




A Truly Symbolic Linear-Time Algorithm for SCC Decomposition

Casper Abild Larsen, Simon Meldahl Schmidt, Jesper Steensgaard,
Anna Blume Jakobsen, Jaco van de Pol ,
and Andreas Pavlogiannis  

Aarhus University, Aarhus, Denmark
{jaco,pavlogiannis}@cs.au.dk

Abstract. Decomposing a directed graph to its strongly connected components (SCCs) is a fundamental task in model checking. To deal with the state-space explosion problem, graphs are often represented *symbolically* using binary decision diagrams (BDDs), which have exponential compression capabilities. The theoretically-best symbolic algorithm for SCC decomposition is Gentilini et al’s SKELETON algorithm, that uses $O(n)$ symbolic steps on a graph of n nodes. However, SKELETON uses $\Theta(n)$ symbolic objects, as opposed to (poly-)logarithmically many, which is the norm for symbolic algorithms, thereby relinquishing its symbolic nature. Here we present CHAIN, a new symbolic algorithm for SCC decomposition that also makes $O(n)$ symbolic steps, but further uses *logarithmic space*, and is thus *truly symbolic*. We then extend CHAIN to COLOREDCHAIN, an algorithm for SCC decomposition on *edge-colored graphs*, which arise naturally in model-checking a family of systems. Finally, we perform an experimental evaluation of CHAIN among other standard symbolic SCC algorithms in the literature. The results show that CHAIN is competitive on almost all benchmarks, and often faster, while it clearly outperforms all other algorithms on challenging inputs.

Keywords: Binary decision diagrams · Strongly connected components · Colored graphs

1 Introduction

Strongly connected components (SCCs) are one of the most elegant and widely applicable concepts of graph theory. They play a fundamental role in model checking for LTL and ω -regular properties, as most model-checking tasks reduce to locating cycles that traverse certain vertices in a graph [26], while strong fairness assumptions typically require an SCC decomposition at hand [21,31]. SCCs are also a key step to characterizing the attractor properties of systems, such as bottom SCCs in Markov Chains [2] and maximal end components in Markov Decision Processes [12]. From an algorithmic point of view, the simplest approach to SCC decomposition is by running a forward-backward reachability analysis

from each vertex, which results in $O(n^2)$ time on a graph of n vertices. The celebrated Tarjan’s algorithm [28], and subsequently Dijkstra’s algorithm [15] and Kosaraju-Sharir’s algorithm [27] have reduced the complexity down to $O(n)$.

In the everyday practice of model checking, systems are represented as *symbolic*, rather than *explicit* graphs. One predominant symbolic representation is via (reduced/ordered) Binary Decision Diagrams (BDDs) [9], which are found at the core of many classic and modern model checkers [13,23,19,24,3]. BDDs can offer exponential compactness of the huge state space typically involved in the model-checking task, by succinctly encoding symmetries abundant in the represented system. On the other hand, this symbolic representation gives only coarse-grained efficient access to the graph. In particular, one can query for the image and preimage of a set of vertices with respect to the edge relation, which accounts for one *symbolic step*. Although the time for performing a symbolic step may vary, it is typically significantly larger than the time taken to perform elementary operations (e.g., incrementing a counter). As such, symbolic steps serve as the complexity measure of symbolic algorithms [8,18,11].

The simplest symbolic algorithm for SCC decomposition is the FWDBWD algorithm, which computes the SCC of a vertex u as the intersection of its forward and backward sets (as in the explicit setting). As this results in $O(n^2)$ time complexity, the algorithm is often too slow in practice. The key challenge towards efficient symbolic SCC algorithms is the seeming difficulty to traverse the input graph G in a depth-first fashion, which is the technical underpinning of the $O(n)$ -time explicit SCC algorithms. Nevertheless, a series of improvements have been made in this direction: (i) a variant of FWDBWD was shown in [30] to run in time $O(\delta n)$, where δ is the diameter of G , and only becomes quadratic when $\delta = \Theta(n)$, (ii) the LOCKSTEP algorithm [7] has complexity $O(n \log n)$, while (iii) the SKELETON algorithm with complexity $O(n)$ is provably optimal [11]. Practical improvements based on heuristics have also been proposed [29,16,31].

One characteristic requirement for symbolic algorithms is that they operate in *logarithmic symbolic space*, i.e., they use logarithmically many objects, with the size of a single symbolic data structure (e.g., a BDD) counting as $O(1)$ [11]. Indeed, without this restriction, an algorithm could extract, and later analyze, an explicit representation of its input graph, thereby relinquishing its symbolic nature. Unfortunately, the theoretically optimal SKELETON algorithm uses $\Theta(n)$ space, thereby violating the logarithmic-space requirement. As such, we find that SKELETON is not truly symbolic, which also has a measurable effect: perhaps paradoxically, SKELETON is often the slowest algorithm in practice.

1.1 Our Contributions

The CHAIN algorithm. We present a new algorithm, CHAIN, for symbolically computing SCC decompositions. On input graph G with n vertices, CHAIN takes time $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1)) = O(n)$, where $\text{SCCs}(G)$ denotes the SCCs of G and $\delta(S)$ is the diameter of S . It is known that $\Omega(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1))$ is also

a lower bound for the problem [11], thus CHAIN is optimal. Moreover, CHAIN uses $O(\log n)$ symbolic data structures, thus being truly symbolic.

It is worth highlighting that CHAIN offers optimality while also being arguably the simplest among all symbolic SCC decomposition algorithms beyond FWDBWD. Indeed, CHAIN simply extends FWDBWD to accept as an argument a set of vertices K , among which to choose a pivot in the current recursive call. It is perhaps surprising that such a simple mechanism has been elusive for decades, as all previous efforts [30,7,17] relied on more elaborate procedures to either reduce or refine the $O(n^2)$ time bound. That being said, our new mechanism is somewhat insightful and with a non-trivial complexity analysis.

The COLOREDCHAIN algorithm. We extend CHAIN to COLOREDCHAIN for computing SCCs on edge-colored graphs, in which edges have colors, and SCCs are formed by restricting to monochromatic paths. Although a graph of p colors can be handled in $O(pn)$ time by breaking it to its monochromatic components and executing CHAIN on each of them, COLOREDCHAIN handles all colors simultaneously, thus benefiting from the symbolic compression of the edge relation across multiple colors. A similar approach was followed recently [6], by extending the standard LOCKSTEP algorithm [7] to colored graphs. However, the corresponding colored LOCKSTEP algorithm runs in time $O(pn \log n)$, as it inherits the $\log n$ factor from the basic LOCKSTEP algorithm.

Experimental evaluation. We implement and evaluate CHAIN in controlled, synthetic, and previously-used experimental settings. We find that CHAIN is never notably slower than other, standard algorithms, except when compared to LOCKSTEP on a few benchmarks. On the other hand, CHAIN is measurably faster than all other algorithms on demanding inputs. We further evaluate COLOREDCHAIN on colored Boolean Networks, used recently for the colored LOCKSTEP algorithm [6]. Our results indicate that COLOREDCHAIN is considerably faster than LOCKSTEP, making it a promising alternative for the analysis of Boolean networks.

2 Preliminaries

Here we set up our main notation on graphs, SCCs, and symbolic algorithms.

General notation. Given a natural number $\ell \in \mathbb{N}$, we let $[\ell] = \{1, 2, \dots, \ell\}$.

