Flow Faster: Efficient Decision Algorithms for Probabilistic Simulations[★]

Lijun Zhang¹, Holger Hermanns¹, Friedrich Eisenbrand²,
and David N. Jansen^{3,4}

¹ Department of Computer Science, Saarland University, Saarbrücken, Germany

² Department of Mathematics, University of Paderborn, Germany

³ Department of Computer Science, University of Twente, Enschede, The Netherlands

⁴ Software Modeling and Verification Group, RWTH Aachen, Germany

Abstract. Abstraction techniques based on simulation relations have become an important and effective proof technique to avoid the infamous state space explosion problem. In the context of Markov chains, strong and weak simulation relations have been proposed [17,6], together with corresponding decision algorithms [3,5], but it is as yet unclear whether they can be used as effectively as their non-stochastic counterparts. This paper presents drastically improved algorithms to decide whether one (discrete- or continuous-time) Markov chain strongly or weakly simulates another. The key innovation is the use of parametric maximum flow techniques to amortize computations.

1 Introduction

To compare the stepwise behaviour of states in transition systems, simulation relations ($\preceq$) have been widely considered [18,16]. Simulation relations are preorders on the state space such that if $s \preceq s'$ (“ s' simulates s ”) state s' can mimic all stepwise behaviour of s ; the converse, i. e., $s' \preceq s$ is not guaranteed, so state s' may perform steps that cannot be matched by s . Thus, if $s \preceq s'$ then every successor of s has a corresponding related successor of s' , but the reverse does not necessarily hold. In the context of model checking, simulation relations can be used to combat the well-known state space explosion problem, owed to the preservation of certain classes of temporal formulas. For instance, if $s \preceq s'$ then for all *safe* CTL* formulas Φ (formulas with universal path-quantifiers only) it follows that $s' \models \Phi$ implies $s \models \Phi$ [9].

Verification of stochastic systems faces very similar state space explosion problems. Therefore, simulation preorders [17,6] have been proposed for discrete- and continuous-time Markov chains (DTMCs and CTMCs). In correspondence to the non-probabilistic setting, these preorders preserve fragments of PCTL [14] and CSL [2,4]. They provide the principal ingredients to perform abstraction of Markov chains, while preserving *safe* fragments of the respective logics. However,

[★] This work is supported by the NWO-DFG bilateral project VOSS and by the DFG as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS.

it is as yet unclear whether these relations can be used with similar effectiveness as in the non-probabilistic setting. One prerequisite is the availability of efficient decision procedures for simulation in finite-state models.

Let n denote the number of states, and m denote the number of transitions. For strong simulation preorder, Baier *et al.* [3] introduced a polynomial decision algorithm with complexity $\mathcal{O}(n^7/\log n)$, by tailoring a network flow algorithm to the problem, embedded into an iterative refinement loop. This technique can not be applied to *weak* simulations [7] directly. In [5], Baier *et al.* proved that probabilistic weak simulation is decidable in polynomial time by reducing it to a linear programming (LP) problem.

In this paper, we present drastically improved algorithms. For strong simulation, the core observation is that the networks on which the maximum flows are calculated, are very similar across iterations of the refinement loop. We exploit this by adaptation of the parametric maximum flow algorithm [10] to solve the maximum flows for the arising sequences of similar networks, arriving at an overall time complexity $\mathcal{O}(m^2n)$.

For weak simulation, adapting the maximum flow idea is not straightforward. This is because successor states might need to be *split* into two fragments and one does not a priori know how to split them. Nevertheless, we manage to incorporate the parametric maximum flow algorithm into a decision algorithm with complexity $\mathcal{O}(m^2n^3)$.

The algorithms are developed for both discrete- and continuous-time Markov chains. Especially in the very common case, where the state fanout of a model is bounded by a constant k (and hence $m \leq kn$), our strong simulation algorithm has complexity $\mathcal{O}(n^2)$ which is faster by a factor of $n^5/\log n$ in comparison to the existing algorithm. This complexity corresponds to the best algorithms for deciding strong simulation over non-probabilistic systems [15,11]. As we will discuss the weak simulation algorithm even leads to an improvement in the order of n^{10} for CTMCs (and n^9 for DTMCs), compared to the one using a polynomial LP routine [20]. Remarkably, our algorithm is polynomial in the RAM-model of computation while no known LP-based algorithm is. We argue that especially for CTMCs, which have a very broad spectrum of applications ranging from disk storage dimensioning to gene regulatory networks, the availability of such algorithms can become a key ingredient to fight the state space explosion problem.

The paper proceeds by first giving necessary definitions and background in Section 2. Section 3 presents algorithms for deciding strong simulations while Section 4 focuses on algorithms for weak simulations. Section 5 concludes the paper.

2 Preliminaries

This section recalls the definitions of fully probabilistic systems, discrete- and continuous-time Markov chains, strong and weak simulations on these models [7]. We also review the preflow algorithm to solve maximum flow problems [13].

Models. Let X, Y be finite sets. For $f : X \rightarrow \mathbb{R}$, let $f(A)$ denote $\sum_{x \in A} f(x)$ for all $A \subseteq X$. If $f : X \times Y \rightarrow \mathbb{R}$ is a two-dimensional function, let $f(x, A)$ denote $\sum_{y \in A} f(x, y)$ for all $x \in X$ and $A \subseteq Y$, and $f(A, y)$ denote $\sum_{x \in A} f(x, y)$ for all $y \in Y$ and $A \subseteq X$. Let AP be a fixed, finite set of atomic propositions.

Definition 1. A labeled fully probabilistic system (FPS) is a tuple $\mathcal{D} = (S, \mathbf{P}, L)$ where S is a finite set of states, $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a probability matrix satisfying $\mathbf{P}(s, S) \in [0, 1]$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a labeling function.

A state s is called stochastic if $\mathbf{P}(s, S) = 1$, absorbing if $\mathbf{P}(s, S) = 0$, and sub-stochastic otherwise.

Definition 2. A labeled discrete-time Markov chain (DTMC) is a FPS $\mathcal{D} = (S, \mathbf{P}, L)$ where s is either absorbing or stochastic for all $s \in S$.

Definition 3. A labeled continuous-time Markov chain (CTMC) is a tuple $\mathcal{C} = (S, \mathbf{R}, L)$ with S and L as before, and a rate matrix $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geq 0}$.

The embedded DTMC of $\mathcal{C} = (S, \mathbf{R}, L)$ is defined by $emb(\mathcal{C}) = (S, \mathbf{P}, L)$ with $\mathbf{P}(s, s') = \mathbf{R}(s, s')/\mathbf{R}(s, S)$ if $\mathbf{R}(s, S) > 0$ and 0 otherwise. We will also use $\mathbf{P}$ for a CTMC directly, without referring to its embedded DTMC explicitly.

