJournal1-8	117878	63870	28	85072	42587	3	85222	43398	13
soc-Pokec-8	49970	8251	9	29870	6068	1	30510	6203	1
web-BerkStan-8	1062014	706	6	3091	499	1	27904	516	1
web-Google-8	282703	3986	14	27155	2612	1	73258	2663	8

The reported construction time includes both graph compression time and index construction time.

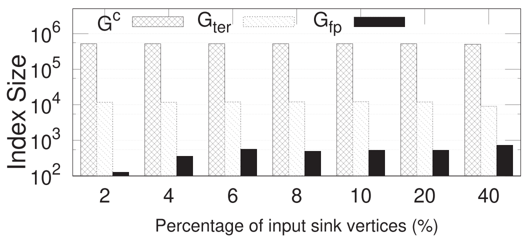
In addition, to illustrate how good are these reduction sequences in terms of length, we also show in column 2 of Table 4 the length of the shortest reduction sequence over $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$ for each sink graph. Note that, the shortest reduction sequence is obtained by doing a pruned complete search over all possible reduction sequences, which may take an exponential time to be found in the worst case and thus is not a good candidate reduction sequence to be used in practice. We can see that the length of the reduction sequence $\mathsf{S} \circ (\mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c})^*$ is at most 60 percent longer than the shortest reduction sequence, except on LiveJournal1-8 which is 116 percent longer. Moreover, the lengths of all six reduction sequences of the form $\mathsf{S} \circ \mathsf{C}^*$ are within 3x the length of the shortest compression sequence; this conforms our theoretical analysis in Section 4.2.3.

Eval-IV: Reduction Sequence to Use in Practice. Following from Eval-III, if we want a reduction sequence that will result in short length in practice as well as in the worst-case scenarios, then we recommend to use $\mathsf{S} \circ (\mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c})^*$. However, if we only care about practical performance, then we actually can drop the condensation operator D. This is because, for any sink graph G, the equivalence relation of D on G is a subset of the equivalence relation of $(\mathsf{M_c})^*$ on G; we omit the proof from the paper. That is, for any sink graph G, we have $\mathsf{S} \circ (\mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c})^*(G) = \mathsf{S} \circ (\mathsf{M_i} \circ \mathsf{M_c})^*(G)$. Nevertheless, we observe that if we apply the condensation operator D once at the beginning, usually we will have a shorter reduction sequence. The lengths of the reduction sequences $\mathsf{S} \circ \mathsf{D} \circ (\mathsf{M_i} \circ \mathsf{M_c})$ and $\mathsf{S} \circ (\mathsf{M_i} \circ \mathsf{M_c})$ on the tested sink graphs are shown in the last two columns of Table 4. We can see that the length of the reduction sequence $\mathsf{S} \circ \mathsf{D} \circ (\mathsf{M_i} \circ \mathsf{M_c})$ is always no larger than and usually is smaller than all other reduction sequences in Table 4. Thus, we recommend to use the reduction sequence $\mathsf{S} \circ \mathsf{D} \circ (\mathsf{M_i} \circ \mathsf{M_c})$ in practice; we also use this as our fixpoint reduction sequence in the remaining experiments. It is worth pointing out that unlike the reduction sequence $\mathsf{S} \circ (\mathsf{D} \circ \mathsf{M_i} \circ \mathsf{M_c})^*$, the length of $\mathsf{S} \circ \mathsf{D} \circ (\mathsf{M_i} \circ \mathsf{M_c})^*$ cannot be bounded by and in the worst case can be much larger than 3x the length of the shortest reduction sequence over $\{\mathsf{S}, \mathsf{D}, \mathsf{M_i}, \mathsf{M_c}\}$.

