Symbolic Model Checking of Weighted PCTL Using Dependency Graphs

Mathias Claus Jensen, Anders Mariegaard(✉), and Kim Guldstrand Larsen

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg, Denmark
{mcje,am,kgl}@cs.aau.dk

Abstract. We present a global and local algorithm for model checking a weighted variant of PCTL with upper-bound weight constraints, on probabilistic weighted Kripke structures where the weights are vectors with non-zero magnitude. Both algorithms under- and over approximate a fixed-point over a symbolic dependency graph, until sufficient evidence to prove or disprove the given formula is found. Fixed-point computations are carried out in the domain of (multidimensional) probabilistic step functions, encoded as interval decision diagrams. The global algorithm works similarly to classic value iteration for PCTL in that it evaluates all nodes of the dependency graph iteratively, while the local algorithm performs a search-like evaluation of the given dependency graph in an attempt to find enough evidence locally to prove/disprove a given formula, without having to evaluate all nodes. Both algorithms are evaluated on several experiments and we show that the local algorithm generally outperforms the global algorithm.

Keywords: Model checking · PCTL · Fixed-point computation

1 Introduction

The ubiquity of embedded systems in modern-day society calls for robust and efficient methodologies for the design, production and implementation of more and more complex systems. These systems usually interact with the physical world, as well as the Internet, as a so called cyber-physical system. In this area, model-driven development is gaining popularity as a way to deal with early design-space exploration and automatic verification. Especially important in this context is the incorporation of non-functional aspects, such as resource consumption, timing constraints and probabilistic behavior. This has lead to a large variety of mathematical models having been created for the purpose of modeling these quantitative systems. In conjunction with these models, an assorted landscape of logics have also been proposed for the sake of specifying desired properties regarding the aforementioned models. Within the model-checking community this has lead to tools such as UPPAAL [16], PRISM [14], MRMC [12] and STORM [7], for analysis of systems involving continuous time,

J. M. Badger and K. Y. Rozier (Eds.): NFM 2019, LNCS 11460, pp. 298–315, 2019.
https://doi.org/10.1007/978-3-030-20652-9_20

stochastic behavior and various types of resources, in an efficient manner. At the heart of such tools are algorithms that verify user given properties on specified models.

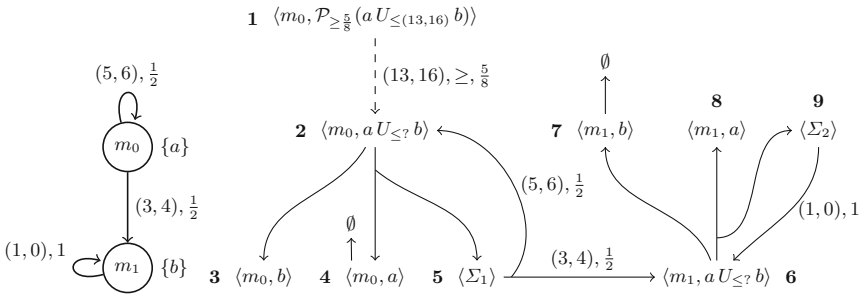
Our Contribution. We present two algorithms for model-checking a weighted subset of probabilistic CTL (PCTL) [8] with upper-bound weight constraints, on probabilistic weighted Kripke structures.

We allow for the weights of both the model and formula to be multidimensional, i.e. vectors. This allows for the modeling of consumption/production of resources in multiple dimensions. E.g. for cyber-physical systems we might be interested in both the time and energy it takes to perform some action.

Both algorithms approximate a fixed point on a *symbolic dependency graph* by repeated computation of under- and over-approximations. Termination is guaranteed as the transition weight vectors are required to have non-zero magnitude, in addition to path-formulae having upper-bound weight-constraints. Our symbolic dependency graphs extend the dependency graphs introduced by Liu and Smolka [17] to cope with the multidimensional probabilistic domain. The first algorithm, the global algorithm, is, with minor modifications, an instance of the approach presented in our previous work in [18] which in turn is an extension of the global algorithm by Liu and Smolka. The second algorithm, the local algorithm, was not a part of [18] and is our novel extension of the local algorithm presented by Liu and Smolka.

Both algorithms have been implemented in a prototype tool written in Python, using Interval Decision Diagrams [19] as the back-end data-structure for symbolic computations. For experimental evaluation, we present results on two case-studies based on PRISM [14] models and show that the local approach is, also in this domain, generally more efficient than the global approach, especially in cases where complete exploration of the underlying dependency graph is not needed to prove/disprove a property of the model. An extended version of the paper, with proofs, can be found online at http://people.cs.aau.dk/~am/nfm19/ext.pdf.

Related Work. The framework of fixed point computations introduced by Liu and Smolka has recently been extended in different ways. A distributed version of the local algorithm, that also deals with negation has been developed in [6]. The framework has also been extended to a weighted domain in [11] for model-checking weighted CTL on weighted Kripke structures where a symbolic graph encoding ensures an efficient local algorithm. An extension for Timed Games has been developed in [4]. In [5] the global algorithm was extended for parametric model-checking, used in [2] for model-checking on models with real-valued random variables as weights. Our global algorithm is reminiscent of the PCTL model-checking algorithm on which PRISM is based [13], in the sense that we consider the parse-tree of the formula and recursively compute the satisfaction of sub-formulae in an iterative manner. For MRMC, the algorithms based on *path graph generation* presented in [1] are used to solver similar model-checking problems, based on a local unfolding of the model as our local approach.



(a) $\mathcal{M}_1$ (b) SDG constructed from m_0 and $\Phi = \mathcal{P}_{\geq \frac{5}{8}}(a\, U_{\leq(13,16)}\, b)$

Fig. 1. A simple PWKS with 2-dimensional weights and its associated SDG

Each node in a path graph represents a certain reward, associated with a number of finite path fragments and their probabilities, in contrast to our approach where nodes encode probabilities associated with the satisfaction of a given formulae in a state.

2 Models and Properties

For any set X, X^n is the set of all n-dimensional vectors with elements from X. For $x \in X$ we let x^n denote the n-dimensional vector with all elements being x. Hence $\mathbb{N}^n$ is the set of all n-dimensional vectors of natural numbers and $\mathbb{N}_+^n = \mathbb{N}^n \setminus 0^n$ restricts $\mathbb{N}^n$ to vectors with strictly positive magnitude. For the remainder of the paper we assume a fixed dimensionality n, with $n > 0$. Any vector is written in boldface e.g. $\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{y} = (y_1, \ldots, y_n)$ are vectors. Finally, we assume a fixed finite set of labels AP.