A distribution μ on S is a function $\mu : S \rightarrow [0, 1]$ satisfying the condition $\mu(S) \leq 1$. We let $Dist(S)$ denote the set of distributions over the set S . μ is called stochastic if $\mu(S) = 1$. If it is not stochastic, we use an auxiliary state (not a *real* state) $\perp \notin S$ and set $\mu(\perp) = 1 - \mu(S)$. Further, for a given FPS, let $\mathbf{P}(s, \cdot)$ denote the distribution defined by the transition probability matrix $\mathbf{P}$ for all $s \in S$. Let $\mathbf{P}(s, \perp) = 1 - \mathbf{P}(s, S)$ for all $s \in S$, and let $S_\perp$ denote the set $S \cup \{\perp\}$. For $s \in S$, let $post(s)$ denote $\{s' \in S \mid \mathbf{P}(s, s') > 0\}$, i. e., the set of successor states of s . Let $post_\perp(s)$ denote $\{s' \in S_\perp \mid \mathbf{P}(s, s') > 0\}$, i. e., $post(s)$ plus the auxiliary state $\perp$ in case that $\mathbf{P}(s, \perp) > 0$. For CTMC $\mathcal{C} = (S, \mathbf{R}, L)$, let $post(s) = \{s' \in S \mid \mathbf{R}(s, s') > 0\}$ for all $s \in S$.

For a given FPS, DTMC or CTMC, its *fanout* is defined by $\max_{s \in S} |post(s)|$, the number of states is usually denoted by n , and the number of transitions is denoted by m . For $s \in S$, $reach(s)$ denotes the set of states that are reachable from s with positive probability. For a relation $R \subseteq S \times S$ and $s \in S$, let $R[s]$ denote the set $\{s' \in S \mid (s, s') \in R\}$, and $R^{-1}[s]$ denote the set $\{s' \in S \mid (s', s) \in R\}$. If $(s, s') \in R$, we write also $s R s'$.

Simulation relations. Strong simulation is based on the notion of a *weight function*. We recall the definition here from [17], adapted to FPS as in [7].

Definition 4. Let $\mu, \mu' \in Dist(S)$ and $R \subseteq S \times S$. A weight function for (μ, μ') w. r. t. R , denoted by $\mu \sqsubseteq_R \mu'$, is a function $\Delta : S_\perp \times S_\perp \rightarrow [0, 1]$ such that $\Delta(s, s') > 0$ implies $s R s'$ or $s = \perp$, $\mu(s) = \Delta(s, S_\perp)$ for $s \in S_\perp$ and $\mu'(s') = \Delta(S_\perp, s')$ for $s' \in S_\perp$.

Strong simulation requires similar states to be related via weight functions on their distributions [17].

Definition 5. Let $\mathcal{D} = (S, \mathbf{P}, L)$ be an FPS. $R \subseteq S \times S$ is a strong simulation on $\mathcal{D}$ iff for all s_1, s_2 with $s_1 R s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$. We say that s_2 strongly simulates s_1 in $\mathcal{D}$, denoted by $s_1 \lesssim_d s_2$, iff there exists a strong simulation R on $\mathcal{D}$ such that $s_1 R s_2$.

For CTMCs we say that s_2 strongly simulates s_1 if, in addition to the DTMC conditions, s_2 can move stochastically *faster* than s_1 [7], which manifests itself by a slower rate.

Definition 6. Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC. $R \subseteq S \times S$ is a strong simulation on $\mathcal{C}$ iff for all s_1, s_2 with $s_1 R s_2$: $L(s_1) = L(s_2)$, $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$ and $\mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S)$. We say that s_2 strongly simulates s_1 in $\mathcal{C}$, denoted by $s_1 \lesssim_c s_2$, iff there exists a strong simulation R on $\mathcal{C}$ such that $s_1 R s_2$.

We now recall the notion of weak simulation on FPSs. Intuitively, s_2 weakly simulates s_1 if they have the same labels, and if their successor states can be grouped into sets U_i and V_i for $i = 1, 2$, satisfying certain conditions. We can view steps to V_i as *stutter* steps while steps to U_i are *visible* steps. It is then required that there exists a weight function for the conditional distributions: $\frac{\mathbf{P}(s_1, \cdot)}{K_1}$ and $\frac{\mathbf{P}(s_2, \cdot)}{K_2}$ where K_i intuitively correspond to the probability of performing a visible step from s_i . For reasons explained in [7], the definition needs to account for states which partially belong to U_i and partially to V_i . Technically, this is achieved by functions δ_i that distribute s_i over U_i and V_i in the definition below [7].

Definition 7. Let $\mathcal{D} = (S, \mathbf{P}, L)$ be an FPS. The relation $R \subseteq S \times S$ is a weak simulation on $\mathcal{D}$ iff for all s_1, s_2 with $s_1 R s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S_{\perp} \rightarrow [0, 1]$ and sets $U_i, V_i \subseteq S_{\perp}$ (for $i = 1, 2$) with

$$U_i = \{u_i \in \text{post}_{\perp}(s_i) \mid \delta_i(u_i) > 0\} \quad \text{and} \quad V_i = \{v_i \in \text{post}_{\perp}(s_i) \mid \delta_i(v_i) < 1\}$$

such that

1. (a) $v_1 R s_2$ for all $v_1 \in V_1 \setminus \{\perp\}$, and (b) $s_1 R v_2$ for all $v_2 \in V_2 \setminus \{\perp\}$
2. there exists a function $\Delta : S_{\perp} \times S_{\perp} \rightarrow [0, 1]$ such that:
 - (a) $\Delta(u_1, u_2) > 0$ implies $u_1 \in U_1, u_2 \in U_2$ and either $u_1 R u_2$ or $u_1 = \perp$,
 - (b) if $K_1 > 0$ and $K_2 > 0$ then for all states $w \in S_{\perp}$:

$$K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w) \quad \text{and} \quad K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$$

where $K_i = \sum_{u_i \in U_i} \delta_i(u_i) \cdot \mathbf{P}(s_i, u_i)$ for $i = 1, 2$.

3. for $u_1 \in U_1 \setminus \{\perp\}$ there exists a path fragment $s_2, w_1, \dots, w_n, u_2$ with positive probability such that $n \geq 0$, $s_1 R w_j$ for $0 < j \leq n$, and $u_1 R u_2$.

We say that s_2 weakly simulates s_1 in $\mathcal{D}$, denoted $s_1 \lesssim_d s_2$, iff there exists a weak simulation R on $\mathcal{D}$ such that $s_1 R s_2$.

Condition (3.) will in the sequel be called the *reachability condition*. If $U_2 = \emptyset$ and $U_1 \neq \emptyset$, which implies that $K_1 > 0$ and $K_2 = 0$, the reachability condition

guarantees that for any visible step $s_1 \rightarrow u_1$ with $u_1 \in U_1$, s_2 can reach a state u_2 which simulates u_1 while passing only through states simulating s_1 .