Eval-V: Sink Reachability Index Construction and Query Processing. In this experiment, we evaluate the effect of graph reduction for sink reachability index construction and query processing. As mentioned in Section 1, we can utilize the existing 2-hop indexing technique [36] to efficiently process queries that check whether $r_G(u) \cap r_G(v)$ is empty. That is, we can either directly construct the index by concatenating G^c with its reverse graph $\overleftarrow{G}^c$, or first reduce it into G_{fp} by our fixpoint reduction and then construct the index by concatenating G_{fp} with its reverse graph $\overleftarrow{G}_{fp}$; note that, we can also first reduce it into G_{ter} by using the techniques in [1] and then construct the index by concatenating



(a) Reduction effectiveness



(b) 2-hop index size

Fig. 10. Varying number of sink vertices on web-BerkStan.

G_{ter} with its reverse graph $\overleftarrow{G}_{ter}$. The index size, construction time and query time of the three different approaches are shown in Table 5. We can see that the construction time of the three approaches are comparable to each other; note that, the reported time also includes the time for reduction. Thus, reduction first does not incur significant overhead due to the reduction operations. When comparing the index sizes, we can see that constructing indexes on G_{ter} reduces the index sizes than directly on G^c; this is because reduction reduces the graph sizes. Constructing indexes on G_{fp} further reduces the index sizes; this is because our reduction technique results in a smaller reduced graph than the techniques in [1]. As the 2-hop index-based query processing time is linear to the index size, query processing on G_{fp} is much faster than that on G^c and on G_{ter}. As index size usually is the main bottleneck for in-memory reachability query processing [1], it is expected that reducing graphs by our fixpoint reduction technique can scale reachability indexing and query processing to much larger graphs. It is worth mentioning that the transitive reduction technique proposed in [1] is orthogonal to our techniques and can potentially be used together with our techniques to further improve the reduction ratio.

Eval-VI: Varying Percentage of Sink Vertices. We also evaluate the effectiveness of reduction techniques by varying the percentage of sink vertices on the sink graph web-BerkStan. Fig. 10a shows the reduction effectiveness, i.e., $\frac{|V_r|}{|V^c|} \times 100$, $\frac{|V_{ter}|}{|V^c|} \times 100$ and $\frac{|V_{fp}|}{|V^c|} \times 100$. We can see that the techniques in [1] are not affected by the number of sink vertices, as they do not distinguish sink vertices and non-sink vertices. When the number of sink vertices increases, the reduction becomes less effective (i.e., has a larger $\frac{|V_r|}{|V^c|}$). Nevertheless, the reduction obtained by our fixpoint reduction is always very close to the kernel condensation. Fig. 10b shows the index size by building 2-hop index on the corresponding graphs. As expected, the index sizes for G^c and G_{ter} almost are not affected by the number of sink vertices. The index size for G_{fp} increases along with the number of sink vertices due to the less effectiveness of the reduction (see Fig. 10a). Nevertheless, the index size for G_{fp} is much smaller than that for G_{ter} and G^c.

6 Conclusion

We have introduced a novel approach to compute reachability in sink graphs. Our approach uses reachability-preserving condensation operators that can be computed in linear time. We demonstrated that the linear reduction algorithms can be composed (chained), and that these composition sequences reach a unique fixpoint, representing the best possible reachability-preserving reduction that can be reached with this technique. Experiments on data sets sourced from different application areas show that our technique is effective in reducing graphs, outperforming existing state-of-the-art techniques developed for general directed graphs. Relatively short sequences yield tight approximations of the best possible reduction defined by the reachability kernel.

Acknowledgments

This research was supported partially by the Australian Government through the ARC Discovery Project funding scheme (DP210101984) and the ARC Future Fellowship funding scheme (FT180100256). The work of the first author was supported by a faculty gift by Oracle Inc.