Graphs. We consider (directed) graphs $G = (V, E)$, where V is a set of n vertices and $E \subseteq V \times V$ is a set of edges. Given a set $X \subseteq V$, the *restriction* of G on X is the graph $G[X] = (X, E \cap (X \times X))$. For a vertex v , we let $\text{Pre}(v) = \{u: (u, v) \in E\}$ and $\text{Post}(v) = \{u: (v, u) \in E\}$ denote the set of *preimage* and *image* of v under E , respectively. We lift this notation to sets of vertices X , by letting $\text{Pre}(X) = \bigcup_{v \in X} \text{Pre}(v)$ and $\text{Post}(X) = \bigcup_{v \in X} \text{Post}(v)$. A path from v to u in G is a sequence of vertices $P: v = w_1, w_2, \dots, w_\ell = u$ such that, for each $i \in [\ell - 1]$, we have $(w_i, w_{i+1}) \in E$. The length of P is $|P| = \ell - 1$,

while a single vertex v serves as a path of length 0. We denote by $v \rightsquigarrow u$ the existence of a path from v to u , and call u reachable from v if there is such a path in G . For a vertex $v \in V$, we let $\text{Fwd}(v)$ and $\text{Bwd}(v)$ denote the reflexive transitive closure of $\text{Post}(v)$ and $\text{Pre}(v)$, respectively. In other words, $\text{Fwd}(v)$ (resp., $\text{Bwd}(v)$) contains the vertices that are reachable from v (resp., can reach v). Given an additional set $X \subseteq V$, we let $\text{Fwd}(v, X)$ and $\text{Bwd}(v, X)$ denote the forward and backward, respectively, set of v in the graph $G[X]$. The distance from v to u is the length of the shortest path $v \rightsquigarrow u$, i.e., $d(v, u) = \min_{P: v \rightsquigarrow u} |P|$, where we take the minimum of an empty set to be ∞ . The diameter of a set $X \subseteq V$ is $\delta(X) = \max_{v, u \in X, v \rightsquigarrow u} d(v, u)$, i.e., it is the maximum distance between any pair of vertices in X , provided that they are connected by a path.

Strongly connected components (SCCs). A set $X \subseteq V$ is strongly connected if, for every two vertices $v, u \in X$, we have $v \rightsquigarrow u$. A strongly connected component (SCC) of G is a maximal strongly connected set $S \subseteq V$. Given a vertex $v \in V$, we let $\text{SCC}(v)$ denote its SCC. We let $\text{SCCs}(G)$ denote the set of SCCs of G ; note that $\text{SCCs}(G)$ induces a partitioning on V . A set $X \subseteq V$ is called *SCC-closed* if for every $S \in \text{SCCs}(G)$, we have either $S \subseteq X$ or $S \cap X = \emptyset$. In other words, for every $v \in X$, we have $\text{SCC}(v) \subseteq X$. We sometimes call $G[X]$ SCC-closed, to indicate that X is SCC-closed (in G).

Symbolic operations and complexity measures. We consider that graphs are represented *symbolically* using Binary Decision Diagrams (BDDs) [9]. The symbolic representation suggests that efficient access to the graph can only be carried out in a *coarse-grained* way. In particular, given a symbolically-represented set of vertices X , a *symbolic operation* on X is either $\text{Pre}(X)$ or $\text{Post}(X)$, and serves as the unit of time in measuring the time complexity of symbolic algorithms. As per standard, we also perform common set operations such as union, intersection, and difference, and use a specialized function $\text{Pick}(X)$ that returns an arbitrary vertex $u \in X$. This operation is natural in symbolic SCC algorithms, as typically one needs to identify a specific vertex u in order to output $\text{SCC}(u)$. In alignment with the symbolic time complexity, the symbolic space complexity of an algorithm is measured in number of (symbolic, or not) objects it uses. As symbolic representations usually allow (in the context they are designed for) large (and sometimes, even exponential) compression, we require symbolic algorithms to operate in logarithmic symbolic space [11].

3 The CHAIN Algorithm

In this section we present the main result of this paper: a new algorithm, called CHAIN, that runs in linear time and is truly symbolic (i.e., it uses $O(\log n)$ symbolic memory). In particular, we establish the following theorem.

Theorem 1. *Given a graph $G = (V, E)$ of n nodes CHAIN computes $\text{SCCs}(G)$ in $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1))$ symbolic time and $O(\log n)$ symbolic space.*

Note that $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1)) = O(n)$, as $\text{SCCs}(G)$ partition G , while for each $S \in \text{SCCs}(G)$ we have $\delta(S) \leq |S|$. It is worth observing that $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1))$ can, however, be much smaller than n : e.g., over cliques G , this bound becomes $O(1)$. On the other hand, it is known that $\Omega(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1))$ is also a lower bound for the problem [11], hence Theorem 1 is tight. As was shown in [11], a more refined analysis of the SKELETON algorithm also achieves the time bound of Theorem 1. However, SKELETON suffers a linear space bound, and thus is not truly symbolic.

In the following, we first present CHAIN in detail in Section 3.1. It’s correctness is relatively straightforward, and stated in Section 3.2. On the other hand, its complexity analysis is more involved, and is presented in Section 3.3.

3.1 Algorithm

Here we present CHAIN in detail, develop some intuition behind its time complexity, and illustrate its execution on a small example.

Algorithm 1: CHAIN

Input: A graph $G = (V, E)$, a vertex set $K \subseteq V$

```

1 if  $V = \emptyset$  then return
2 if  $K \neq \emptyset$  then // Pick a pivot on the chain, if possible
3    $v = \text{Pick}(K)$ 
4 else
5    $v = \text{Pick}(V)$ 
6  $F = \emptyset$ ; Last =  $\emptyset$ ; Layer =  $\{v\}$ ;  $S = \{v\}$ 
7 while Layer  $\neq \emptyset$  do // Compute Fwd( $v, V$ )
8    $F = F \cup \text{Layer}$ 
9   Last = Layer
10  Layer =  $\text{Post}(\text{Layer}) \setminus F$ 
11 while  $\text{Pre}(S) \cap F \not\subseteq S$  do // Compute SCC( $v$ )
12   $S = S \cup (\text{Pre}(S) \cap F)$ 
13 output  $S$ 
14 CHAIN( $G[F \setminus S]$ , Last  $\setminus S$ ) // Recursive call on the forward set
15 CHAIN( $G[V \setminus F]$ ,  $\text{Pre}(S) \setminus F$ ) // Recursive call on the rest

```

The CHAIN algorithm. Algorithm 1 presents CHAIN in pseudocode. The principle of operation of the algorithm is, perhaps, surprisingly simple. Given a $G = (V, E)$ and a pivot vertex v of G , the algorithm computes $\text{SCC}(v)$ in two phases, similarly to the standard FWDBWD algorithm. In particular:

1. The first phase computes $\text{Fwd}(v, V)$ (i.e., the forward set of v in V) as the least fixed point $F = \mu X. \{v\} \cup \text{Post}(X)$ (loop in Line 7).
2. The second phase outputs $\text{SCC}(v)$ by iteratively computing the least fixed point $S = \mu X. \{v\} \cup (\text{Pre}(X) \cap F)$ (loop in Line 11).
3. Finally, the computation proceeds recursively on the SCC-closed components $G[F \setminus S]$ and $G[V \setminus F]$ that partition $V \setminus S$ (Line 14 and Line 15).

However, in order to avoid the high complexity, CHAIN passes along each recursive call the K argument (initially $K = \emptyset$). This argument restricts the recursive call to pick its next pivot v such that $v \in K$; choosing the right set to pass as K makes the algorithm achieve its tight time complexity.