Weak simulation on DTMCs arises as a special case of the above definition, as a DTMC is an FPS (where each state is absorbing or stochastic). Weak simulation for CTMCs is defined as follows.

Definition 8 ([7,6]). Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC. $R \subseteq S \times S$ is a weak simulation on $\mathcal{C}$ iff for $s_1 R s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S \rightarrow [0, 1]$ and sets $U_i, V_i \subseteq S_\perp$ (for $i = 1, 2$) satisfying conditions (1.) and (2.) of Definition 7 and the rate condition holds: (3') $K_1 \cdot \mathbf{R}(s_1, S) \leq K_2 \cdot \mathbf{R}(s_2, S)$.

We say that s_2 weakly simulates s_1 in $\mathcal{C}$, denoted $s_1 \lesssim_c s_2$, iff there exists a weak simulation R on $\mathcal{C}$ such that $s_1 R s_2$.

In this definition, the rate condition (3') replaces the reachability condition of the preceding definition. We refer to [7] for a discussion of subtleties in this definition.

Simulation up to R . For an arbitrary relation R on the state space S of an FPS with $s_1 R s_2$, we say that s_2 simulates s_1 strongly up to R , denoted $s_1 \lesssim_R s_2$, if $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$. Otherwise we write $s_1 \not\lesssim_R s_2$. Note that $s_1 \lesssim_R s_2$ does not imply $s_1 \lesssim_d s_2$ unless R is a strong simulation, since only the first step is considered for $\lesssim_R$. Likewise, we say that s_2 simulates s_1 weakly up to R , denoted by $s_1 \lesssim_R s_2$, if there are functions δ_i , sets U_i, V_i as required by Definition 7 for this pair. Otherwise, we write $s_1 \not\lesssim_R s_2$. Similar to strong simulation up to R , $s_1 \lesssim_R s_2$ does not imply $s_1 \lesssim_d s_2$, since no conditions are imposed on pairs in R different from (s_1, s_2) . These conventions extend to DTMCs and CTMC in the obvious way.

Preflow algorithm. We briefly recall the preflow algorithm [13, p. 925] for finding the maximum flow over the network $\mathcal{N} = (V, E, u)$ where V is a finite set of vertices and $E \subseteq V \times V$ is a set of edges. V contains a distinguished *source* vertex $\uparrow$ and a distinguished *sink* vertex $\downarrow$. $u : E \rightarrow \mathbb{R}_{>0}$ is the capacity function. We extend the capacity function to all vertex pairs: $u(v, w) = 0$ if $(v, w) \notin E$. A *flow* f on $\mathcal{N}$ is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies:

1. $f(v, w) \leq u(v, w)$ for all $(v, w) \in V \times V$ *capacity constraints*
2. $f(v, w) = -f(w, v)$ for all $(v, w) \in V \times V$ *antisymmetry constraint*
3. $f(v, V) = 0$ at vertices $v \in V \setminus \{\uparrow, \downarrow\}$ *conservation rule*

The value of a flow function f is given by $f(\uparrow, V)$. A *maximum flow* is a flow of maximum value.

A *preflow* is a function $f : V \times V \rightarrow \mathbb{R}$ satisfying (1.) and (2.) above, and the relaxation of (3.): $f(V, v) \geq 0$ for all $v \in V \setminus \{\uparrow\}$. The *excess* $e(v)$ of a vertex v is defined by $f(V, v)$. A vertex $v \notin \{\uparrow, \downarrow\}$ is called *active* if $e(v) > 0$. Observe that a flow is a preflow in which no vertex v is active for $v \in V \setminus \{\uparrow, \downarrow\}$. A pair (v, w) is a *residual edge* of f if $f(v, w) < u(v, w)$. The set of residual edges w.r.t. f is denoted by E_f . The *residual capacity* $u_f(v, w)$ of the residual edge (v, w) is defined by $u(v, w) - f(v, w)$. If (v, w) is not a residual edge, it is called *saturated*. A *valid distance function* (called *valid labeling* in [13]) d is a function

$\text{SIMREL}_s(\mathcal{D})$

```

1   $R, R_{new} \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ 
2  do
3     $R \leftarrow R_{new}$  and  $R_{new} \leftarrow \emptyset$ 
4    for  $((s_1, s_2) \in R)$ 
5      if  $(s_1 \not\prec_R s_2)$ 
6         $R_{new} \leftarrow R_{new} \cup \{(s_1, s_2)\}$ .
7  until  $(R_{new} = R)$ 
8  return  $R$ 

```

Fig. 1. Basic algorithm to decide strong simulation

$V \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ satisfying: $d(\nearrow) = |V|$, $d(\searrow) = 0$ and $d(v) \leq d(w) + 1$ for every residual edge (v, w) . A residual edge (v, w) is *admissible* if $d(v) = d(w) + 1$.

We initialise the preflow f by: $f(v, w) = u(v, w)$ if $v = \nearrow$ and 0 otherwise. The distance function d is initialised by: $d(v) = |V|$ if $v = \nearrow$ and 0 otherwise. The *preflow algorithm* maintains the preflow f and the valid distance function d . If there is an active vertex v such that (v, w) is admissible, it pushes $\min\{e(v), u_f(v, w)\}$ flows from v toward the sink along the admissible edge (v, w) . If v is active but there are no admissible edges leaving it, the *relabeling* of v sets the distance of v equal to $\min\{d(w) + 1 \mid (v, w) \in E_f\}$. If there are no active vertices, the preflow f is a maximum flow.

3 Algorithms for Deciding Strong Simulation

We recall first the basic algorithm to decide strong simulation preorder. Then, we refine this algorithm to deal with strong simulations on FPSs, DTMCs and CTMCs respectively.

Basic algorithm to decide strong simulation. The algorithm in [3], depicted as SIMREL_s in Fig. 1, takes as a parameter a model, which, for now, is an FPS $\mathcal{D}$. The subscript 's' stands for strong simulation; later, a very similar algorithm, i.e., SIMREL_w , will be used for weak simulation. To calculate the strong simulation relation for $\mathcal{D}$, the algorithm starts with the trivial relation $R_{init} = \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ and removes each pair (s_1, s_2) if s_2 cannot strongly simulate s_1 up to the current relation R , i.e., $s_1 \not\prec_R s_2$. This proceeds until there is no such pair left, i.e., $R_{new} = R$. Invariantly throughout the loop it holds that R is coarser than $\preceq_d$. Hence, we obtain the strong simulation preorder $\preceq_d = R$, once the algorithm terminates.