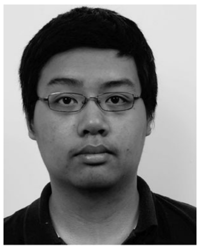
References

[1] J. Zhou, S. Zhou, J. X. Yu, H. Wei, Z. Chen, and X. Tang, “DAG reduction: Fast answering reachability queries,” in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 375–390.
[2] R. Jin, N. Ruan, S. Dey, and J. Y. Xu, “Scarab: Scaling reachability computation on large graphs,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 169–180.
[3] T. Abdessalem and I. B. Dhia, “A reachability-based access control model for online social networks,” in *Proc. Databases Soc. Netw.*, 2011, pp. 31–36.
[4] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu, “Dual labeling: Answering graph reachability queries in constant time,” in *Proc. 22nd Int. Conf. Data Eng.*, 2006, pp. 75–75.
[5] H. Yıldırım, V. Chaoji, and M. J. Zaki, “GRAIL: A scalable index for reachability queries in very large graphs,” *VLDB J.*, vol. 21, no. 4, pp. 509–534, 2012.
[6] J. Cheng, S. Huang, H. Wu, and A. W.-C. Fu, “TF-Label: A topological-folding labeling scheme for reachability querying in a large graph,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 193–204.
[7] J. X. Yu and J. Cheng, “Graph reachability queries: A survey,” in *Managing and Mining Graph Data*. Berlin, Germany: Springer, 2010, pp. 181–215.
[8] J. Dietrich, K. Jezek, S. Rasheed, A. Tahir, and A. Potanin, “Evil pickles: Dos attacks based on object-graph engineering,” in *Proc. 31st Eur. Conf. Object-Oriented Program.*, 2017, pp. 10:1–10:32.
[9] S. Rasheed, J. Dietrich, and A. Tahir, “Laughter in the wild: A study into DoS vulnerabilities in YAML libraries,” in *Proc. 18th IEEE Int. Conf. Trust Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng.*, 2019, pp. 342–349.
[10] O. Mason and M. Verwoerd, “Graph theory and networks in biology,” *IET Syst. Biol.*, vol. 1, no. 2, pp. 89–119, 2007.
[11] T. Reps, “Program analysis via graph reachability,” *Inf. Softw. Technol.*, vol. 40, no. 11, pp. 701–726, 1998.
[12] R. Nasre, “Exploiting the structure of the constraint graph for efficient points-to analysis,” *ACM SIGPLAN Notices*, vol. 47, no. 11, pp. 121–132, 2012.
[13] Y. Smaragdakis *et al.*, “Pointer analysis,” *Found. Trends Program. Lang.*, vol. 2, no. 1, pp. 1–69, 2015.
[14] M. Sridharan, D. Gopan, L. Shan, and R. Bodík, “Demand-driven points-to analysis for Java,” *ACM SIGPLAN Notices*, vol. 40, no. 10, pp. 59–76, 2005.
[15] J. Yang and J. Leskovec, “Patterns of temporal variation in online media,” in *Proc. 4th ACM Int. Conf. Web Search Data Mining*, 2011, pp. 177–186. [Online]. Available: http://doi.acm.org/10.1145/1935826.1935863