Conceptually, after $\text{Fwd}(v, V)$ has been computed, the first recursive call (Line 14) chooses K to be the set of vertices that are of maximum distance from v (and not in $\text{SCC}(v)$, as those are output in Line 13). On the other hand, the second recursive call (Line 15) chooses K to be the predecessors of $\text{SCC}(v)$. Although the formal complexity analysis is somewhat involved (see Section 3.3), the key, high-level idea is as follows. When computing $\text{Fwd}(v, V)$, the algorithm has taken a number of symbolic steps that is proportional to the maximum distance of a vertex from v . The chain of recursive calls starting in Line 14 and followed by all recursive calls in Line 15 until $\text{Pre}(S) \cap F = \emptyset$, ensures that the algorithm will output all SCCs, in reverse order, along a maximal path from v to a vertex in $\text{Fwd}(v, V) \setminus \text{SCC}(v)$. This amortizes the high cost of computing $\text{Fwd}(v, V)$ in the current call to the cost of outputting these SCCs in future calls, leading to only a constant factor increase in the overall complexity.

Besides viewing CHAIN as an augmentation of the FWDBWD algorithm with a restriction on pivots, the algorithm can also be seen as a simplification of the SKELETON algorithm [17]. Indeed, the computation of skeletons in the latter serves the exact purpose to force the recursion to output SCCs in the same order as in our chain argument above. As we show here, computing skeletons is redundant: dropping them makes the algorithm simpler, truly symbolic, while not sacrificing any of its time-complexity guarantees.

Example. Fig. 1 illustrates CHAIN on a graph $G = (V, E)$ (left). The tree T (right) represents the recursion of CHAIN as it outputs $\text{SCCs}(G)$. We identify every vertex of T by a vertex $v \in V$ for which $\text{SCC}(v)$ is computed in the corresponding step. We subscript variables of the algorithm with v to denote their value at that step. E.g., V_v denotes the vertex set in the recursive call that computed $\text{SCC}(v)$, and F_v denotes the forward set computed after the loop of Line 7 has completed. The edges of T are labeled with the line that performed the respective recursive call.

The key observation for understanding the complexity of CHAIN is as follows. In the first step, the algorithm has paid the high cost of 5 symbolic steps to compute F_1 , while its output is a small SCC of 2 vertices. However, the path $1 \xrightarrow{14} 6 \xrightarrow{15} 4 \xrightarrow{15} 3$ in T forms a chain from vertex 6, which is of maximum distance from 1, back to vertex 3 that is adjacent to $\text{SCC}(1)$. The cost of computing F_1 can thus be amortized to outputting the SCCs along this chain (i.e., $\text{SCC}(3)$, $\text{SCC}(4)$, $\text{SCC}(6)$), yielding only a linear overhead. As we prove in Section 3.3, this behavior is not accidental, but guaranteed in every recursive call.

3.2 Correctness

We start with the soundness of CHAIN, i.e., it only outputs SCCs of G .

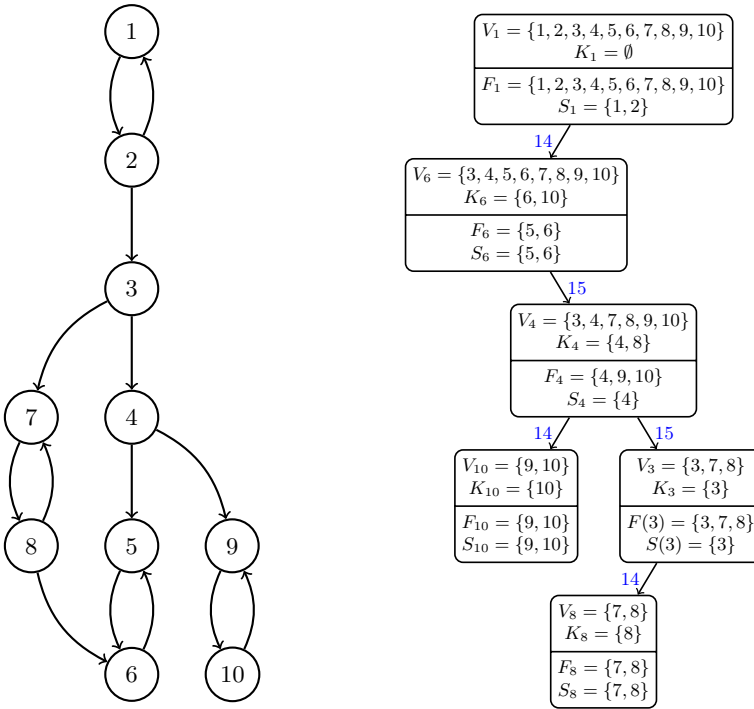


Fig. 1. An input graph (left), and the recursive computation of CHAIN (right).

Lemma 1. *In every call of CHAIN, Line 13 outputs an SCC of G.*

Proof. Consider any call to CHAIN on input $G' = (V', E'), K'$, with $K' \subseteq V'$. The algorithm first picks a vertex v from either V' or K' , with $v \in S$, where S is the set outputted in Line 13. It is straightforward to see that, after the loop in Line 7 has executed, we have $F = \text{Fwd}(u, V')$, while after the loop in Line 11 has executed, we have $S = \text{Fwd}(u, V') \cap \text{Bwd}(u, V')$. It suffices to argue that G' is an SCC-closed subgraph of G , which implies that $S = \text{SCC}(v)$.

The statement is true initially, as $G' = G$. Now, assuming that the statement holds on some input $G' = (V', E'), K'$ we argue each of $G'[F \setminus S]$ and $G'[V' \setminus F]$, in Line 14 and Line 15, respectively, is SCC-closed. Indeed, F is closed under Post operations and thus SCC-closed. As S is an SCC of X , we have that $F \setminus S$ is also SCC closed. Since $F \setminus S, S,$ and $V' \setminus F$ partition V' , we have that $G'[V' \setminus F]$ is also SCC-closed. The desired result follows. \square

Lemma 2. *CHAIN outputs every SCC in $\text{SCCs}(G)$ exactly once.*

Proof. The statement follows from the fact that, in every recursive call on input $G' = (V', E')$, the sets $F \setminus S, S,$ and $V' \setminus F$ partition V' . \square

3.3 Complexity Analysis

We now present the (symbolic) time and space complexity analysis of CHAIN. For measuring time, we only count the number of $\text{Pre}(\cdot)$ and $\text{Post}(\cdot)$ operations.

Consider any input $G = (V, E)$, and let T be the recursion tree produced by the execution of CHAIN on G , as in Fig. 1. We will use lowercase (resp., uppercase) letters to refer to the vertices of G (resp., T), and we will subscript the variables of the algorithm with vertices of T (e.g., V_A) to refer to variables in the recursive call associated with the recursive step (at A). T has labeled directed edges $A \xrightarrow{f} B$, where $f \in \{14, 15\}$ denotes the line of the recursive call that made B a child of A in T . Without loss of generality, we consider that every vertex A of T corresponds to a recursive call with $V_A \neq \emptyset$.

Main complexity analysis. Consider an edge $A \xrightarrow{14} B$ in T , and the path $A \xrightarrow{14} B_1 \xrightarrow{15} B_2 \xrightarrow{15} \dots \xrightarrow{15} B_k$, where B_k is the first vertex B for which $\text{Pre}(S_B) \setminus F_B = \emptyset$ in Line 15. Let $\text{Levels}(A)$ denote the number of iterations executing in Line 7, and note that $\text{Levels}(A) = \max_{u \in V_A} d(v_A, u)$. The crux of the complexity proof of CHAIN is the following lemma.