The decisive part of the algorithm is the check whether $s_1 \prec_R s_2$ in line 5. The answer is computed with the help of a maximum flow computation on a particular network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R)$ constructed out of $\mathbf{P}(s_1, \cdot)$, $\mathbf{P}(s_2, \cdot)$ and R . This network is constructed via a graph containing a copy $\bar{t} \in \overline{S_\perp}$ of each state $t \in S_\perp$ where $\overline{S_\perp} = \{\bar{t} \mid t \in S_\perp\}$ as follows: Let $\nearrow$ (the source) and $\searrow$ (the sink) be two additional vertices not contained in $S_\perp \cup \overline{S_\perp}$. For functions

$\mu, \mu' : S \rightarrow \mathbb{R}_{\geq 0}$ and a relation $R \subseteq S \times S$ we define the network $\mathcal{N}(\mu, \mu', R) = (V, E, u)$ with the set of vertices

$$V = \{\nearrow, \searrow\} \cup \{s \in S_{\perp} \mid \mu(s) > 0\} \cup \{\overline{s} \in \overline{S_{\perp}} \mid \mu'(s) > 0\}$$

and the set of edges E is defined by

$$E = \{(s, \overline{t}) \mid (s, t) \in R \vee s = \perp\} \cup \{(\nearrow, s)\} \cup \{(\overline{t}, \searrow)\}$$

where $s, t \in S_{\perp}$ with $\mu(s) > 0$ and $\mu'(t) > 0$. The capacity function u is defined as follows: $u(\nearrow, s) = \mu(s)$ for all $s \in S_{\perp}$, $u(\overline{t}, \searrow) = \mu'(t)$ for all $t \in S_{\perp}$, $u(s, \overline{t}) = \infty$ for all $(s, t) \in E$ and $u(v, w) = 0$ otherwise. This network is a bipartite network, where the vertices can be partitioned into two subsets $V_1 := \{s \in S_{\perp} \mid \mu(s) > 0\} \cup \{\searrow\}$ and $V_2 := \{\overline{s} \in \overline{S_{\perp}} \mid \mu'(s) > 0\} \cup \{\nearrow\}$ such that all edges have one endpoint in V_1 and another in V_2 . For two states s_1, s_2 of an FPS or a CTMC, we let $\mathcal{N}(s_1, s_2, R)$ denote the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R)$.

The following lemma expresses the crucial relationship between maximum flows and weight functions on which the algorithm is based. It is a direct extension of [3, Lemma 5.1] now accounting for sub-stochasticity.

Lemma 1. *Let S be a finite set of states and R be a relation on S . Let $\mu, \mu' \in \text{Dist}(S)$. Then, $\mu \sqsubseteq_R \mu'$ iff the maximum flow in $\mathcal{N}(\mu, \mu', R)$ is 1.*

Thus we can decide $s_1 \lesssim_R s_2$ by computing the maximum flow in $\mathcal{N}(s_1, s_2, R)$. Using the best known flow algorithm for this type of networks [8,12], one obtains the overall complexity $\mathcal{O}(n^7 / \log n)$ for the algorithm SIMREL_s [3]. The space complexity is $\mathcal{O}(n^2)$.

An improved algorithm for FPSs. We first analyse the behaviour of SIMREL_s in more detail. We consider an arbitrary pair (s_1, s_2) , and assume that (s_1, s_2) stays in relation R throughout the iterations, until the pair is either found not to satisfy $s_1 \lesssim_R s_2$ or the algorithm terminates with a fix-point. If we let $l_{(s_1, s_2)}$ denote the number of iterations until either of these happens, then altogether $l_{(s_1, s_2)}$ maximum flow algorithms are run for this pair. However, the networks $\mathcal{N}(s_1, s_2, \cdot)$ constructed in successive iterations are very similar, and may often be identical across iterations: They differ from iteration to iteration only by deletion of some edges induced by the successive clean up of R . For our particular pair (s_1, s_2) the network might not change at all in some iterations, because the deletions from R do not affect their direct successors. We are going to exploit this observation by an algorithm that re-uses the already computed maximum flows, in a way that whatever happens is good: If no changes occur to $\mathcal{N}(s_1, s_2, \cdot)$, then the maximum flow is equal to the one in the last iteration. If changes occur, the preflow algorithm can be applied to get the new maximum flow very fast, using the maximum flow constructed in the last iteration as a preflow.

To understand the algorithm, we look at the network $\mathcal{N}(s_1, s_2, R_{\text{init}})$. Let $D_1, \dots, D_k$ be pairwise disjoint subsets of R_{init} , which correspond to the pairs deleted from R_{init} in iteration i . Let $\mathcal{N}(s_1, s_2, R_i)$ denote $\mathcal{N}(s_1, s_2, R_{\text{init}})$ if $i = 1$, and $\mathcal{N}(s_1, s_2, R_{i-1} \setminus D_{i-1})$ if $1 < i \leq k + 1$. Let f_i denote the maximum flow

of the network $\mathcal{N}(s_1, s_2, R_i)$ for $i = 1, \dots, k + 1$. We address the problem of checking $|f_i| = 1$ for all $i = 1, \dots, k + 1$. Very similar to the *parametric maximum algorithm* [10, p. 34], our algorithm $\text{SMF}_{(s_1, s_2)}$ (*sequence of maximum flows*) for the pair (s_1, s_2) consists of initialising the preflow $f_{(s_1, s_2)}$ and the distance function $d_{(s_1, s_2)}$ as for the preflow algorithm, setting $i = 0$, and repeating the following steps at most k times:

$\text{SMF}_{(s_1, s_2)}$

1. Increase i by 1. If $i = 1$ go to step 2. Otherwise, for all pairs $(u_1, u_2) \in D_{i-1}$, set $f_{(s_1, s_2)}(u_1, \overline{u_2}) = 0$ and replace the flow $f_{(s_1, s_2)}(\overline{u_2}, \downarrow)$ by $f_{(s_1, s_2)}(\overline{u_2}, \downarrow) - f_{(s_1, s_2)}(u_1, \overline{u_2})$. Set $\mathcal{N}(s_1, s_2, R_i) = \mathcal{N}(s_1, s_2, R_{i-1} \setminus D_{i-1})$. Let $f_{(s_1, s_2)}$ and $d_{(s_1, s_2)}$ be the resulting flow and final valid distance function.
2. Apply the preflow algorithm to calculate the maximum flow for $\mathcal{N}(s_1, s_2, R_i)$ with preflow $f_{(s_1, s_2)}$ and distance function $d_{(s_1, s_2)}$.
3. If $|f_{(s_1, s_2)}| < 1$ return false for all $j \geq i$. Otherwise, return true and continue with step 1.

To understand this algorithm, assume $i > 1$. At step (1.), before we remove the edges D_{i-1} from the network $\mathcal{N}(s_1, s_2, R_{i-1})$, we modify the flow $f_{(s_1, s_2)}$, which is the maximum flow of the network $\mathcal{N}(s_1, s_2, R_{i-1})$, by

- setting $f_{(s_1, s_2)}(u_1, \overline{u_2}) = 0$ for all deleted edges $(u_1, u_2) \in D_{i-1}$, and
- modifying $f_{(s_1, s_2)}(\overline{u_2}, \downarrow)$ such that the flow $f_{(s_1, s_2)}$ becomes consistent with the flow conservation rule.