Definition 1 (Probabilistic Weighted Kripke Structure). *A* Probabilistic Weighted Kripke Structure *(PWKS) is a structure $\mathcal{M} = (M, \rightarrow, \ell)$ where M is a finite set of states, $\rightarrow \subseteq M \times \mathbb{N}_+^n \times (0, 1] \times M$ is the finite weighted probabilistic transition relation such that for all $m \in M$, $\sum_{(m,w_i,p_i,m_i)\in\rightarrow} p_i = 1$ and $\ell\colon M \rightarrow 2^{AP}$ is the labeling function, assigning to each state a set of atomic propositions.*

Whenever $(m, \boldsymbol{w}, p, m') \in \rightarrow$ we write $m \xrightarrow{\boldsymbol{w},p} m'$. A *path* from a state m_0 is an infinite sequence of transitions $\pi = (m_0, \boldsymbol{w_0}, p_0, m_1), (m_1, \boldsymbol{w_1}, p_1, m_2), \ldots$ with $m_i \xrightarrow{\boldsymbol{w_i},p_i} m_{i+1}$ for any $i \in \mathbb{N}$. We denote by $\pi[j]$ the j'th state of π, m_j and by $\mathcal{W}(\pi)(j)$ the accumulated weight along path π up until m_j. Hence $\mathcal{W}(\pi)(0) = 0$ and $\mathcal{W}(\pi)(j) = \sum_{i=0}^{j-1} \boldsymbol{w_i}$ for $j > 0$. See Fig. 1a for an example PWKS with two states and weights from $\mathbb{N}^2$.

As specification language we define the logic Probabilistic Weighted CTL (PWCTL), extending a subset of Probabilistic CTL (PCTL), with weight-vectors.

Definition 2 (PWCTL). *The set of PWCTL state formulae, $\mathcal{L}$, is given by the following grammar:*

$$\mathcal{L}: \quad \Phi ::= a \mid \neg a \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \mathcal{P}_{\triangleright\lambda}(\Psi)$$

where $a \in AP, \lambda \in [0,1]$ and $\triangleright = \{>, \geq\}$. The path formulae *are given by the following grammar, with $\boldsymbol{k} \in \mathbb{N}^n$:*

$$\Psi ::= X_{\leq \boldsymbol{k}} \Phi \mid \Phi_1 U_{\leq \boldsymbol{k}} \Phi_2 \ .$$

We also define a set of symbolic unbounded until-formulae for later use, namely $\mathcal{S} = \{\Phi_1 U_{\leq ?} \Phi_2 \mid \Phi_1, \Phi_2 \in \mathcal{L}\} \cup \{X_{\geq ?} \Phi \mid \Phi \in \mathcal{L}\}$.

For the probabilistic modality $\mathcal{P}_{\triangleright\lambda}(\Psi)$, the satisfaction is dependent on the probability of picking a path satisfying the path-formulae Ψ, from some state m. To this end we employ the standard cylinder-set construction (see [3, Chapter 10]) to obtain a unique probability measure $\mathbb{P}$, assigning probabilities to sets of paths sharing a common prefix (a cylinder).

Definition 3 (PWCTL Semantics). *For a PWKS $\mathcal{M} = (M, \rightarrow, \ell)$ with state $m \in M$, the* satisfiability relation $\models$ *is inductively defined by:*

$$\begin{array}{lll}
\mathcal{M}, m \models a & \textit{iff} & a \in \ell(m) \\
\mathcal{M}, m \models \neg a & \textit{iff} & a \notin \ell(m) \\
\mathcal{M}, m \models \Phi_1 \wedge \Phi_2 & \textit{iff} & \mathcal{M}, m \models \Phi_1 \textit{ and } \mathcal{M}, m \models \Phi_2 \\
\mathcal{M}, m \models \Phi_1 \vee \Phi_2 & \textit{iff} & \mathcal{M}, m \models \Phi_1 \textit{ or } \mathcal{M}, m \models \Phi_2 \\
\mathcal{M}, m \models \mathcal{P}_{\triangleright\lambda}(\Psi) & \textit{iff} & \mathbb{P}(\pi \mid \pi[0] = m, \mathcal{M}, \pi \models \Psi) \triangleright \lambda
\end{array}$$

where, for any path π:

$$\begin{array}{lll}
\mathcal{M}, \pi \models X_{\leq \boldsymbol{k}} \Phi & \textit{iff} & \pi[0] \xrightarrow{\boldsymbol{w}, p} \pi[1], \mathcal{M}, \pi[1] \models \Phi, \textit{ and } \boldsymbol{w} \leq \boldsymbol{k} \\
\mathcal{M}, \pi \models \Phi_1 U_{\leq \boldsymbol{k}} \Phi_2 & \textit{iff} & \textit{there exists a } j \textit{ such that } \mathcal{M}, \pi[j] \models \Phi_2, \\
 & & \mathcal{M}, \pi[i] \models \Phi_1 \textit{ for all } i < j \textit{ and } \mathcal{W}(\pi)(j) \leq \boldsymbol{k}.
\end{array}$$

If $\mathcal{M}$ is clearly implied by the context, we simply write $m \models \Phi$ if the state m of PWKS $\mathcal{M}$ satisfies the formula Φ and similarly $\pi \models \Psi$ if π is a path in $\mathcal{M}$.

Example 1. For the PWKS $\mathcal{M}_1$ in Fig. 1a, we have that $m_0 \models \mathcal{P}_{\geq\lambda}(a\, U_{\leq \boldsymbol{k}}\, b)$ with $\boldsymbol{k} = (8, 10)$ and $\lambda = \frac{5}{8}$ as $\mathbb{P}(\pi \mid \pi[0] = m_0, \pi \models a\, U_{\leq (8,10)}\, b) = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{6}{8}$. In fact, this is the case for any $\boldsymbol{k} \geq (8, 10)$. Finally, if $\lambda \leq \frac{1}{2}$, considering only the path $m_0 \xrightarrow{(3,4), \frac{1}{2}} m_1 \cdots$ instead of the entire set of paths, would be sufficient.

3 Symbolic Dependency Graphs

As the semantics of PWCTL is given by induction in the structure of the formula, a solution to the model-checking problem $m \models \Phi$ is *dependent* on the solution to related model-checking problems involving sub-formulae of Φ and the reachable states of m. We encode these dependencies as edges between nodes in a *Symbolic Dependency Graph* and reduce the model-checking problem to fixed-point computations on these graphs.

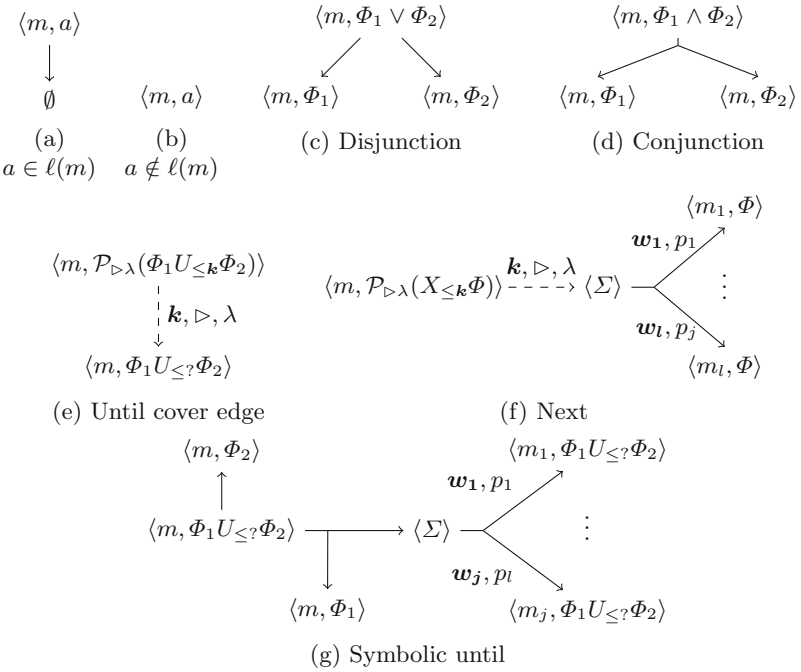


Fig. 2. SDG construction rules for state m where $m \xrightarrow{\boldsymbol{w}_i, p_i} m_i$ for all i with $1 \leq i \leq j$.