Lemma 3. $\text{Levels}(A) \leq \delta(\text{SCC}(v_A)) + 1 + \sum_{i \in [k]} (\delta(\text{SCC}(v_{B_i})) + 1)$.

Before we prove Lemma 3, we show how it leads to the complexity of Theorem 1. Given a vertex A of T , let $\mathcal{T}(A)$ denote the running time of CHAIN on the subtree of T rooted at A . Let $A \xrightarrow{14} B$ and $A \xrightarrow{15} C$ be the children of A , and the path $A \xrightarrow{14} B_1 \xrightarrow{15} B_2 \xrightarrow{15} \dots \xrightarrow{15} B_k$ as defined above (thus $B_1 = B$). Then $\mathcal{T}(A)$ satisfies the following recurrence.

$$\begin{aligned}
 \mathcal{T}(A) &\leq \overbrace{\text{Levels}(v_A)}^{\text{loop in Line 7}} + \overbrace{\delta(\text{SCC}(v_A)) + 1}^{\text{loop in Line 11}} + \overbrace{\mathcal{T}(B)}^{\text{Line 14}} + \overbrace{1 + \mathcal{T}(C)}^{\text{Line 15}} \\
 &\leq \sum_{i \in [k]} (\delta(\text{SCC}(v_{B_i})) + 1) + \delta(\text{SCC}(v_A)) + 1 \\
 &\quad + \delta(\text{SCC}(v_A)) + 1 + \mathcal{T}(B) + 1 + \mathcal{T}(C) \qquad \qquad \qquad \text{[Lemma 3]} \\
 &= \sum_{i \in [k]} (\delta(\text{SCC}(v_{B_i})) + 1) + 2\delta(\text{SCC}(v_A)) + 3 + \mathcal{T}(B) + \mathcal{T}(C)
 \end{aligned}$$

For every i iterating in $\sum_{i \in [k]} (\delta(\text{SCC}(v_{B_i})) + 1)$, the vertex v_{B_i} will not appear in such a sum in any other vertex A' of T . Indeed assume towards contradiction that for some vertex B_i there are two vertices $A \neq A'$ and paths

$$P: A \xrightarrow{14} B_1 \xrightarrow{15} B_2 \xrightarrow{15} \dots \xrightarrow{15} B_i \quad \text{and} \quad P': A' \xrightarrow{14} B'_1 \xrightarrow{15} B'_2 \xrightarrow{15} \dots \xrightarrow{15} B'_i$$

with $B'_i = B_i$. Due to the edge labels, none can be a sub-path of the other, which, in turn, contradicts the tree structure of T . Given such a vertex B_i , let $\mathcal{A}(B_i)$ denote its unique ancestor in T that appears as vertex A in the path P above.

The total running time of CHAIN on G is $\leq \sum_{B \in T} (3\delta(\text{SCC}(v_B)) + 4)$, obtained by counting for each vertex B of T (i) the $2\delta(\text{SCC}(v_B)) + 3$ symbolic operations from its own recursive call, plus (ii) $\delta(\text{SCC}(v_B)) + 1$ symbolic operations from the call at $\mathcal{A}(B)$. Hence the total number of symbolic steps is $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1))$.

Proof of Lemma 3. We now turn our attention to the proof of Lemma 3. Consider again the path $A \xrightarrow{14} B_1 \xrightarrow{15} B_2 \xrightarrow{15} \dots \xrightarrow{15} B_k$ of T as defined above. For simplicity of notation, let $v_i = v_{B_i}$, for $i \in [k]$. Clearly $\text{SCC}(v_i) \neq \text{SCC}(v_j)$ for $i \neq j$. We start with two simple lemmas.

Lemma 4. *For every $i \in [k]$, we have $K_{B_i} \neq \emptyset$.*

Proof. The statement holds for $i = 1$, since otherwise $\text{Last}_A \setminus S_A = \emptyset$, implying that $F_A \setminus S_A = V_{B_1} = \emptyset$, and thus B_1 would not be a vertex of T . The statement also holds for all $i > 1$, by construction of the path to B_k . \square

Lemma 5. *For all $i \in [k - 1]$, we have $v_i \in \text{Fwd}(v_k)$.*

Proof. The lemma follows from the more general statement that $v_i \in \text{Fwd}(v_{i+1})$. Indeed, by Lemma 4, we have that $v_{i+1} \in \text{Pre}(S_{B_i})$, while $S_{B_i} = \text{SCC}(v_i)$. \square

We call a vertex u *critical* if it is the first vertex w in a path from v_A to v_k in V_A , such that $w \notin \text{SCC}(v_A)$. We further call a path $u \rightsquigarrow v_k$ *critical* if u is a critical vertex. In the example of Fig. 1, for the first call to CHAIN, where $v_A = 1$, vertex 3 is a critical vertex and the path $3 \rightarrow 4 \rightarrow 5 \rightarrow 6$ is a critical path. The following lemma captures the fact that every recursive call B_i is performed on a vertex set V_{B_i} that is adjacent to $\text{SCC}(v_A)$.

Lemma 6. *For all $i \in [k]$, the set V_{B_i} has a critical path.*

Proof. The proof follows induction on i . For $i = 1$, we have $V_{B_1} = \text{Fwd}(v_A, V_A) \setminus \text{SCC}(v_A)$. Since $A \xrightarrow{14} B_1$ in T , we have $\text{Fwd}(v_A, V_A) \setminus \text{SCC}(v_A) = V_{B_1} \neq \emptyset$, thus the statement holds for $i = 1$. Now assume that the statement holds for some $i \geq 1$, and we argue that it holds for $i + 1$. Take any critical path $P: u \rightsquigarrow v_k$ in V_{B_i} , and assume towards contradiction that P is not a path in $V_{B_{i+1}}$ (i.e., at least one vertex of P is outside $V_{B_{i+1}}$). Since $V_{B_{i+1}} = V_{B_i} \setminus \text{Fwd}(v_i, V_{B_i})$, we obtain that P has a vertex w with $w \in \text{Fwd}(v_i, V_{B_i})$, and hence $v_k \in \text{Fwd}(v_i)$. By Lemma 5, we also have $v_i \in \text{Fwd}(v_k)$, thus $\text{SCC}(v_i) = \text{SCC}(v_k)$, violating the choices of v_i . Thus $V_{B_{i+1}}$ has a critical path. \square

Specifically for the case $i = k$, the following is a strengthening of Lemma 6, showing that $\text{SCC}(v_k)$ (only a subset of V_{B_k}) is also adjacent to $\text{SCC}(v_A)$.

Lemma 7. *$\text{SCC}(v_k)$ contains a critical vertex.*

Proof. By Lemma 6, we have a critical path $u \rightsquigarrow v_k$ in V_{B_k} . By construction, $(\text{Pre}(\text{SCC}(v_k)) \cap V_{B_k}) \setminus \text{SCC}(v_k) = \emptyset$, thus $u \in \text{SCC}(v_k)$. \square

Let v_{k+1} be a critical vertex in $\text{SCC}(v_k)$, whose existence is guaranteed by Lemma 7. Given a vertex $u \in V_A$, we write $\ell(u)$ for the distance of u from v_A in V_A . Note that $\text{Levels}(A) = \ell(v_1)$. Observe that for all $u, v \in V_A$, if $u \in \text{SCC}(v)$ then $\ell(u) - \ell(v) \leq \delta(\text{SCC}(v))$. The following two lemmas relate the distances $\ell(v_i)$ with the diameters of SCCs, and lead to the proof of Lemma 3.