The excess $e(v)$ is increased if there exists w such that $(v, w) \in D_{i-1}$, and unchanged otherwise. Hence, the modified flow is a preflow. The distance function $d_{(s_1, s_2)}$ keeps valid, since by removing the set of edges D_{i-1} , no new residual edges are induced. This guarantees that, at step (2.), the *preflow algorithm* finds a maximum flow over the network $\mathcal{N}(s_1, s_2, R_i)$. If $|f_{(s_1, s_2)}| < 1$ at some iteration i , then $|f_{(s_1, s_2)}| < 1$ for all iterations $j \geq i$ because more edges will be deleted in subsequent iterations. Therefore, at step (3.), the algorithm returns true and continues with step (1.) if $|f_{(s_1, s_2)}| = 1$, otherwise, returns false for all subsequent iterations. We derive the complexity of the algorithm as follows:

Lemma 2. *Let $D_1, \dots, D_k$ be pairwise disjoint subsets of $R_{init} \cap \text{post}(s_1) \times \text{post}(s_2)$. Let f_i denote the flow constructed at the end of step (2.) in iteration i . Assume that $|\text{post}(s_1)| \leq |\text{post}(s_2)|$. The algorithm $\text{SMF}_{(s_1, s_2)}$ correctly computes maximum flow f_i for $\mathcal{N}(s_1, s_2, R_i)$ where $i = 1, \dots, k + 1$, and runs in time $\mathcal{O}(|\text{post}(s_1)| |\text{post}(s_2)|^2)$.*

The improved algorithm SIMREL'_s for FPSs is depicted in Fig. 2. The variable l (line 2) denotes the number of iterations of the until-loop, and the set D (line 9) contains edges removed from R . For every pair $(s_1, s_2) \in R_{init}$, the network $\mathcal{N}(s_1, s_2, R_{init})$ (line 4), the flow function $f_{(s_1, s_2)}$ and the distance function $d_{(s_1, s_2)}$ are initialised as for the preflow algorithm (line 5). At line 6 a set

$$\text{Listener}_{(s_1, s_2)} = \{(u_1, u_2) \mid u_1 \in \text{pre}(s_1) \wedge u_2 \in \text{pre}(s_2) \wedge L(u_1) = L(u_2)\}$$

$\text{SIMREL}'_s(\mathcal{D})$

```

1   $R, R_{new} \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ 
2   $l \leftarrow 0$  // auxiliary variable to count the number of iterations.
3  for  $((s_1, s_2) \in R)$ 
4      Construct the initial network  $\mathcal{N}(s_1, s_2, R_{init}) := \mathcal{N}(s_1, s_2, R)$ 
5      Initialise the flow function  $f_{(s_1, s_2)}$  and the distance function  $d_{(s_1, s_2)}$ 
6       $\mathbf{Listener}_{(s_1, s_2)} \leftarrow \{(u_1, u_2) \mid u_1 \in \text{pre}(s_1) \wedge u_2 \in \text{pre}(s_2) \wedge L(u_1) = L(u_2)\}$ 
7  do
8       $l++$ 
9       $D \leftarrow R \setminus R_{new}$  and  $R \leftarrow R_{new}$  and  $R_{new} \leftarrow \emptyset$ 
10     for  $((s_1, s_2) \in D)$ 
11         for  $((u_1, u_2) \in \mathbf{Listener}_{(s_1, s_2)})$ 
12              $D_l^{(u_1, u_2)} \leftarrow D_l^{(u_1, u_2)} \cup \{(s_1, s_2)\}$ 
13     for  $((s_1, s_2) \in R)$ 
14         if  $(\text{SMF}_{(s_1, s_2)} \text{ returns true on the set } D_l^{(s_1, s_2)})$ 
15              $R_{new} \leftarrow R_{new} \cup \{(s_1, s_2)\}$ .
16 until  $(R_{new} = R)$ 
17 return  $R$ 

```

Fig. 2. Efficient algorithm for deciding strong simulation

is saved, where $\text{pre}(s) = \{t \in S \mid \mathbf{P}(t, s) > 0\}$. The set $\mathbf{Listener}_{(s_1, s_2)}$ contains all pairs (u_1, u_2) such that the network $\mathcal{N}(u_1, u_2, R)$ contains the edge $(s_1, \overline{s_2})$. In lines 10–12, the pair (s_1, s_2) is inserted into the set $D_l^{(u_1, u_2)}$ if $(s_1, s_2) \in D$ and $(u_1, u_2) \in \mathbf{Listener}_{(s_1, s_2)}$. $D_l^{(u_1, u_2)}$ contains edges which should be removed to update the network for (u_1, u_2) in iteration l . At line 14, the algorithm $\text{SMF}_{(s_1, s_2)}$ constructs the maximum flow for the set $D_l^{(s_1, s_2)}$. Note that l corresponds to i in SMF . The initialisation of SMF corresponds to lines 4–5. In the first iteration (in which $D_1^{(s_1, s_2)} = \emptyset$), $\text{SMF}_{(s_1, s_2)}$ skips the computations in step (1.) and proceeds directly to step (2.), in which the maximum flow f_1 for $\mathcal{N}(s_1, s_2, R_{init})$ is constructed. In iteration $l > 1$, $\text{SMF}_{(s_1, s_2)}$ takes the set $D_l^{(s_1, s_2)}$, updates the flow f_{l-1} and the network, and constructs the maximum flow f_l for the network $\mathcal{N}(s_1, s_2, R_l)$. If $\text{SMF}_{(s_1, s_2)}$ returns true, (s_1, s_2) is inserted into R_{new} and survives this iteration.

Lemma 3. $\text{SIMREL}'_s(\mathcal{D})$ runs in time $\mathcal{O}(m^2n)$ and in space $\mathcal{O}(m^2)$. If the fanout is bounded by a constant, it has complexity $\mathcal{O}(n^2)$, both in time and space.

Algorithm for DTMCs and CTMCs. We now consider how to handle DTMCs and CTMCs. Since each DTMC is a special case of an FPS the algorithm SIMREL'_s applies directly. For CTMCs, we replace line 1 of the algorithm by

$$R, R_{new} \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge \mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S)\}$$

to check the rate condition of Definition 6. We arrive at the same complexity.