Definition 4 (Symbolic Dependency Graph). *For a PWKS* $\mathcal{M} = (M, \rightarrow, \ell)$*, a symbolic dependency graph (SDG) is a tuple,* $G = (C, E_H, E_C, E_\Sigma)$*, where*

- $C \subseteq M \times \mathcal{L} \cup M \times \mathcal{S} \cup \{\Sigma\} \times M \times \mathcal{S}$ *is a finite set of configurations (nodes),*
- $E_H \subseteq C \times 2^C$ *is a finite set of hyper-edges,*
- $E_C \subseteq C \times \mathbb{N}^n \times \triangleright \times [0,1] \times C$ *is a finite set of cover-edges, and*
- $E_\Sigma \subseteq C \times 2^{\mathbb{N}_+^n \times [0,1] \times C}$ *is a finite set of sum-edges.*

We will refer to elements of: $M \times \mathcal{L}$ as concrete-, $M \times \mathcal{S}$ as symbolic-, and $\{\Sigma\} \times M \times \mathcal{S}$ as sum-configurations. For brevity, we will often write $\langle \Sigma \rangle$ for sum-configurations when the state and symbolic formula is clear from context. If a configuration $s \in C$ can transition to another configuration $t \in C$ using any type of edge, we write $s \rightsquigarrow t$. Given a state m and formula Φ, one can construct the SDG rooted in $\langle m, \Phi \rangle$ by recursively applying the rules of Fig. 2. Singular *hyper-edges* are used to encode conjunction (Fig. 2d) and multiple *hyper-edges* encode disjunction (Fig. 2c). *Cover-edges* are used to abstract away concrete bounds on probabilities and weights and introduces symbolic configurations (Fig. 2e and f). Lastly, *sum-edges* encode the probabilistic weighted transitions of the underlying model (Fig. 2g and f).

Example 2. Consider again the PWKS $\mathcal{M}_1$ of Fig. 1a and the formula $\Phi = \mathcal{P}_{\geq \frac{5}{8}}(a\, U_{\leq (13,16)}\, b)$. The SDG obtained by applying the construction rules in Fig. 2 can be seen in Fig. 1b.

For the rest of this section we assume a fixed model, $\mathcal{M} = (M, \to, \ell)$, with $m \in M$ and a fixed PWCTL-formula, Φ. Let $G = (C, E_H, E_C, E_\Sigma)$ be the SDG constructed using the rules given in Fig. 2 with root $s_0 = \langle m, \Phi \rangle$. The semantics of configurations is given by assignments, encoding the probability of satisfaction, given a cost-bound (weight).

Definition 5 (Assignments). *An* assignment *is a function,* $a : \mathbb{N}^n \to [0,1]$*, assigning to each vector a probability.*

We use $\mathcal{A}$ *to denote the set of assignments.*

For any $a_1, a_2 \in \mathcal{A}$*,* $a_1 \sqsubseteq a_2$ *iff* $\forall \boldsymbol{w} \in \mathbb{N}^n.\, a_1(\boldsymbol{w}) \leq a_2(\boldsymbol{w})$*.*

Assignments naturally extends to SDGs.

Definition 6 (Assignment Mapping). *An* assignment mapping *on* G *is a function,* $A : C \to \mathcal{A}$*, mapping each configuration to an assignment.*

We use $\mathcal{C}_G$ *to denote the set of assignment mappings over* G*.*

For any $A_1, A_2 \in \mathcal{C}_G$*,* $A_1 \sqsubseteq_G A_2$ *iff* $\forall s \in C.\, A_1(s) \sqsubseteq A_2(s)$*.*

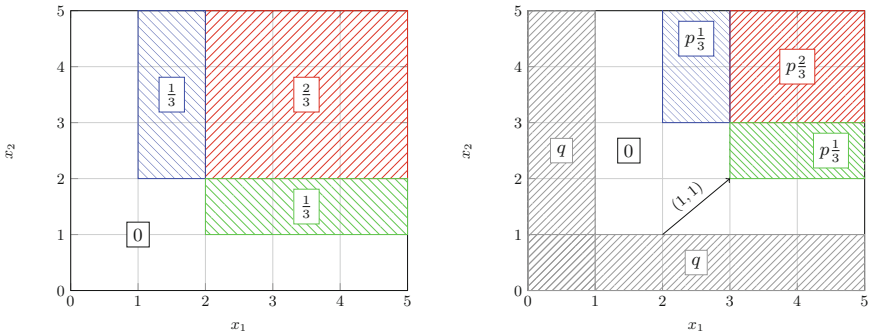
We define $\mathbb{0}, \mathbb{1} \in \mathcal{A}$ to be the assignments that map any vector to the probabilities 0 and 1, respectively. We will refer to these assignments as Boolean assignments. Similarly, we define $A^{\mathbb{0}}, A^{\mathbb{1}}$ to be the assignment mappings that map any configuration to the Boolean assignments $\mathbb{0}$ and $\mathbb{1}$, respectively. Generally, concrete configurations will receive Boolean assignments. For symbolic configurations, e.g. $\langle m, \Phi_1\, U_{\leq ?} \Phi_2 \rangle$, assignments will be used to compute probabilities associated with the sets of paths satisfying any concrete instance of the formula induced by replacing ? with a cost-bound $\boldsymbol{k} \in \mathbb{N}^n$.