Lemma 8. *We have $\ell(v_{k+1}) \leq \delta(\text{SCC}(v_A)) + 1$.*

Proof. By definition, there is a vertex $w \in \text{Pre}(v_{k+1}) \cap \text{SCC}(v_A)$. We have $\ell(w) \geq \ell(v_{k+1}) - 1$, while $\ell(w) \leq \delta(\text{SCC}(v_A))$, hence $\ell(v_{k+1}) \leq \delta(\text{SCC}(v_A)) + 1$. \square

Lemma 9. *For every $i \in [k]$, we have $\ell(v_i) - \ell(v_{i+1}) \leq \delta(\text{SCC}(v_i)) + 1$.*

Proof. The statement holds trivially when $\ell(v_i) \leq \ell(v_{i+1})$. Now consider the case that $\ell(v_i) > \ell(v_{i+1})$. If $i = k$, then by our choice of v_{k+1} , we have $v_{i+1} \in \text{SCC}(v_i)$, thus $\ell(v_i) - \ell(v_{i+1}) \leq \delta(\text{SCC}(v_i))$. Now consider that $i < k$. By construction, there is a vertex $w \in \text{SCC}(v_i) \cap \text{Post}(v_{i+1})$. Then $\ell(v_i) - \ell(w) \leq \delta(\text{SCC}(v_i))$, while $\ell(w) \leq \ell(v_{i+1}) + 1$, resulting in $\ell(v_i) - \ell(v_{i+1}) \leq \delta(\text{SCC}(v_i)) + 1$. \square

Proof (of Lemma 3).

$$\begin{aligned} \text{Levels}(A) = \ell(v_1) &= \sum_{i \in [k]} (\ell(v_i) - \ell(v_{i+1})) + \ell(v_{k+1}) && \text{[algebra]} \\ &\leq \sum_{i \in [k]} (\ell(v_i) - \ell(v_{i+1})) + \delta(\text{SCC}(v_A)) + 1 && \text{[Lemma 8]} \\ &\leq \sum_{i \in [k]} (\delta(\text{SCC}(v_i)) + 1) + \delta(\text{SCC}(v_A)) + 1 && \text{[Lemma 9]} \end{aligned}$$

\square

Space complexity. Finally, we address the $O(\log n)$ symbolic-space complexity of Theorem 1. CHAIN uses $O(1)$ symbolic sets in each recursive call. To achieve the $O(\log n)$ bound, it suffices to first follow the recursive call between Line 14 and Line 15 with the smaller graph input. This results in $O(\log n)$ pending recursive calls at any step of the execution, leading to storing $O(\log n)$ symbolic sets overall. Note that this requires a function $\text{Count}(X)$ that returns the size of a symbolically represented set X . This is not a problem: BDDs are equipped with such operations, and their complexity is only linear in the size of the *representation* of X , even though X might be exponentially large.

4 Extension to Colored Graphs

In this section we turn our attention to colored graphs, where the edge relation is parameterized by colors, and SCCs are formed with respect to monochromatic

components of the graph. Each edge color stands for a different binary relation, and all colors together allow to superpose several graphs on top of each other. Although each monochromatic graph could be represented in isolation, this superpositioning allows for an efficient symbolic representation, especially when the edge relations are highly similar. In turn, this asks for efficient symbolic algorithms that are able to exploit similarities between colors. Our study of this setting is inspired by the recent extension of LOCKSTEP to colored graphs [6].

4.1 Edge-Colored Graphs

Here we lift some of our graph notation from Section 2 to the colored setting.

Colored graphs. An edge-colored graph $G = (V, C, E)$ consists of a set of n vertices V , a set of p colors C , and an edge relation $E \subseteq V \times C \times V$. Given a color $c \in C$, we let $G_c = (V, E_c)$ be the *projection* of G on c , where $E_c = E \cap (V \times \{c\} \times V)$ restricts the edge relation to color c . Given two vertices $v, u \in V$, we write $v \overset{c}{\rightsquigarrow} u$ to denote that there is a path $v \rightsquigarrow u$ in G_c , and say that u is c -reachable from v in G . A *colored vertex set* is a set $X \subseteq V \times C$. The *restriction* of G on X is the colored graph $G[X] = (V', C', E')$, where (i) $V' = \{v : \exists c \in C. (v, c) \in X\}$, (ii) $C' = \{c : \exists v \in V. (v, c) \in X\}$, and (iii) $E' = \{(u, c, v) : (u, c), (v, c) \in X\}$. Given such a set X , we let $\text{Pre}(X) = \{(u, c) : \exists (v, c) \in X. (u, c, v) \in E\}$, and $\text{Post}(X) = \{(u, c) : \exists (v, c) \in X. (v, c, u) \in E\}$. We call a set $\mathcal{V} \subseteq V \times C$ *degenerate* if for all $c \in \text{Colors}$, we have $|\mathcal{V} \cap (V \times \{c\})| \leq 1$, i.e., \mathcal{V} has at most one vertex per color. Given a degenerate set \mathcal{V} , we let $\text{Fwd}(\mathcal{V}) = \{(v, c) : \exists (u, c) \in \mathcal{V} \text{ and } u \overset{c}{\rightsquigarrow} v\}$, i.e., it is the set of colored vertices reached by each colored vertex in \mathcal{V} . We similarly let $\text{Bwd}(\mathcal{V}) = \{(v, c) : \exists (u, c) \in \mathcal{V} \text{ and } v \overset{c}{\rightsquigarrow} u\}$. Note that for degenerate sets, $\text{Fwd}(\text{Bwd})$ is the transitive closure of $\text{Post}(\text{Pre})$. Further, given a colored vertex set X , we let $\text{Fwd}(\mathcal{V}, X)$ (resp., $\text{Bwd}(\mathcal{V}, X)$) be the set of colored vertices reached by (resp., reaching) each colored vertex in \mathcal{V} in the subgraph $G[X]$.

Colored SCCs. Given a colored graph $G = (V, C, E)$, a c -colored SCC of G is a pair $S = (R, c) \subseteq V \times \{c\}$ such that R is an SCC of G_c . Given a vertex $v \in V$ and a color $c \in C$, we write $\text{SCC}(v, c)$ for the SCC of v in G_c . We let $\text{SCCs}(G)$ denote the set of SCCs of G , and observe that $\text{SCCs}(G)$ partitions $V \times C$. A set $X \subseteq V \times C$ is *SCC-closed* if for every color $c \in C$, the set $X \cap (V \times \{c\})$ is SCC closed in G_c . Given an SCC-closed set X , we will also call $G[X]$ SCC-closed. Given a degenerate set \mathcal{V} , we write $\text{SCC}(\mathcal{V})$ for the set of SCCs $\{(R, c) : (v, c) \in \mathcal{V} \text{ and } R = \text{SCC}(v) \text{ in } G_c\}$.

Symbolic operations. Similarly to the non-colored setting, we use symbolic operations $\text{Pre}(X)$ and $\text{Post}(X)$ on sets $X \subseteq V \times C$, which incur a unit time cost. We further perform unions, intersections and differences on subsets of $V \times C$, and use a specialized operation $\text{Pick}(X)$ that returns an arbitrary pair $(v, c) \in X$. Finally, we consider at our disposal a function $\text{Pivots}(X)$, that acts on sets $X \subseteq V \times C$ and returns a maximal degenerate subset of X containing one pair (v, c) per color c appearing in X . This operation can be performed by combining Pick with basic set operations, and has also appeared in other works [6].