Fig. 3. A CTMC example

Example 1. Consider the CTMC in Fig. 3 (it has 10 states) where labels are indicated by shades of grey. Consider the pair $(s_1, s_2) \in R_{init}$. The network $\mathcal{N}(s_1, s_2, R_1)$ is depicted on the right of the figure. Assume that we get the maximum flow f_1 which sends $\frac{1}{2}$ flow along the path $\nearrow, u_2, \overline{u_4}, \searrow$ and $\frac{1}{2}$ along $\nearrow, u_1, \overline{u_3}, \searrow$. Hence, the check for (s_1, s_2) is successful in the first iteration. The checks for the pairs (u_1, u_3) , (u_1, u_4) and (u_2, u_3) are also successful in the first iteration. However, the check for the pair (u_2, u_4) is unsuccessful, as no successor of u_2 has the same label as x_1 . In the second iteration, the network $\mathcal{N}(s_1, s_2, R_2)$ is obtained from $\mathcal{N}(s_1, s_2, R_1)$ by deleting the edge $(u_2, \overline{u_4})$. In $\mathcal{N}(s_1, s_2, R_2)$, the flows on $(u_2, \overline{u_4})$ and $(\overline{u_4}, \searrow)$ are set to 0, and the vertex u_2 has a positive excess $\frac{1}{2}$. Applying the preflow algorithm, we push the excess from u_2 , along $\overline{u_3}, u_1, \overline{u_4}$ to $\searrow$. We get a maximum flow f_2 for $\mathcal{N}(s_1, s_2, R_2)$ which sends $\frac{1}{2}$ flow along the path $\nearrow, u_2, \overline{u_3}, \searrow$ and $\frac{1}{2}$ along $\nearrow, u_1, \overline{u_4}, \searrow$. Hence, the check for (s_1, s_2) is also successful in the second iteration. Once the fix-point is reached, R still contains (s_1, s_2) .

4 Algorithms for Deciding Weak Simulation

We now turn our attention to algorithms to decide weak simulation $\approx_c$. We first focus on FPSs before addressing DTMCs and CTMCs. The theoretical complexity of the algorithms for DTMCs and CTMCs are the same as the one for FPSs (except for bounded fanout). Nevertheless, we shall present dedicated algorithms for DTMCs and CTMCs, because their specific properties can be exploited for significant improvements in practice.

An algorithm for FPSs. The basic weak simulation algorithm $\text{SIMREL}_w(\mathcal{D})$ is obtained by replacing line 5 of $\text{SIMREL}_s(\mathcal{D})$ in Fig. 1 by: **if** $(s_1 \approx_R s_2)$. Thus instead of checking the pairs w.r.t. $\preceq_R$ we check them w.r.t. $\approx_R$. The latter check is performed by $\text{WS}(\mathcal{D}, s_1, s_2, R)$, shown in Fig. 4.

Here, line 1 corresponds to the case that s_1 has only *stutter* steps, i.e., $K_1 = 0$: Assuming $\text{post}(s_1) \subseteq R^{-1}[s_2]$ we choose $U_1 = \emptyset, V_1 = \text{post}_\perp(s_1)$ and $U_2 = \text{post}(s_2), V_2 = \{\perp\}$ to fulfill the conditions in Definition 7. Hence, $s_1 \approx_R s_2$. If line 3 is reached in WS, s_1 has at least one *visible* step, and all successors of s_2 can simulate s_1 . In this case we need to check the reachability condition (3.) of Definition 7, which is performed in lines 3–6. Line 7 of the algorithm is only

$\text{WS}(\mathcal{D}, s_1, s_2, R)$

```

1  if ( $\text{post}(s_1) \subseteq R^{-1}[s_2]$ ) return true
2  if ( $\text{post}(s_2) \subseteq R[s_1]$ )
3    if ( $\exists s \in \text{reach}(s_2)$ , such that  $s \notin R[s_1]$ ) return true
4     $U_1 \leftarrow \{s'_1 \in \text{post}(s_1) \mid s'_1 \notin R^{-1}[s_2]\}$ 
5    if ( $\forall u_1 \in U_1. \exists s \in \text{reach}(s_2)$ , such that  $s \in R[u_1]$ ) return true
6    else return false
7  return  $\text{WSFPS}(\mathcal{D}, s_1, s_2, R)$ 

```

Fig. 4. Algorithm to check whether $s_1 \lesssim_R s_2$

touched if the checks in line 1 and 2 both return false. In this case, more work is needed, and this work is delegated to a parametric maximum flow algorithm, which is called by $\text{WSFPS}(\mathcal{D}, s_1, s_2, R)$.

To understand the details of this algorithm, we require a bit of notation. We focus on a particular pair $(s_1, s_2) \in R$, where R is the current relation. We partition the set $\text{post}_\perp(s_i)$ into MU_i (for: must be in U_i) and PV_i (for: potentially in V_i). The set PV_1 consists of those successors of s_1 which can be either put into U_1 or V_1 or both. For technical reasons, we assume additionally that $\perp \in PV_1$ if s_1 is not stochastic. Hence,

$$PV_1 = \text{post}_\perp(s_1) \cap (R^{-1}[s_2] \cup \{\perp\})$$

The set MU_1 equals $\text{post}_\perp(s_1) \setminus PV_1$ which consists of the successor states which can only be placed in U_1 . The sets PV_2 and MU_2 are defined similarly:

$$PV_2 = \text{post}_\perp(s_2) \cap (R[s_1] \cup \{\perp\})$$

and $MU_2 = \text{post}_\perp(s_2) \setminus PV_2$. Obviously, $\delta_i(u) = 1$ for $u \in MU_i$ for $i = 1, 2$.

We write $\gamma \mathbf{P}$ to denote a distribution $\mathbf{P}$ scaled by a constant $\gamma \in \mathbb{R}_{>0}$. If s_1, s_2 and R are clear from the context, we let $\mathcal{N}(\gamma)$ denote the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \gamma \mathbf{P}(s_2, \cdot), R)$. We say a flow function f of $\mathcal{N}(\gamma)$ is valid for $\mathcal{N}(\gamma)$ iff f saturates all edges $(\uparrow, u_1)$ with $u_1 \in MU_1$ and all edges $(\overline{u_2}, \downarrow)$ with $u_2 \in MU_2$. For $\gamma \in \mathbb{R}_{>0}$, we address the problem of checking whether there exists a valid flow f for $\mathcal{N}(\gamma)$. This is a *feasible flow problem* with lower bounds (f saturates edges to MU_1 and from MU_2) and upper bounds (the capacities) on the flows. It can be solved by applying a simple transformation to the graph (in time $\mathcal{O}(|MU_1| + |MU_2|)$), solving the maximum flow problem for the transformed graph, and checking whether the flow is large enough. Details are, for example, described in [1, p. 169–170].

If there exists a valid flow f for $\mathcal{N}(\gamma)$, we say that γ is valid for $\mathcal{N}(\gamma)$. The following lemma shows that the algorithm $\text{WSFPS}(\mathcal{D}, s_1, s_2, R)$ can be reduced to checking whether there exists a valid γ for $\mathcal{N}(\gamma)$.