Clearly, $(\mathcal{A}, \sqsubseteq)$ and $(\mathcal{C}_G, \sqsubseteq_G)$ are complete lattices with $\mathbb{0}$, $\mathbb{1}$ and $A^{\mathbb{0}}$, $A^{\mathbb{1}}$, as their respective bottom and top elements. We will use $\bigsqcup X$ and $\bigsqcap X$ to denote the supremum and infimum of any subset $X \subseteq \mathcal{A}$. As usual we let $\bigsqcup \emptyset = \mathbb{0}$ and $\bigsqcap \emptyset = \mathbb{1}$. The supremum of X can be realised as the assignment defined, for arbitrary $\boldsymbol{w} \in \mathbb{N}^n$, as $(\bigsqcup X)(\boldsymbol{w}) = \sup\{p \in [0,1] \mid a \in X, a(\boldsymbol{w}) = p\}$. The infimum can be realised in a similar fashion. For $a_1, a_2 \in \mathcal{A}$, we define $(a_1 + a_2)$ to be the assignment given, for arbitrary $\boldsymbol{w} \in \mathbb{N}^n$ as, $(a_1 + a_2)(\boldsymbol{w}) = a_1(\boldsymbol{w}) + a_2(\boldsymbol{w})$. Another useful operation on assignments as that of *shifting*:

Definition 7 (Shifting). *For* $\boldsymbol{w} \in \mathbb{N}^n$*, and* $p, q \in [0,1]$*,* $\mathtt{shift}_{\boldsymbol{w},p,q} : \mathcal{A} \to \mathcal{A}$ *is a function that, given an assignment* $a \in \mathcal{A}$*, produces a* shifted *assignment* $\mathtt{shift}_{\boldsymbol{w},p,q}(a) \in \mathcal{A}$*, defined for any* $\boldsymbol{v} \in \mathbb{N}^n$ *as,*

$$\mathtt{shift}_{\boldsymbol{w},p,q}(a)(\boldsymbol{v}) = \begin{cases} a(\boldsymbol{v} - \boldsymbol{w}) \cdot p & \text{if } \boldsymbol{w} \leq \boldsymbol{v} \\ q & \text{otherwise} \end{cases}$$

Shifting an assignment increases the cost of satisfaction, represented by the assignment, by an amount $\boldsymbol{w}$ while adjusting the degree of satisfaction by p and setting the probabilities to q if the cost is below $\boldsymbol{w}$.



(a) Assignment a^* as a plot (b) The assignment $\texttt{shift}_{(1,1),p,q}(a^*)$

Fig. 3. Assignment shifting

Example 3. Suppose $m \xrightarrow{(1,1),\frac{1}{2}} m_1$ and $m \xrightarrow{(2,2),\frac{1}{2}} m_2$. Let the assignment in Fig. 3a be a^* with the property $a^*(\boldsymbol{w}) = \mathbb{P}(\pi \mid \pi[0] = m_1, m_1 \models X_{\leq \boldsymbol{w}} \Phi) = \mathbb{P}(\pi \mid \pi[0] = m_2, m_2 \models X_{\leq \boldsymbol{w}} \Psi)$ for all $\boldsymbol{w} \in \mathbb{N}^n$ and some state-formula Φ. a^* thus encodes the exact probability of paths starting in m_1 (or m_2) that satisfy $X_{\leq \boldsymbol{w}} \Phi$, for all $\boldsymbol{w}$. By applying the shift operator, and addition on assignments, we get $a_m = \texttt{Shift}_{(1,1),\frac{1}{2},0}(a^*) + \texttt{Shift}_{(2,2),\frac{1}{2},0}(a^*)$. a_m has the desired property that $a_m(\boldsymbol{w}) = \mathbb{P}(\pi \mid \pi[0] = m, m \models X_{\leq \boldsymbol{w}} \Phi)$ for any $\boldsymbol{w}$. Figure 3b shows the result of the first term ($q = 0, p = \frac{1}{2}$).

We now introduce the fixed-point operator from our previous work in [18].

Definition 8. *For a SDG $G = (C, E_H, E_C, E_\Sigma)$, $F \colon \mathcal{C}_G \to \mathcal{C}_G$ is a function that, given an assignment mapping A on G, produces a new updated assignment mapping, $F(A)$. F is given for any node $s \in C$ as follows,*

$$F(A)(s) = \begin{cases} \begin{cases} \mathbb{1} & \text{if } A(t)(\boldsymbol{k}) \rhd \lambda \\ \mathbb{0} & \text{otherwise} \end{cases} & \text{if } (s, \boldsymbol{k}, \rhd, \lambda, t) \in E_C \\ \sum\limits_{(\boldsymbol{w},p,t) \in T} \texttt{shift}_{\boldsymbol{w},p,0}\,(A(t)) & \text{if } (s, T) \in E_\Sigma \\ \bigsqcup\limits_{(s,T) \in E_H} \prod\limits_{t \in T} A(t) & \text{otherwise} \end{cases}$$

F is well-defined as all configuration have at most one type of outgoing edge. For cover-edges we simply check the cover-condition. For sum-edges we compute a sum over all assignments to targets, shifted by the corresponding weight and probability as exemplified in Example 3. Lastly, for configurations with outgoing hyper-edges or no outgoing edges we compute a supremum over all hyper-edges and for each hyper-edge an infimum.

As F is monotonic (see [18]) on a complete lattice, we get, by Tarski's fixed point theorem [20], that F must have a unique least fixed point, $A_{\min}$. The following theorem states that our construction of a SDG from a pair $\langle m, \Phi \rangle$ along with its associated least fixed point, $A_{\min}$, as given by F, is indeed sound.

Theorem 1 (Soundness). $m \models \Phi$ *iff* $A_{\min}(\langle m, \Phi \rangle) = \mathbb{1}$.

Corollary 1. $\mathbb{P}(\pi \mid \pi[0] = m, \pi \models \Phi_1 U_{\leq \boldsymbol{k}} \Phi_2) = A_{\min}(\langle m, \Phi_1 U \Phi_2 \rangle)(\boldsymbol{k})$.

For any concrete configuration, i.e. on the form $\langle m, \Phi \rangle$, we have that any assignment mapping generated by F will be assigned either $\mathbb{0}$ or $\mathbb{1}$. Thus, we have the following corollary.

Corollary 2. $m \not\models \Phi$ *iff* $A_{\min}(\langle m, \Phi \rangle) = \mathbb{0}$.

As $(\mathcal{C}_G, \sqsubseteq_G)$ can be a lattice of infinite size, it is not given that we can construct $A_{\min}$ through repeated applications of F on the bottom element $\mathbb{0}$. The following theorem however states that F can be used to sufficiently approximate $A_{\min}$, from above and below, in a finite number of iterations, so that we may answer our model checking query.

Theorem 2 (Realisability). *There exists an* $i \in \mathbb{N}$ *such that,*

$$m \models \Phi \iff F^i(A^{\mathbb{0}})(\langle m, \Phi \rangle) = \mathbb{1} \text{ and } m \not\models \Phi \iff F^i(A^{\mathbb{1}})(\langle m, \Phi \rangle) = \mathbb{0}.$$

This theorem follows in part from our SDGs being finite. If the SDG is acyclic then it is trivial to show. If not, then the only cycles that occur, occur within the sub-tree of a node of type $\langle \mathcal{P}_{\rhd \lambda}(\Phi_1 U_{\leq \boldsymbol{k}} \Phi_2) \rangle$, which is directly dependent on $\langle m, \Phi_1 U_{\leq ?} \Phi_2 \rangle$. Since the weights of transitions are of positive magnitude, there is only a finite number of ways to concretely unfold the symbolic node $\langle m, \Phi_1 U_{\leq ?} \Phi_2 \rangle$ and its dependencies for any given $\boldsymbol{k} \in \mathbb{N}^n$. As such, there exists a $j \in \mathbb{N}$, such that $F^j(A^{\mathbb{0}})(\langle m, \Phi_1 U_{\leq ?} \Phi_2 \rangle)(\boldsymbol{k}) = F^j(A^{\mathbb{1}})(\langle m, \Phi_1 U_{\leq ?} \Phi_2 \rangle)(\boldsymbol{k}) = A_{\min}(\boldsymbol{k})$.