4.2 The COLOREDCHAIN Algorithm

Here we present our extension of CHAIN for handling edge colored graphs.

Algorithm 2: COLOREDCHAIN

```

Input: A graph  $G = (V, C, E)$ , two colored vertex sets  $X, K \subseteq V \times C$ ,
1 if  $X = \emptyset$  then return
2  $\mathcal{V} = \text{Pivots}(K \cup (X \setminus (V \times \text{Colors}(K))))$  // A degenerate set of pivots
3  $F = \emptyset$ ;  $\text{Last} = \emptyset$ ;  $\text{Layer} = \mathcal{V}$ ;  $S = \mathcal{V}$ 
4 while  $\text{Layer} \neq \emptyset$  do // Compute  $\text{Fwd}(\mathcal{V}, X)$ 
5    $F = F \cup \text{Layer}$ 
6    $\text{Last} = \text{Layer} \cup (\text{Last} \setminus (V \times \text{Colors}(\text{Layer})))$ 
7    $\text{Layer} = \text{Post}(\text{Layer}) \setminus F$ 
8 while  $\text{Pre}(S) \cap F \not\subseteq S$  do // Compute  $\text{SCC}(\mathcal{V})$ 
9    $S = S \cup (\text{Pre}(S) \cap F)$ 
10 output  $S$ 
11  $\text{COLOREDCHAIN}(G[F \setminus S], F \setminus S, \text{Last} \setminus S)$ 
12  $\text{COLOREDCHAIN}(G[X \setminus F], X \setminus F, \text{Pre}(S) \setminus F)$ 

```

The COLOREDCHAIN algorithm. Algorithm 2 presents COLOREDCHAIN in pseudocode. The algorithm takes as input an edge-colored graph $G = (V, C, E)$, as well as two colored vertex sets X and K (initially $X = V \times \text{Colors}$ and $K = \emptyset$). In words, the current and future recursive steps will compute the colored SCCs of G that are subsets of X . The set K serves the same purpose as in the basic CHAIN algorithm, i.e., to restrict the set of vertices over which we select pivots in the current recursive call, towards the linear-time properties of the algorithm. The algorithm starts by selecting a degenerate set of pivots \mathcal{V} in Line 2, with the goal to output each $\text{SCC}(v, c)$, for $(v, c) \in \mathcal{V}$ in the current recursive step. The pivot set is constructed to contain one pair (v, c) for every color c present in X . If c is also present in K , then the algorithm selects a pivot $(v, c) \in K$, otherwise, it chooses an arbitrary pivot from X . The algorithm then computes $\text{SCC}(\mathcal{V})$ as $\text{Fwd}(\mathcal{V}, X) \cap \text{Bwd}(\mathcal{V}, X)$, similarly to the non-colored case (where \mathcal{V} is simply a non-colored vertex). In the i -th iteration of the loop of Line 4, the variable Last contains the vertices (u, c) that have maximum distance $\leq i$ from $(v, c) \in \mathcal{V}$. As these maximal distances might converge at different lengths for different colors, extra care is taken in Line 6 to maintain the converged colors in the next iteration. Finally, the algorithm outputs $\text{SCC}(\mathcal{V})$ (Line 10), and proceeds recursively on the disjoint subsets $F \setminus S$ and $X \setminus F$ (Line 11 and Line 12). The K argument is passed on each recursive call in the same way as in the CHAIN algorithm, so that, in effect, the time taken to compute F is amortized by the time to output colored SCCs in subsequent recursive calls (where now the amortization also takes place among colors). Observe that, in the special case of $p = 1$ color, COLOREDCHAIN operates identically to CHAIN.

Correctness and complexity. Due to the similarity of COLOREDCHAIN to CHAIN, we will only sketch the main arguments for its correctness and complexity. The key observation for correctness is that each recursive call processes an

SCC-closed subgraph of G . Indeed, given an SCC-closed colored vertex set X , for any vertex $(v, c) \in X$, we have $\text{SCC}(v, c) = \text{Fwd}(\{(v, c)\}, X) \cap \text{Bwd}(\{(v, c)\}, X)$. Hence $S = \text{SCC}(\mathcal{V})$ in Line 10. As $F \cup S$ is closed under Post operations and S is an SCC of X , we have $F \setminus S$ (and thus also $X \setminus F$) is SCC closed.

The complexity of COLOREDCHAIN is $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1)) = O(pn)$, as every vertex v belongs to exactly one $\text{SCC}(v, c)$ for each color $c \in \mathcal{C}$. This bound follows from amortizing the number of iterations of the loop in Line 4 to the diameter of a color that converges last in the loop. Observe that the computation on the remaining colors comes “for free”. This is the benefit of treating all colors symbolically (as opposed to each monochromatic graph G_c separately). The same observation holds for the while loop in Line 8.

5 Experiments

In this section we report our experimental evaluation of the new algorithms CHAIN and COLOREDCHAIN on three classes of benchmarks. We compared their performance to the standard algorithms FWDBWD [30], LOCKSTEP [7] (and its recent colored variant [6]) and SKELETON [17]. Our experiments were run on a Linux machine with 2.4GHz CPU speed and 60GB of memory (using 1 core).

5.1 Experiments on Synthetic Benchmarks

To better illustrate the behavior of the various algorithms, we start with a controlled setting of synthetic benchmarks.

Setup. We performed a controlled experiment on product graphs $G_k^i = \mathcal{L}_{k-i} \times \mathcal{C}_i$, where \mathcal{L}_j (resp, \mathcal{C}_j) denotes a line graph (resp., cycle graph) of size 2^j . This setup follows [4]. Observe that G_k^i has 2^{k-i} SCCs, of size (and diameter) 2^i each. Our implementation is in C++ and based on the Sylvan BDD library [14]. Recall that the behavior of each algorithm depends on the non-determinism involved in the Pick operation, that returns an arbitrary vertex of a given vertex set. Sylvan returns the vertex with the smallest (binary encoded) ID. We generated two variants of this setting: one in which vertex IDs follow an incremental order in each graph component, and one in which they are uniformly random.

Results. Fig. 2 shows the number of symbolic steps per algorithm, for graphs G_{10}^i , $i \in \{0\} \cup [10]$. When the vertex encoding follows sequential IDs (left), FWDBWD exhibits its worst-case $\Theta(n^2)$ performance on graphs with many SCCs (i.e., small i) as it repeatedly Pick’s pivots with large forward sets. As i increases, the number of SCCs decreases, and FWDBWD eventually terminates in the first call (for $i = 10$). On the other hand, the other algorithms exhibit almost identical, $O(n)$ performance. In particular, every recursive call of LOCKSTEP Pick’s a vertex v whose backward set equals $\text{SCC}(v)$; thus the algorithm converges in a number of steps that is proportional to $\delta(\text{SCC}(v))$, leading to $\Theta(n)$ performance. Finally, after the first call, SKELETON and CHAIN output SCCs in the

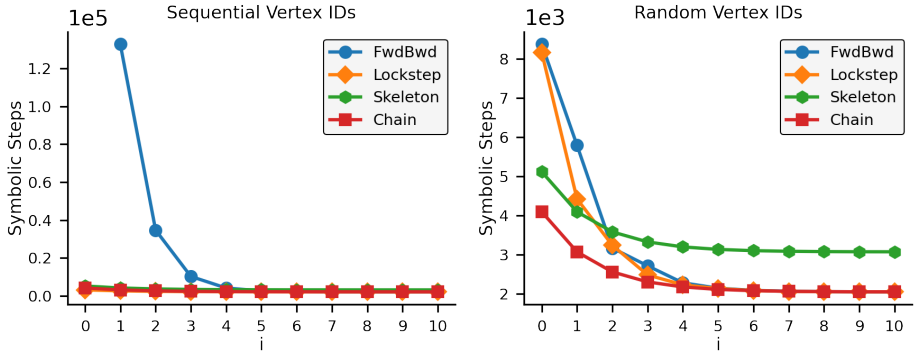


Fig. 2. Experimental results on product graphs $G_{10}^i = \mathcal{L}_{10-i} \times \mathcal{C}_i$.

reverse order of FWDBWD, performing in each step a number of symbolic steps that is proportional to the diameter of the SCC, like LOCKSTEP.