Lemma 4. *If $\text{WS}(\mathcal{D}, s_1, s_2, R)$ reaches line 7, $s_1 \lesssim_R s_2$ iff there exists a valid γ for $\mathcal{N}(\gamma)$.*

In the network $\mathcal{N}(\gamma)$ the capacities of the edges leading to the sink are an increasing function of a real-valued parameter γ . $\mathcal{N}(\gamma)$ is a *parametric network*¹ as described in [10, p. 33]. We recall briefly the *breakpoints* [10, p. 37–42] of $\mathcal{N}(\gamma)$. Let $\kappa(\gamma)$ denote the *minimum cut capacity function*, which is the capacity of a minimum cut as a function of γ . A breakpoint is a value γ_0 at which the slope of $\kappa(\gamma)$ changes. $\kappa(\gamma)$ is a piecewise-linear concave function with at most $|V| - 2$ breakpoints where $|V|$ denotes the number of vertices of $\mathcal{N}(\gamma)$. The $|V| - 1$ or fewer line segments forming the graph of $\kappa(\gamma)$ correspond to $|V| - 1$ or fewer distinct cuts. The same minimum cut can be chosen on the same slope of $\kappa(\gamma)$, and at breakpoints certain edges become saturated or unsaturated. As we expect, it is sufficient to consider only the breakpoints of $\mathcal{N}(\gamma)$:

Lemma 5. *There exists a valid γ for $\mathcal{N}(\gamma)$ iff one of the breakpoints of $\mathcal{N}(\gamma)$ is valid.*

All of the breakpoints can be obtained by the *breakpoint algorithm* [10, p. 40], which we embed into our algorithm WSFPS as follows:

WSFPS($\mathcal{D}, s_1, s_2, R$)

1. Compute all of the breakpoints $b_1 < b_2 < \dots < b_j$ of $\mathcal{N}(\gamma)$.
2. Return true, iff for some $i \in \{1, \dots, j\}$, b_i is valid for $\mathcal{N}(b_i)$.

The following lemma gives the correctness of the algorithm WS:

Lemma 6. *WS($\mathcal{D}, s_1, s_2, R$) returns true iff $s_1 \lesssim_R s_2$.*

For each given breakpoint, we need to solve one feasible flow problem to check whether it is valid. So overall we apply at most $|V| - 2$ times feasible flow algorithms for all breakpoints. Applying a binary search method over the breakpoints, a better bound can be achieved where only $\log(|V|)$ maximum flow problems need to be solved. This allows us to achieve the following complexity result:

Lemma 7. *SIMREL_w($\mathcal{D}$) runs in time $\mathcal{O}(m^2 n^3)$ and in space $\mathcal{O}(n^2)$. If the fanout g is bounded by a constant, the time complexity is $\mathcal{O}(n^5)$.*

An algorithm for DTMCs. Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC. We exploit the absence of sub-stochasticity in DTMC to arrive at an improved algorithm, in which we achieve the effect of WSFPS($\mathcal{D}, s_1, s_2, R$) via only one maximum flow problem.

Let H denote the sub-relation $R \cap [(post(s_1) \cup \{s_1\}) \times (post(s_2) \cup \{s_2\})]$ which is the local fragment of the relation R . Now let $A_1, A_2, \dots, A_h$ enumerate the classes of the equivalence relation $(H \cup H^{-1})^*$ generated by H where h denotes the number of classes. W.l.o.g., we assume in the following that A_h is the equivalence class containing s_1 and s_2 , i.e., $s_1, s_2 \in A_h$. The following lemma gives some properties of the sets A_i provided that $s_1 \lesssim_R s_2$:

¹ In [10, p. 33], the capacities leading to $\downarrow$ is a non-increasing function of γ . As indicated in [10, p. 36], if we reverse the directions of all the edges and exchange the source and sink, the algorithms presented there can be used directly.

WSDTMC($\mathcal{D}, s_1, s_2, R$)

```

1  Construct the partition  $A_1, \dots, A_h$ 
2  if ( $h = 1$ ) return WSFPS( $\mathcal{D}, s_1, s_2, R$ )
3  foreach  $i \leftarrow 1, 2, \dots, h-1$ 
4      if ( $\mathbf{P}(s_1, A_i) = \mathbf{P}(s_2, A_i) = 0$ ) raise error
5      else if ( $\mathbf{P}(s_1, A_i) = 0$  or  $\mathbf{P}(s_2, A_i) = 0$ ) return false
6      else  $\gamma_i \leftarrow \frac{\mathbf{P}(s_1, A_i)}{\mathbf{P}(s_2, A_i)}$ 
7  if ( $\gamma_i \neq \gamma_j$  for some  $i, j < h$ ) return false
8  return true iff  $\gamma_1$  is valid for  $\mathcal{N}(\gamma_1)$ .
```

Fig. 5. Algorithm to check whether $s_1 \lesssim_R s_2$ tailored to DTMCs

Lemma 8. For $(s_1, s_2) \in R$, assume that there exists a state $s'_1 \in \text{post}(s_1)$ such that $s'_1 \notin R^{-1}[s_2]$, and $s'_2 \in \text{post}(s_2)$ such that $s'_2 \notin R[s_1]$. Let $A_1, \dots, A_h$ be the sets constructed for (s_1, s_2) as above. If $s_1 \lesssim_R s_2$, the following hold:

1. $\mathbf{P}(s_1, A_i) > 0$ and $\mathbf{P}(s_2, A_i) > 0$ for all $i < h$
2. $\gamma_i = \frac{K_1}{K_2}$ for all $i < h$ where $\gamma_i = \frac{\mathbf{P}(s_1, A_i)}{\mathbf{P}(s_2, A_i)}$

The algorithm WSDTMC is presented in Fig. 5. The partition $A_1, \dots, A_h$ is constructed in line 1. If $h = 1$ (line 2), it is reduced to WSFPS($\mathcal{D}, s_1, s_2, R$). Lines 3–7 follows directly from Lemma 8. Line 8 follows from the following lemma, which is the counterpart of Lemma 4:

Lemma 9. Assume WSDTMC($\mathcal{D}, s_1, s_2, R$) reaches line 7 and $h > 1$, $s_1 \lesssim_R s_2$ iff γ_1 is valid for $\mathcal{N}(\gamma_1)$.

One might expect that this lemma allows us to establish a better time bound for DTMCs in the order of $\log n$. This is indeed the case if $h > 1$ for each pair (s_1, s_2) in the initial R_{init} , which is a peculiar structural restriction: the labels of at least one successor of s_1 or s_2 must differ from $L(s_1)$ (or s_2). In this special case we can even establish the time bound $\mathcal{O}(m^2n)$, the same as for strong simulation.