3.1 Global Algorithm

We now introduce an algorithm based on the function in Definition 8. This algorithm will be referred to as the global algorithm as it updates the entire assignment mapping of a given SDG each iteration, therefore in a sense, globally applying the iterator. The algorithm is as follows: repeatedly apply F on all configurations $s \in C$ until $F^i(\mathbb{0})(s_0) = \mathbb{1}$ or $F^j(\mathbb{1})(s_0) = \mathbb{0}$ for some $i, j \in \mathbb{N}$, where s_0 is the root of the SDG. Termination and correctness is guaranteed by Theorems 1 and 2.

Example 4. Consider the repeated application of F on the root of the SDG from Fig. 1b, starting from $\mathbb{0}$. Table 1 shows the results, with configurations in bold and one row per iteration. Only configurations that change value from $\mathbb{0}$ are listed. Assignments are written as pairs of weights and probabilities e.g. $\{((3,4), \frac{1}{2}), ((8,10), \frac{3}{4})\}$ is the assignment a s.t $a(\boldsymbol{w}) = 0$ for $\boldsymbol{w} < (3,4)$, $a(\boldsymbol{w}) = \frac{1}{2}$ for $(3,4) \leq \boldsymbol{w} < (8,10)$ and $a(\boldsymbol{w}) = \frac{3}{4}$ for $\boldsymbol{w} \geq (8,10)$. As seen, F^7 assigns $\mathbb{1}$ to $\mathbf{1}$ i.e $m_0 \models \mathcal{P}_{\geq \frac{5}{8}}(a\, U_{\leq (8,10)}\, b)$ as expected.

Table 1. Lower bound assignments for Fig. 1b

	1	**2**	**4**	**5**	**6**	**7**	**9**
1	-	-	$\mathbb{1}$	-	-	$\mathbb{1}$	-
2	-	-	-	-	$\mathbb{1}$	-	-
3	-	-	-	$\{((3,4),\frac{1}{2})\}$	-	-	$\{((1,0),1)\}$
4	-	$\{((3,4),\frac{1}{2})\}$	-	-	-	-	-
5	-	-	-	$\{((3,4),\frac{1}{2}),((8,10),\frac{3}{4})\}$	-	-	-
6	-	$\{((3,4),\frac{1}{2}),((8,10),\frac{3}{4})\}$	-	-	-	-	-
7	$\mathbb{1}$	-	-	-	-	-	-

In practice, we need a finite representation of assignments. To this end we use Interval Decision Diagrams (IDDs) [19], a generalization of Binary Decision Diagrams (BDDs). IDDs, like BDDs, test on variables but now the values of variables are partitioned into disjoint intervals that must be independent, in the sense that any two values within the same interval must produce the same function value.

Recall that all assignments are of the form $\alpha : \mathbb{N}^n \to [0,1]$ i.e. functions that, given a vector of natural numbers, yields some probability. The simplest assignments, $\mathbb{0}$ and $\mathbb{1}$ are encoded directly as IDD terminals. In the general case, we have n variables, where n is the dimension of the weight-vectors. As all assignments of interest are built from $\mathbb{0}$ and $\mathbb{1}$ by applying operations $\mathtt{shift}_{\boldsymbol{w},p,q}(a)$, $\bigsqcup\{a,b\}$, $\bigsqcap\{a,b\}$, and $a+b$, where $a,b \in \mathcal{A}$, we only need that IDDs are closed under these operations. For the binary operations, this is a straight-forward extension of the procedure for BDDs and it is easy to show that IDDs are closed under shifting.

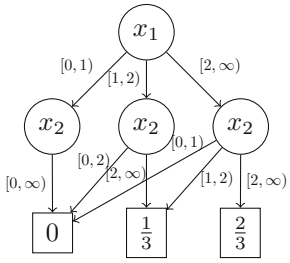


Fig. 4. Example IDD

Example 5. As an example of an IDD encoding a non-trivial assignment, see Fig. 4, here encoding the assignment $a*$ of Fig. 3a. a_* can be generated by the assignment $\mathtt{shift}_{(1,2),\frac{1}{3},0}(\mathbb{1}) + \mathtt{shift}_{(2,1),\frac{1}{3},0}(\mathbb{1})$.

4 Local Algorithm

As an alternative to globally updating the assignment to all nodes in each iteration, we propose a *local* algorithm. The pseudocode for the local algorithm can be seen in Algorithm 1. It takes as input a SDG $G = (C, E_H, E_C, E_\Sigma)$ and a configuration $s_0 = \langle m, \Phi \rangle$ and outputs $A_{\min}(s_0)$. The fixed-point computation is done in the while loop (lines 15–17) by calling `nextlower` and `nextupper` to compute the next lower and upper bound of $A_{\min}$, respectively. To this end, single edges are processed locally according to the type of the edge. Assignments to configurations are now from the domain $\mathcal{A} \cup \{\bot, \top\}$, $\bot(\top)$ being the smallest

(largest) assignment w.r.t $\sqsubseteq$. The SDG G is assumed to be constructed according to the rules in Fig. 2 for a model-checking problem $m \models \Phi$. For termination, we define a required precision $\boldsymbol{k} = \sup K(\Phi)$ on assignments, as the supremum of all the weight-bounds found in Φ. Given $\boldsymbol{k}$, we are only interested in assignment with this precision. We therefore introduce a $\boldsymbol{k}$-ordering of assignments.

Definition 9 ($\boldsymbol{k}$-ordering). *For a given $\boldsymbol{k} \in \mathbb{N}^n$, we define the binary relation $\sqsubseteq_{\boldsymbol{k}}$ on assignments by $a_1 \sqsubseteq_{\boldsymbol{k}} a_2$ iff $\forall \boldsymbol{w} \leq \boldsymbol{k} \;.\; a_1(\boldsymbol{w}) \leq a_2(\boldsymbol{w})$, where $a_1, a_2 \in \mathcal{A}$ and let $a_1 =_{\boldsymbol{k}} a_2$ if $a_1 \sqsubseteq_{\boldsymbol{k}} a_2$ and $a_2 \sqsubseteq_{\boldsymbol{k}} a_1$. For a given SDG $G = (C, E_H, E_C, E_\Sigma)$, we extend the relation to assignment mappings by $A_1 \sqsubseteq_{\boldsymbol{k}} A_2$ iff $\forall s \in C \;.\; A_1(s) \sqsubseteq_{\boldsymbol{k}} A_2(s)$, where $A_1, A_2 \in \mathcal{C}_G$.*