When the vertex encoding follows random IDs (right), every recursive call of FWDBWD and LOCKSTEP Pick’s a pivot whose first component is roughly in the middle of the line segment that is processed in that call. Hence the two algorithms have similar performance, which follows $\Theta(n \log n)$ behavior for large lines (i.e., when i is small). On the other hand, SKELETON and CHAIN spend $O(\sum_{S \in \text{SCCs}(G)} (\delta(S) + 1))$ symbolic steps. Naturally, for larger lines, the two algorithms spend more steps for computing the forward sets of their pivots, a cost that is amortized in later recursive calls by a constant factor. Observe, however, that SKELETON pays a larger constant factor, as the construction of skeletons requires the forward sets to also be traversed backwards. This results in SKELETON having the worst performance relative to the other algorithms when the number of SCCs decreases (i.e., as i gets larger), as there are fewer recursive calls to amortize the high cost of skeleton computation. Finally, we remark that for small and large i , SKELETON constructs (in expectation) $\Theta(n)$ BDDs, hence this is a family of graphs exposing the non-symbolic nature of the algorithm.

5.2 Experiments on Uncolored Graphs

To better understand the performance of the various algorithms in the wild, we continue with their evaluation on standard model-checking benchmarks.

Setup. We considered benchmarks from the following categories:

- 1-safe Petri Net models from MCC, the Model Checking Contest [22].
- DiVinE models from BEEM, the Benchmark of Explicit Models [25].

In order to create equal experimental circumstances for all models, we used the language-independent model checker LTSmin [19] to generate the disjointly partitioned symbolic transition relations for all these models. As symbolic representation, we chose the multi-core BDD package Sylvan [14]. We implemented all four algorithms of the previous section inside LTSmin. We disregarded graphs

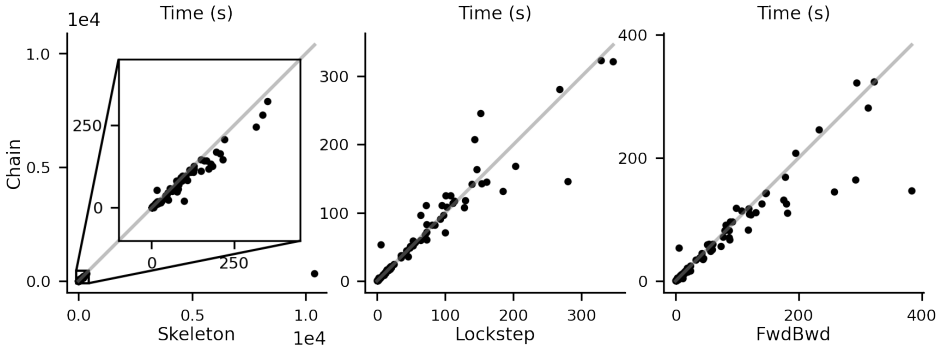


Fig. 3. Experimental results on PNML and DiVinE models.

of size $< 10^4$, as such graphs are handled more efficiently by explicit algorithms. This led to a pool of 101 benchmarks. We measured the average time (across three runs) each algorithm took on each benchmark, while discarding the overhead due to state-space generation.

Results. Fig. 3 shows the running times of CHAIN against SKELETON, LOCKSTEP and FWD BWD. Compared to the only other theoretically optimal algorithm SKELETON, CHAIN is almost always somewhat faster, with the exception of one benchmark on which CHAIN is an order of magnitude faster. When compared to LOCKSTEP, we find the two algorithms to be incomparable, with CHAIN being slower on some benchmarks but faster on others. Indeed, we expect that LOCKSTEP behaves adequately in most practical scenarios, while its $\log n$ slowdown (as demonstrated in Section 5.1) is witnessed only rarely. Finally, we find that CHAIN is measurably and consistently equally-or-better performing than FWD BWD.

5.3 Experiments on Colored Graphs

Finally, we turn our attention to colored graphs. We used models of discrete control systems representing Biological Genetic Networks [20]. In high level, a Boolean Network (BN) is defined by a set of Boolean variables $X = \{x_1, \dots, x_k\}$ and update functions of the form $x_i := \varphi_i$, where each φ_i is a Boolean combination over variables X . State updates are performed by nondeterministic applications of the functions φ_i . In Colored Boolean Networks (CBNs), uninterpreted function symbols are used to represent uncertainty. For instance, $x_1 := x_2 \wedge f(x_3, x_4)$ represents that x_1 has a positive dependence on x_2 and an unknown dependence on x_3 and x_4 . A single color corresponds to an assignment of Boolean functions to the uninterpreted function symbols. The set of colors is further restricted by constraints representing biological knowledge. This setting is inspired by its use to evaluate the recently introduced colored LOCKSTEP [6].

Setup. We implemented our new COLOREDCHAIN-algorithm in Scala, using JavaBDD (wrapping the classical BDD package BuDDy) with recommended

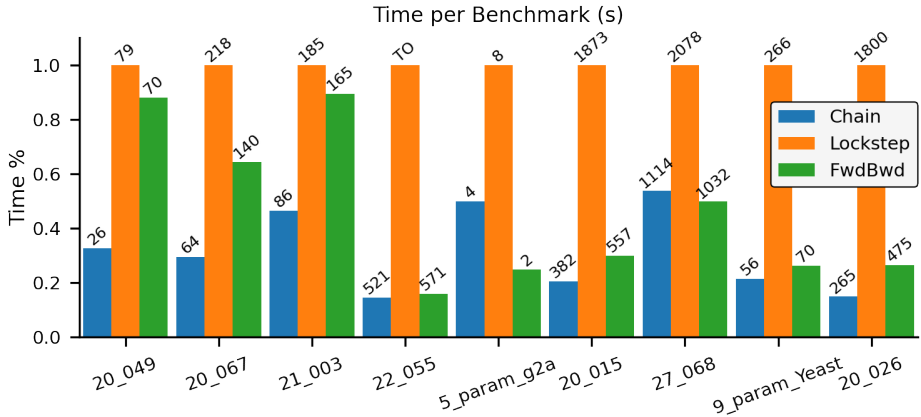


Fig. 4. Experimental results on colored graphs from AEON models (seconds).

settings. We also reimplemented colored LOCKSTEP from [6] (without preprocessing) and FWDBWD in Scala/JavaBDD. We used the CBNs coming from the GINsim Boolean network database [10], represented in the AEON format that supported the experiments in [6], accessed at [1]. We focused on benchmarks with $np \geq 10^4$, as the rest were run in $< 0.2s$ by all algorithms. We remark that most of these CBNs generate huge graphs; for the purposes of our evaluation, we timed our experiments within 1h, which yielded a pool of 9 benchmarks.