An algorithm for CTMCs. We now discuss how to handle CTMCs. Recall that in Definition 8, we have the rate condition (3'): $K_1 \mathbf{R}(s_1, S) \leq K_2 \mathbf{R}(s_2, S)$. To determine the weak simulation $\lesssim_c$, we simplify the algorithm for DTMCs as follows. If $K_1 > 0$ and $K_2 = 0$, we must have $s_1 \not\lesssim_R s_2$ because of the rate condition. Hence, the check of the reachability condition in lines 2–6 of the algorithm Ws($\mathcal{C}, s_1, s_2, R$) can be skipped. At line 7 the algorithm WSDTMC($\mathcal{C}, s_1, s_2, R$) is called as before. To check the additional rate condition in WSDTMC we use the following lemma:

Lemma 10. Assume that $s_1 \lesssim_R s_2$ in $\text{emb}(\mathcal{C})$ and there exists $s'_1 \in \text{post}(s_1)$ such that $s'_1 \notin R^{-1}[s_2]$. We let $\gamma_{\min}$ denote the minimal valid breakpoint for $\mathcal{N}(\gamma_{\min})$ in $\text{emb}(\mathcal{C})$. Then, $s_1 \lesssim_R s_2$ in $\mathcal{C}$ iff $\gamma_{\min} \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$.

To check the rate condition for the case $h > 1$, we replace line 8 of the algorithm WSDTMC by:

return true iff $\gamma_1 \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ and γ_1 is valid for $\mathcal{N}(\gamma_1)$

In case $h = 1$, WSDTMC calls WsFPS($\mathcal{C}, s_1, s_2, R$) in line 2. We replace line 2 of WsFPS by:

Return true iff, for some $i \in \{1, \dots, j\}$, $b_i \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ and b_i is valid for $\mathcal{N}(b_i)$

to check the rate condition. The existential quantifier corresponds to the minimal valid breakpoint requirement. Similar to FPSs, a binary search method over the breakpoints can be used to find the minimal valid breakpoint. As checking the reachability condition is not required for CTMCs, we get even a better bound for sparse CTMCs:

Lemma 11. *If the fanout g of CTMC $\mathcal{C}$ is bounded by a constant, the time complexity is $\mathcal{O}(n^4)$.*

5 Conclusions

We have introduced efficient algorithms to decide simulation on Markov models. For sparse models where the fanout is bounded by a constant, we achieve the complexities $\mathcal{O}(n^2)$ for strong and $\mathcal{O}(n^4)$ for weak simulation relations on CTMCs, and $\mathcal{O}(n^5)$ for DTMCs, respectively. If instead one uses the original algorithm for weak simulation combined with the polynomial method to solve such an LP ($\mathcal{O}(n^{10} \cdot r)$) [20], one would obtain a time complexity of $\mathcal{O}(n^{14} \cdot r)$ where r is the maximal binary encoding length of a coefficient of the LP. The weak simulation algorithm is polynomial in the RAM-model of computation while no known linear programming based algorithm is.

We believe that the strong and weak simulation algorithms are core contributions in the quest for model checking techniques of ever larger Markov chains. Currently, the main bottleneck is the prohibitively unstructured computations required in the numerical solution phase, resulting in the need to store an n -dimensional vector of floating point values in memory without much chance for an efficient symbolic representation [19].

At first sight our situation is worse, since we have to keep flows in the order of m^2 across the iterations of our algorithm. But since these flows are resulting from very local computations (on bipartite, loop-free networks of diameter 3), they are much more structured, making it possible to utilise symbolic and hashing techniques effectively in their internal representation. We therefore expect that the algorithms in this paper can effectively be employed to reduce the – otherwise prohibitive – size of a Markov chain prior to numerically checking a safe temporal logic formula. By doing so, we trade time against memory, because the direct numerical solution is practically of quadratic complexity (in n) – but only if the above vector fits in memory.

Acknowledgments. The authors are grateful to Björn Wachter (Saarland University) for helpful comments at an early state of this paper.

References

1. R. K. Ahuja, T. L. Magnanti, J. B. Orlin: Network Flows: theory, algorithms, and applications. Prentice Hall, 1993
2. A. Aziz, K. Sanwal, V. Singhal, R. K. Brayton: Verifying Continuous Time Markov Chains. In CAV (1996) 269–276
3. C. Baier, B. Engelen, M. E. Majster-Cederbaum: Deciding Bisimilarity and Similarity for Probabilistic Processes. J. Comput. Syst. Sci. **60**(1) (2000) 187–231
4. C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen: Model-Checking Algorithms for Continuous-Time Markov Chains. IEEE Trans. Software Eng. **29**(6) (2003) 524–541
5. C. Baier and H. Hermanns and J.-P. Katoen: Probabilistic weak simulation is decidable in polynomial time. Inf. Process. Lett. **89**(3) (2004) 123–130
6. C. Baier, J.-P. Katoen, H. Hermanns, B. Haverkort: Simulation for Continuous-Time Markov Chains. In CONCUR (2002) 338–354
7. C. Baier, J.-P. Katoen, H. Hermanns, V. Wolf: Comparative branching-time semantics for Markov chains. Inf. Comput **200**(2) (2005) 149–214
8. J. Cheriyan, T. Hagerup, K. Mehlhorn: Can a Maximum Flow be Computed in $\mathcal{O}(nm)$ Time? In Proc. ICALP (1990) 235–248
9. E. M. Clarke, O. Grumberg, D. E. Long: Model Checking and Abstraction. ACM Transactions on Programming Languages and Systems **16**(5) (1994) 1512–1542
10. G. Gallo, M. D. Grigoriadis, R. E. Tarjan: A fast parametric maximum flow algorithm and applications. SIAM J. Comput. **18**(1) (1989) 30–55
11. R. Gentilini, C. Piazza, A. Policriti: From Bisimulation to Simulation: Coarsest Partition Problems. J. Autom. Reasoning **31**(1) (2003) 73–103
12. A. V. Goldberg: Recent Developments in Maximum Flow Algorithms (Invited Lecture). In SWAT (1998) 1–10
13. A. V. Goldberg, R. E. Tarjan: A new approach to the maximum-flow problem. J. ACM **35**(4) (1988) 921–940
14. H. Hansson, B. Jonsson: A Logic for Reasoning about Time and Reliability. Formal Asp. Comput. **6**(5) (1994) 512–535
15. M. R. Henzinger, T. A. Henzinger, P. W. Kopke: Computing Simulations on Finite and Infinite Graphs. In FOCS (1995) 453–462
16. B. Jonsson: Simulations Between Specifications of Distributed Systems. In CONCUR (1991) 346–360
17. B. Jonsson, K. G. Larsen: Specification and Refinement of Probabilistic Processes. In LICS (1991) 266–277
18. R. Milner: Communication and Concurrency. Prentice Hall, 1989
19. D. Parker: Implementation of Symbolic Model Checking for Probabilistic Systems. University of Birmingham, 2002
20. A. Schrijver: Theory of Linear and Integer Programming. Wiley, 1986