Algorithm 1. Symbolic Local Algorithm

input : SDG $G = (C, E_H, E_C, E_\Sigma)$ and configuration $s_0 = \langle m, \Phi \rangle \in C$
output : $A_{\min}(s_0)$

1 $\boldsymbol{k} = \sup K(\Phi)$;
2 **foreach** $v \in \{L, U\}$ **do**
3 **foreach** $s \in C$ **do** $R^v(s) = \emptyset$;
4 **foreach** $(s, T) \in E_\Sigma$ **do**
5 **foreach** $(\boldsymbol{w}, p, t) \in T$ **do** $\Sigma^v_\Delta(s)(t) = \mathbb{0}$;
6 **foreach** $s \in C$ **do**
7 $A^L(s) = \bot$;
8 $A^U(s) = \top$;
9 $A^L(s_0) = a^L_{def} = a^U_{max} = \mathbb{0}$;
10 $A^U(s_0) = a^U_{def} = a^L_{max} = \mathbb{1}$;
11 $W^\downarrow_L = W^\downarrow_U =$ `succ`(s_0);
12 $W^\uparrow_L = W^\uparrow_U = \emptyset$;
13 **while** $(W^\downarrow_L \cup W^\uparrow_L) \neq \emptyset \wedge (W^\downarrow_U \cup W^\uparrow_U) \neq \emptyset$ **do**
14 `nextlower()`;
15 `nextupper()`;
16 **if** $(W^\downarrow_L \cup W^\uparrow_L) = \emptyset$ **then return** $A^L(s_0)$;
17 **else return** $A^U(s_0)$;

Given $\boldsymbol{k}$, termination and correctness does not rely on computing both the upper and lower bound and as the two functions are almost identical, we only show pseudo-code responsible for computing lower bounds. In practice, computing upper bounds can be beneficial in cases where the query is unsatisfied. The pseudo-code for `nextlower` can be seen in Algorithm 2, with edge processing functions in Algorithm 4.

Functions `nextlower` and `nextupper` utilize different data-structures to process edges of the graph. Let $\alpha \in \{L, U\}$. Then the data-structures are: $Crt\colon C \rightharpoonup \mathcal{A}, A^L\colon C \to \mathcal{A} \cup \{\bot\}, A^U\colon C \to \mathcal{A} \cup \{\top\}, W_\alpha^\downarrow, W_\alpha^\uparrow \subseteq 2^E, \Sigma_\Delta^\alpha\colon C \to (C \to \mathcal{A})$, $R^\alpha\colon C \to 2^C$ and $a_{def}^\alpha, a_{max}^\alpha \in \mathcal{A}$.

All data-structures with $\alpha = L$ ($\alpha = U$) are only used in `nextlower` (`nextupper`) Crt is a partial function with $Crt(s) = a$ if s has received its fixed-point assignment a. Crt is used to skip the processing edges for which the source configuration has already received its fixed point assignment. A^L and A^U contain the current approximations of the fixed point. If $A^L(s) = \bot$ ($A^U(s) = \top$) then configuration s has no under-approximation (over-approximation) yet. $W_\alpha^\downarrow$ and $W_\alpha^\uparrow$ are sets containing all edges to be processed for exploration and back-propagation, respectively. For a sum-edge (s, T) with $T = \{(\boldsymbol{w_1}, p_1, t_1), \ldots (\boldsymbol{w_j}, p_j, t_j)\}$, $\Sigma_\Delta^\alpha(s)(t_i)$ contains the contribution from the partial sum-edge $(s, \boldsymbol{w_i}, p_i, t_i)$ to the assignment of s. For two configuration $s, t \in C$, $s \in R^\alpha(t)$ indicates that the assignment to s, $A^\alpha(s)$ is dependent on the assignment to t, $A^\alpha(t)$ and that $A^\alpha(t)$ was changed since the last update to $A^\alpha(s)$. When processing an edge e with s as the source (`source`(e) = s), we can thus safely skip any $t \in$ `targets`(e) for which $s \notin R^\alpha(t)$, when updating $A^\alpha(s)$. We will refer to $R^\alpha(t)$ as the *read list* of t. a_{def}^α is the default assignment given to newly discovered configurations and a_{max}^α is the maximal possible assignment any configuration can get. All data structures are initialized in Algorithm 1 and are used throughout Algorithms 1–3. For any edge e, we let `source`(e) be its source configuration and `targets`$(e) = \{t \mid$ `source(`e`)` $\rightsquigarrow t\}$ be its set of targets. For any configuration s we let `succ`$(s) = \{(s, T) \in E_H \cup E_\Sigma\} \cup \{(s, \boldsymbol{w}, \triangleright, p, t) \in E_C\}$ be the set of edge-successors of s and `D`$(s) = \{e \mid e \in E_H \cup E_\Sigma \cup E_C \wedge s \in$ `targets`$(e)\}$ the set of edges dependent on the assignment to s. Informally, if the assignment to s is changed, edges from `D`(s) should be processed.

Algorithm 2. `nextlower`

```
1  function nextlower():
2      Pop e from W_L^↓ or W_L^↑;
3      s = source(e);
4      if Crt(s) = a then
5          A^L(s) = a;
6          W_L^↑ = W_L^↑ ∪ D(s);
7      else if e ∈ E_H then
8          HyperEdgeLower(e);
9      else if e ∈ E_Σ then
10         SumEdgeLower(e);
11     else if e ∈ E_C then
12         CoverEdgeLower(e);
```

For edge processing, we utilize helper functions. Algorithm 3 shows the pseudo-code. We use `InitLower`(t) when a new target t is discovered. The assignment of t is set to the specified default value a^L_{def} and a forward exploration is prepared from t by adding all successors of t to $W^{\downarrow}_L$. Finally, t is added to all relevant read lists. `BackPropLower`(s, a_{new}) is used when updating assignment $A^L(s)$ to a_{new}. If it cannot be further improved, $Crt(s) = a_{new}$. Back-propagation is prepared by adding all dependent edges, `D`(s), to $W^{\uparrow}_L$. If the source of any such edge has not been discovered and therefore has assignment $\bot$, it is initialized. Finally, for any edge in `D`(s), the newly updated assignment should be read. Hence the read list of s, $R^L(s)$ is updated to include the sources of all such edges.

Algorithm 3. Helper functions

```
1  function InitLower(s):
2      A^L(s) = a^L_def;
3      W^↓_L = W^↓_L ∪ succ(s);
4      foreach e ∈ succ(s) do
5          foreach t ∈ targets(e) do
6              if A^L(t) ≠ ⊥ then
7                  R^L(t) = R^L(t) ∪ {s};
8  function BackPropLower(s, a_new):
9      foreach e ∈ D(s) do
10         if A^L(source(e)) = ⊥ then
11             InitLower(source(e));
12         R^L(s) = R^L(s) ∪ {source(e)};
13         W^↑_L = W^↑_L ∪ {e};
14     if a_new = a^L_max then
15         Crt(s) = a^L_max;
```

We now present our termination and correctness theorems, saying that the while loop in Algorithm 1 terminates and that the computed assignments for explored nodes are equal to the assignment given by the minimal fixed-point $A_{\min}$, within the given precision $\boldsymbol{k}$.