Results. Fig. 4 shows the running time of each of the three algorithms. Perhaps surprisingly, LOCKSTEP is consistently the slowest and by a large margin. On the other hand, COLOREDCHAIN was always considerably faster than LOCKSTEP, and consistently the fastest algorithm overall. The two exceptions are on the CBNs 5_param_g2a and 27_068, where FWDBWD finished first in 2s and 1032s (as opposed to 4s and 1114s for COLOREDCHAIN). On the other hand, FWDBWD was considerably slower than COLOREDCHAIN in some CBNs (e.g., 20_049). Although a wider experimental setting is required for conclusive results, our evaluation indicates that COLOREDCHAIN is very effective in handling CBNs.

6 Conclusion

We have introduced CHAIN, a new, truly symbolic, and time-optimal algorithm for SCC decomposition. The simplicity of CHAIN makes it theoretically elegant, while our experimental evaluation demonstrates a potential for practical impact. Some opportunities for future research include introducing saturation techniques [31] to CHAIN, as well as specializing it to the computation of bottom SCCs, which have received special attention [5].

Acknowledgements. This work was supported in part by Villum Fonden (Project VIL42117).

References

1. AEON models repository (2022), <https://github.com/sybila/biodivine-lib-par-am-bn/tree/lmcs>, Last accessed on 2022-10-01
2. Abraham, E., Jansen, N., Wimmer, R., Katoen, J.P., Becker, B.: DTMC model checking by SCC reduction. In: Proceedings of the 2010 Seventh International Conference on the Quantitative Evaluation of Systems. p. 37–46. QEST '10, IEEE Computer Society, USA (2010). <https://doi.org/10.1109/QEST.2010.13>
3. Amparore, E.G., Donatelli, S., Gallà, F.: starMC: an automata based CTL* model checker. *PeerJ Comput. Sci.* **8**, e823 (2022)
4. Barnat, J., Chaloupka, J., van de Pol, J.: Distributed algorithms for SCC decomposition. *J. Log. Comput.* **21**(1), 23–44 (2011)
5. Benes, N., Brim, L., Pastva, S., Safránek, D.: Computing bottom SCCs symbolically using transition guided reduction. In: Silva, A., Leino, K.R.M. (eds.) *Computer Aided Verification, CAV 2021, Part I*. LNCS, vol. 12759, pp. 505–528. Springer (2021). https://doi.org/10.1007/978-3-030-81685-8_24
6. Benes, N., Brim, L., Pastva, S., Safránek, D.: BDD-based algorithm for SCC decomposition of edge-coloured graphs. *Logical Methods in Computer Science* **18**(1) (2022). [https://doi.org/10.46298/lmcs-18\(1:38\)2022](https://doi.org/10.46298/lmcs-18(1:38)2022)
7. Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. *Formal Methods in System Design* **28**(1), 37–56 (2006)
8. Bloem, R., Ravi, K., Somenzi, F.: Efficient decision procedures for model checking of linear time logic properties. In: Proceedings of the 11th International Conference on Computer Aided Verification. p. 222–235. CAV '99, Springer (1999)
9. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.* **24**(3), 293–318 (1992)
10. Chaouiya, C., Naldi, A., Thieffry, D.: Logical modelling of gene regulatory networks with GINsim. *Bacterial Molecular Networks* p. 463–479 (2012). https://doi.org/10.1007/978-1-61779-361-5_23
11. Chatterjee, K., Dvořák, W., Henzinger, M., Loitzenbauer, V.: Lower bounds for symbolic computation on graphs: Strongly connected components, liveness, safety, and diameter. In: Proc. 29th ACM-SIAM Symp. on Discrete Algorithms. p. 2341–2356. SODA '18, Soc. for Industrial and Applied Mathematics, USA (2018)
12. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: Proc. 22nd ACM-SIAM Symp. on Discrete Algorithms. p. 1318–1336. SODA '11, Society for Industrial and Applied Mathematics, USA (2011)
13. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: CAV. LNCS, vol. 2404, pp. 359–364. Springer (2002)
14. van Dijk, T., van de Pol, J.: Sylvan: multi-core framework for decision diagrams. *Int. Journal on Software Tools for Technology Transfer* **19**(6), 675–696 (2017)
15. Dijkstra, E.W.: *A Discipline of Programming*. Prentice Hall PTR, USA, 1st edn. (1997)
16. Fisler, K., Fraer, R., Kamhi, G., Vardi, M.Y., Yang, Z.: Is there a best symbolic cycle-detection algorithm? In: Proc. 7th IC on Tools and Algorithms for the Construction and Analysis of Systems. p. 420–434. TACAS 2001, Springer (2001)

17. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. p. 573–582. SODA '03, Society for Industrial and Applied Mathematics, USA (2003)
18. Hardin, R.H., Kurshan, R.P., Shukla, S.K., Vardi, M.Y.: A new heuristic for bad cycle detection using BDDs. *Form. Methods Syst. Des.* **18**(2), 131–140 (mar 2001). <https://doi.org/10.1023/A:1008727508722>
19. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: High-performance language-independent model checking. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 692–707. Springer (2015)
20. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology* **22**(3), 437–67 (1969). [https://doi.org/10.1016/0022-5193\(69\)90015-0](https://doi.org/10.1016/0022-5193(69)90015-0)
21. Kesten, Y., Pnueli, A., Raviv, L., Shahar, E.: Model checking with strong fairness. *Formal Methods Syst. Des.* **28**(1), 57–84 (2006)
22. Kordon, F., Garavel, H., Hillah, L., Paviot-Adet, E., Jezequel, L., Hulin-Hubard, F., Amparore, E.G., Beccuti, M., Berthomieu, B., Evrard, H., Jensen, P.G., Botlan, D.L., Liebke, T., Meijer, J., Srba, J., Thierry-Mieg, Y., van de Pol, J., Wolf, K.: MCC'2017 - the seventh model checking contest. *Trans. Petri Nets Other Model. Concurr.* **13**, 181–209 (2018)
23. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011)
24. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: an open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.* **19**(1), 9–30 (2017)
25. Pelánek, R.: BEEM: benchmarks for explicit model checkers. In: SPIN. Lecture Notes in Computer Science, vol. 4595, pp. 263–267. Springer (2007)
26. Schwoon, S., Esparza, J.: A note on on-the-fly verification algorithms. In: TACAS. Lecture Notes in Computer Science, vol. 3440, pp. 174–190. Springer (2005)
27. Sharir, M.: A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications* **7**(1), 67–72 (1981). [https://doi.org/https://doi.org/10.1016/0898-1221\(81\)90008-0](https://doi.org/https://doi.org/10.1016/0898-1221(81)90008-0)
28. Tarjan, R.: Depth-first search and linear graph algorithms. *SIAM J. Comput.* **1**(2), 146–160 (jun 1972). <https://doi.org/10.1137/0201010>
29. Wang, C., Bloem, R., Hachtel, G.D., Ravi, K., Somenzi, F.: Divide and compose: SCC refinement for language emptiness. In: Proceedings of the 12th International Conference on Concurrency Theory. p. 456–471. CONCUR '01, Springer-Verlag, Berlin, Heidelberg (2001)
30. Xie, A., Beerel, P.A.: Implicit enumeration of strongly connected components and an application to formal verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **19**(10), 1225–1230 (2000)
31. Zhao, Y., Ciardo, G.: Symbolic computation of strongly connected components and fair cycles using saturation. *Innov. Syst. Softw. Eng.* **7**(2), 141–150 (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