[16] A. Langevin and D. Riopel, *Logistics Systems: Design and Optimization*. Berlin, Germany: Springer, Jan. 2005.
[17] D. Coppersmith and S. Winograd, “Matrix multiplication via arithmetic progressions,” in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 1–6.
[18] V. Strassen, “Gaussian elimination is not optimal,” *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.
[19] H. Yildirim, V. Chaoji, and M. J. Zaki, “GRAIL: Scalable reachability index for large graphs,” *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 276–284, 2010.
[20] R. R. Veloso, L. Cerf, W. Meira Jr, and M. J. Zaki, “Reachability queries in very large graphs: A fast refined online search approach,” in *Proc. 17th Int. Conf. Extending Database Technol.*, 2014, pp. 511–522.
[21] E. Nuutila, “Efficient transitive closure computation in large digraphs,” Ph.D. dissertation, Lab. Inf. Process. Sci., Helsinki University of Technology, Finland, 1995.
[22] S. J. van Schaik and O. de Moor, “A memory efficient reachability data structure through bit vector compression,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 913–924.
[23] S. Seufert, A. Anand, S. Bedathur, and G. Weikum, “FERRARI: Flexible and efficient reachability range assignment for graph indexing,” in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 1009–1020.
[24] J. Cheng, S. Huang, H. Wu, and A. W.-C. Fu, “TF-Label: A topological-folding labeling scheme for reachability querying in a large graph,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 193–204.
[25] W. Fan, J. Li, X. Wang, and Y. Wu, “Query preserving graph compression,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 157–168.
[26] J. Zhou, J. X. Yu, N. Li, H. Wei, Z. Chen, and X. Tang, “Accelerating reachability query processing based on DAG reduction,” *Int. J. Very Large Data Bases*, vol. 27, no. 2, pp. 271–296, 2018.
[27] M. M. Tikir and J. K. Hollingsworth, “Efficient instrumentation for code coverage testing,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 27, no. 4, pp. 86–96, 2002.
[28] R. M. McConnell and F. De Montgolfier, “Linear-time modular decomposition of directed graphs,” *Discrete Appl. Math.*, vol. 145, no. 2, pp. 198–209, 2005.
[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd Edition. Cambridge, Massachusetts, United States: MIT Press, 2009.
[30] R. Tarjan, “Depth-first search and linear graph algorithms,” *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.
[31] T. Lengauer and R. E. Tarjan, “A fast algorithm for finding dominators in a flowgraph,” *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 1, pp. 121–141, 1979.
[32] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook, “Linear-time algorithms for dominators and other path-evaluation problems,” *SIAM J. Comput.*, vol. 38, no. 4, pp. 1533–1573, 2008. [Online]. Available: https://doi.org/10.1137/070693217
[33] T. Gallai, “Transitiv orientierbare graphen,” *Acta Mathematica Hungarica*, vol. 18, no. 1/2, pp. 25–66, 1967.
[34] S. M. Blackburn *et al.*, “The DaCapo benchmarks: Java benchmarking development and analysis,” *ACM SIGPLAN Notices*, vol. 41, no. 10, pp. 169–190, 2006.
[35] Y. Smaragdakis and M. Bravenboer, “Using datalog for fast and easy program analysis,” in *Proc. Int. Datalog 2.0 Workshop*, 2011, pp. 245–251.
[36] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, “Reachability and distance queries via 2-hop labels,” *SIAM J. Comput.*, vol. 32, no. 5, pp. 1338–1355, 2003.

Jens Dietrich received the MSc degree in mathematics and the PhD degree in computer science from the University of Leipzig, Germany. He is an associate professor with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. Prior to joining Victoria University, New Zealand he worked at Massey University, New Zealand, and as a software consultant in Germany, Namibia, Switzerland and the United Kingdom. His research interests are in program analysis, software modularity, and evolution.

Lijun Chang received the bachelor’s degree from the Renmin University of China, China, in 2007, and the PhD degree from The Chinese University of Hong Kong, Hong Kong, in 2011. He is a senior lecturer and ARC Future fellow with the School of Computer Science, The University of Sydney, Australia. He worked as a postdoc and then DECRA research fellow at the University of New South Wales, Australia from 2012 to 2017. His research interests are in the fields of big graph (network) analytics, with a focus on designing practical algorithms and developing theoretical foundations for massive graph analysis.


Long Qian received the BSc degree from Massey University, New Zealand majoring in mathematics and computer science. He is currently working toward the BSc(Hons) degree at the Victoria University of Wellington, New Zealand majoring in mathematics.

Lyndon M. Henry received the BS (Hons) degree in computer science from The University of Sydney, Australia. He is currently working toward the master’s by research degree at The University of Sydney, Australia. Their research interests include programming languages, with an emphasis on logic programming, and distributed computing.

Catherine McCartin received the MS degree from Cornell University, Ithaca, New York, and the PhD degree from Victoria University Wellington, New Zealand. She is a senior lecturer with the School of Fundamental Sciences, Massey University, New Zealand. Her research interests concentrate on graph algorithms and parameterized algorithm design, with a focus on practical applications in the areas of phylogenetics, program analysis and complex networks.

Bernhard Scholz is an associate professor with the School of Computer Science, The University of Sydney, Australia. Before joining The University of Sydney, Australia, he worked for the Technical University of Vienna, Austria and the University of Vienna, Austria in academic/research roles. He has held various visiting professorships at the University of Victoria, BC, Canada, Yonsei University, South Korea, at the Sun Microsystems Laboratories, and Oracle Labs in Brisbane. His research interests are in programming languages, static program analysis, and logic programming.