Lemma 1 (Termination). *The local algorithm (Algorithm 1) terminates with*

$$(W^{\downarrow}_L \cup W^{\uparrow}_L) = \emptyset \text{ or } (W^{\downarrow}_U \cup W^{\uparrow}_U) = \emptyset.$$

Theorem 3 (Correctness). *Upon termination, the local algorithm has computed assignments A^L, A^U, such that for any $s \in C$,*

- *If $(W^{\downarrow}_L \cup W^{\uparrow}_L) = \emptyset$ then $A^L(s) \neq \bot \implies A^L(s) =_{\boldsymbol{k}} A_{\min}(s)$.*
- *If $(W^{\downarrow}_U \cup W^{\uparrow}_U) = \emptyset$ then $A^U(s) \neq \top \implies A^U(s) =_{\boldsymbol{k}} A_{\min}(s)$.*

5 Experiments

Both the local and global algorithm have been implemented in a prototype tool written in Python. For both algorithms, we use two separate processes to compute the under- and over-approximations in parallel. For the local algorithm, successors of configurations are generated on-the-fly. Both algorithms terminate when the root configuration is fixed. For the local algorithm we may also terminate when the waiting lists of either the under- or over-approximator are empty.

Algorithm 4. Processing functions for `nextlower`

```
1  function HyperEdgeLower(e = (s, T)):
2      if ∃t ∈ T.A^L(t) = 𝕆 then return ;
3      else if ∃t ∈ T.A(t)^L = ⊥ then InitLower(t) ;
4      else
5          a = ⨆{⨅{A^L(t) | t ∈ T}, A^L(s)};
6          if a ⊒_k A^L(s) then
7              BackPropLower(s, a);
8              A^L(s) = a;
9  function SumEdgeLower(e = (s, T)):
10     a = A^L(s);
11     foreach (w, p, t) ∈ T do
12         if A^L(t) ≠ ⊥ ∧ s ∈ R^L(t) then
13             R^L(t) = R^L(t) \ {s};
14             Δ^new = shift_{w,p,0}(A^L(t));
15             Δ^old = Σ^L_Δ(s)(t);
16             if Δ^new ≠_k Δ^old then
17                 a = a + (Δ^new − Δ^old);
18                 Σ^L_Δ(s)(t) = Δ^new;
19     if a ⊒_k A^L(s) then
20         BackPropLower(s, a);
21         A^L(s) = a;
22     if ∃(w, p, t) ∈ T.A^L(t) = ⊥ then InitLower(t) ;
23     if ∃(w, p, t) ∈ T.A^L(t) = ⊥ then W^↓_L = W^↓_L ∪ {e} ;
24 function CoverEdgeLower(e = (s, k, ▷, λ, t)):
25     if A^L(t) = ⊥ then InitLower(t) ;
26     if A^L(t)(k) ▷ λ and A^L(s) ≠ 𝟙 then
27         BackPropLower(s, 𝟙);
28         A^L(s) = 𝟙;
```

For experimental evaluation, our prototype tool supports DTMC models (with transition rewards/costs), written in the PRISM language [14]. PRISM cannot directly handle our models with weights from $\mathbb{N}^n$ with $n > 1$. To this end, we interpret multiple reward structures as defining a vector in n dimensions, with n being the number of reward structures. Hence one can define a proper PWKS in the PRISM language and use it as input to our tool.

We run both the global and local algorithm on two PRISM models (synchronous leader election [10] and the bounded retransmission protocol [9]), derived from DTMC models of the PRISM benchmark suite [15]. All models can be found online at http://people.cs.aau.dk/~am/nfm19/code/prism_ex/.

For all models, the model-checking query is an instance of cost-bounded probabilistic reachability: $m \models \mathcal{P}_{\triangleright\lambda}(tt\, U_{\leq \boldsymbol{k}}\, prop)$, where m is the initial state of the underlying DTMC and *prop* is a label assigned to all states satisfying the property of interest. Our tool invokes PRISM on a given model and exports the underlying DTMC (with transition rewards), which our tool then parses to construct a PWKS. For each instantiation of a PRISM model we run four queries with a fixed cost-bound $\boldsymbol{k} \in \mathbb{N}^n$ where n is the arity. These four queries differ in the comparison of probabilities in the formula. For comparator and probability we use the following four configurations: $> p + \frac{1-p}{10}$, $> p - \frac{p}{10}$, > 0, and ≥ 1; where p is the exact probability of picking a path from state m, that satisfies the given until-formula. The expressions > 0 and ≥ 1 encode existential and universal quantification, respectively.

5.1 Results

We evaluate different hyper-parameters on the case-studies consisting of data-structures for the waiting lists and the weights associated with preferring forward exploration to back-propagation. For waiting lists we experimented with *queues* (Q), *stacks* (S), and *counters* (C), where a counter is a priority queue with priority given by the number of times an element is added. We weigh the decision of either forward exploring or back-propagating with integers weights in $[1, 5]$.

For what follows, we present only the results of experiments involving the global algorithm and the local algorithm with the best hyper-parameters. We will use weighted tuples of data-structures to indicate the hyper-parameters of the local algorithm, e.g. (Q1,S3) would indicate that we used the local algorithm with a *queue* for forward exploration, a *stack* for backwards propagation, and that when given the choice, we are 3 times more likely to back-propagate than forward explore. Additionally, we will use GMC to indicate the use of the global algorithm. All experiments were run using 2 cores of an AMD Opteron 6376 processor allowing for parallelism.

Synchronous Leader Election. Table 2 shows our results for synchronous leader election protocol. The data is an average over the run times using 1-, 2-, and 3-dimensional weights. We find that the local algorithm outperforms the global one, with average speedups of around 6 when answering the non-existential or -universal queries. For the existential and universal queries we find that the local algorithm is on average 400 times faster. In Table 3 we compare the relative speedup across 1-, 2-, and 3-dimensional weights.

Table 2. Results for synchronous leader election.

Leader election, N = number of processes, K = number of probabilistic choices									
N	K	$> p + \frac{1-p}{10}$		$> p - \frac{p}{10}$		> 0		≥ 1	
		GMC	Q5,Q1	GMC	C5,Q1	GMC	Q1,C3	GMC	Q1,C3
4	3	10.59	1.71	12.41	2.32	6.01	0.09	1.60	0.11
	4	39.98	9.49	47.96	6.33	17.51	0.13	42.22	0.23
	5	45.25	11.70	101.87	13.95	46.96	0.13	104.22	0.15
	6	102.77	20.40	264.65	46.69	118.33	0.29	275.88	0.28
	8	259.07	121.42	657.20	200.43	354.57	0.94	752.16	0.83
5	2	7.80	0.91	9.60	0.93	3.92	0.09	10.53	0.17
	3	44.56	9.13	57.28	9.59	22.89	0.13	60.59	0.16
	4	227.02	25.64	265.09	34.36	117.54	0.35	264.75	0.36
	5	385.77	110.85	958.42	179.50	394.71	0.67	93.05	0.67
	6	921.52	395.49	2040.88	433.37	1182.46	1.71	283.81	1.91
6	2	21.45	2.29	29.47	2.68	10.31	0.08	30.68	0.17
	3	199.64	73.93	247.00	71.63	100.63	0.30	23.18	0.36
	4	1670.63	315.67	1742.46	346.03	724.10	0.89	2008.89	0.88

Table 3. Average relative speedup for synchronous leader election per arity.

Leader election				
Arity	GMC/(Q5,Q1)		GMC/(Q1,C3)	
	$> p + \frac{1-p}{10}$	$> p - \frac{p}{10}$	> 0	≥ 1
1	4.65	6.34	112.71	144.92
2	5.40	7.69	230.29	343.69
3	6.09	9.21	364.82	503.89

Bounded Retransmission Protocol. Table 4 shows our results for the bounded retransmission protocol. All data is for 1-dimensional weights. Again, we find that the local algorithm outperforms the global. In the unsatisfied and satisfied case we see speedups averaging around 25 and 30, respectively. For the existential queries we find the local algorithm to be, on average, 850 times faster than the global. In the universal case we see only a speedup of about 30.

Table 4. Results for the bounded retransmission protocol.

BRP, M = max number of retransmissions, N = number of chunks									
N	M	$> p + \frac{1-p}{10}$		$> p - \frac{p}{10}$		> 0		≥ 1	
		GMC	S1,S3	GMC	S1,S3	GMC	S1,Q5	GMC	S1,S3
16	2	191.20	11.93	229.28	6.94	8.02	0.03	227.62	6.75
	3	253.99	16.80	313.45	9.65	13.15	0.03	315.30	9.64
	4	327.73	20.65	395.51	11.37	18.78	0.03	396.06	11.34
	5	503.98	23.59	621.86	13.72	33.50	0.04	593.56	13.57
32	2	420.88	27.26	507.14	14.71	16.92	0.03	506.48	14.58
	3	522.58	37.05	650.74	19.59	26.10	0.03	649.89	19.32
	4	864.38	49.78	660.95	24.41	49.91	0.04	669.76	24.26
	5	810.14	52.34	959.49	28.76	52.58	0.05	958.92	28.87
62	2	797.97	54.61	961.92	28.87	31.13	0.04	961.50	28.67
	3	1051.64	75.64	1319.02	38.43	51.60	0.05	**n/a**	37.96
	4	1339.32	92.68	1631.08	47.91	75.77	0.05	**n/a**	47.50
	5	1610.09	108.73	1923.06	57.34	102.77	0.06	**n/a**	57.18

6 Conclusion

We have presented two approaches for model-checking a variant of PCTL, interpreted over probabilistic weighted Kripke structures. We introduce a reduction to fixed-point computations on symbolic dependency graphs where nodes represent model-checking problems and edges explicitly encode dependencies among said problems. The first approach, the global algorithm, is a minor extension of the algorithm presented in [18] which iteratively computes an update to each node of the entire graph. The second approach, the local algorithm, is a novel adaptation of existing dependency graph algorithms, to our probabilistic weighted domain. The algorithm performs a local search-like exploration of the graph and lends itself to an on-the-fly unfolding. Both algorithms were implemented in a prototype tool, using Interval Decision Diagrams (IDDs) as the back-end data-structure. It is shown that the local algorithm generally outperforms the global algorithm, especially in cases where a complete exploration of the model is not needed to prove or disprove a property of the model. Our work could be extended to incorporate negation in the logic as shown in [6].

Future work includes investigating clever memoization schemes to deal with the expensive IDD operations, as has been previously done for BDDs. Preliminary experiments by the authors with a naïve caching mechanism has shown that it provides a significant speed-up, especially for the global algorithm. A process calculus is another direction that could be promising as our local approach lends itself to a local-unfolding of the model, instead of an up-front construction of the entire state-space. Lastly, more research is required to develop better search strategies such that the local algorithm more robustly can efficiently solve most queries.

References

1. Andova, S., Hermanns, H., Katoen, J.-P.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 88–104. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-40903-8_8
2. Bacci, G., Hansen, M., Larsen, K.G.: On the verification of weighted kripke structures under uncertainty. In: McIver, A., Horvath, A. (eds.) QEST 2018. LNCS, vol. 11024, pp. 71–86. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99154-2_5
3. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005). https://doi.org/10.1007/11539452_9
5. Christoffersen, P., Hansen, M., Mariegaard, A., Ringsmose, J.T., Larsen, K.G., Mardare, R.: Parametric verification of weighted systems. In: 2nd International Workshop on Synthesis of Complex Parameters, SynCoP 2015, London, United Kingdom, 11 April 2015, pp. 77–90 (2015). https://doi.org/10.4230/OASIcs.SynCoP.2015.77
6. Dalsgaard, A.E., et al.: A distributed fixed-point algorithm for extended dependency graphs. Fundam. Inform. **161**(4), 351–381 (2018). https://doi.org/10.3233/FI-2018-1707
7. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017, Part II. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
8. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Asp. Comput. **6**(5), 512–535 (1994). https://doi.org/10.1007/BF01211866
9. Helmink, L., Sellink, M.P.A., Vaandrager, F.W.: Proof-checking a data link protocol. In: Barendregt, H., Nipkow, T. (eds.) TYPES 1993. LNCS, vol. 806, pp. 127–165. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58085-9_75
10. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Inf. Comput. **88**(1), 60–87 (1990). https://doi.org/10.1016/0890-5401(90)90004-2
11. Jensen, J.F., Larsen, K.G., Srba, J., Oestergaard, L.K.: Efficient model-checking of weighted CTL with upper-bound constraints. STTT **18**(4), 409–426 (2016). https://doi.org/10.1007/s10009-014-0359-5
12. Katoen, J., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), Torino, Italy, 19–22 September 2005, pp. 243–244 (2005). https://doi.org/10.1109/QEST.2005.2
13. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72522-0_6
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
15. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, 17–20 September 2012, pp. 203–204 (2012). https://doi.org/10.1109/QEST.2012.14

16. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT **1**(1–2), 134–152 (1997). https://doi.org/10.1007/s100090050010
17. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 53–66. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055040
18. Mariegaard, A., Larsen, K.G.: Symbolic dependency graphs for PCTL$^{\geq}_{\leq}$ model-checking. In: Abate, A., Geeraerts, G. (eds.) FORMATS 2017. LNCS, vol. 10419, pp. 153–169. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65765-3_9
19. Strehl, K., Thiele, L.: Symbolic model checking of process networks using interval diagram techniques. In: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1998, San Jose, CA, USA, 8–12 November 1998, pp. 686–692 (1998). https://doi.org/10.1145/288548.289117
20. Tarski, A., et al.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**(2), 285–309 (1955)