Approximate Probabilistic Parallel Multiset Rewriting using MCMC

Stefan Lüdtke, Max Schröder, and Thomas Kirste

Institute of Computer Science, University of Rostock, Germany
{stefan.luedtke2, max.schroeder, thomas.kirste}@uni-rostock.de

Abstract. Probabilistic parallel multiset rewriting systems (PPMRS) model probabilistic, dynamic systems consisting of multiple, (inter-) acting agents and objects (entities), where multiple individual actions can be performed in parallel. The main computational challenge in these approaches is computing the distribution of parallel actions (*compound actions*), that can be formulated as a constraint satisfaction problem (CSP). Unfortunately, computing the partition function for this distribution exactly is infeasible, as it requires to enumerate all solutions of the CSP, which are subject to a combinatorial explosion.
The central technical contribution of this paper is an efficient Markov Chain Monte Carlo (MCMC)-based algorithm to approximate the partition function, and thus the compound action distribution. The proposal function works by performing backtracking in the CSP search tree, and then sampling a solution of the remaining, partially solved CSP.
We demonstrate our approach on a Lotka-Volterra system with PPMRS semantics, where exact compound action computation is infeasible. Our approach allows to perform simulation studies and Bayesian filtering with PPMRS semantics in scenarios where this was previously infeasible.

Keywords: Bayesian filtering, probabilistic multiset rewriting system, Metropolis-Hastings algorithm, Markov chain monte carlo, constraint satisfaction problem

1 Introduction

Modelling dynamic systems is fundamental for a variety of AI tasks. Multiset Rewriting Systems (MRSs) provide a convenient mechanism to represent dynamic systems that consist of multiple (inter-)acting entities where the system dynamics can be described in terms of *rewriting rules* (also called actions). Typically, MRS are used for simulation studies, e.g. in chemistry [2], systems biology [13] or ecology [16].

Recently, *Lifted Marginal Filtering* (LiMa) [12, 18] was proposed, an approach that uses a MRS to describe the state dynamics and maintains the *state distribution* over time, which is repeatedly updated based on observations (i.e. it performs Bayesian filtering). More specifically, the transition model of LiMa is described in terms of a *probabilistic parallel MRS* (PPMRS) [1], a specific class

of MRSs that model systems where multiple entities act in parallel. This allows to perform Bayesian filtering in scenarios where multiple entities can simultaneously perform activities between consecutive observations, but the order of actions between observations is not relevant.

A multiset of actions that is executed in parallel is called *compound action*. In PPRMS, each state s defines a distribution of compound actions k, $p(k|s)$. This distribution defines the transition distribution $p(s'|s)$, where s' is the result of applying k to s (called *transition model* in the Bayesian filtering context).

One of the computational challenges in probabilistic parallel MRSs is the computation of $p(k|s)$: This distribution is calculated as the normalized *weight* $v_s(k)$ of the compound actions: $p(k|s) = v_s(k)/\sum_{k_i} v_s(k_i)$. To compute this normalization factor (called partition function) exactly, it is necessary to sum over all compound actions. Unfortunately, the number of compound actions can be very large, due to the large number of combinations of actions that can be applied in parallel to a state. Thus, in general, complete enumeration is infeasible. Therefore, we are concerned with methods for approximating this distribution.

A problem closely related to computing the value of the partition function is *weighted model counting* (WMC), where the goal is to find the summed weight of all models of a weighted propositional theory (W-SAT). Exact [4] and approximate [19, 7] algorithms for WMC have been proposed. However, our approach requires to sample from the distribution $p(k|s)$, not just compute its partition function. For W-SAT, a method was proposed [3] to sample solutions, based on partitioning the set of satisfying assignments into "cells", containing equal numbers of satisfying assignments. The main reason why these approaches cannot be used directly for our domain is that they assume a specific structure of the weights (weights factorize into weights of literals), whereas in our domain, only weights $v(k)$ of complete samples k are available. Another related line of research is efficiently sampling from distributions with many zeros (hard constraints) [9], which can also be achieved by a combination of sampling and backtracking. However, they assume that the distribution to sample from is given in factorized form (e.g. as a graphical model).

The main technical contribution of this paper is a sampling approach for compound actions, based on the Metropolis-Hastings algorithm. Compound action computation can be formulated as a constraint satisfaction problem (CSP), where each compound action is a solution of the CSP. The algorithm works by iteratively proposing new CSP solutions, based on backtracking of the current solution (i.e. compound action).

We will proceed as follows. In Section 2, we introduce probabilistic parallel MRSs in more detail. The exact and approximate algorithms for computing the compound action distribution are presented in Section 3. We present an empirical evaluation of our approach in Section 4, showing that the transition model can be approximated accurately for situations with thousands of entities, where the exact algorithm is infeasible.

2 Probabilistic Parallel Multiset Rewriting

In the following, we introduce probabilistic parallel multiset rewriting systems (PPMRSs), and show how such a system defines the state transition distribution (also called *transition model*) $p(S_{t+1}|S_t)$.

Such systems have previously been investigated in the context of P Systems [15], a biologically inspired formalism based on parallel multiset rewriting across different regions (separated by *membranes*). Several probabilistic variants of P Systems have been proposed [1, 16, 5]. We present a slightly different variant here, that does not use membranes, but structured entities (the variant that is used in LiMa [12, 18]).

Let $\mathcal{E}$ be a set of entities. A **multiset** over $\mathcal{E}$ is a map $s : \mathcal{E} \to \mathbb{N}$ from entities to multiplicities. We denote a multiset of entities $e_1, \ldots, e_i$ and their multiplicities $n_1, \ldots, n_i$ as $[\![\, n_1\ e_1, \ldots, n_i\ e_i \,]\!]$, and define multiset union $s \uplus s'$, multiset difference $s \uplus s'$, and multiset subsets $s \sqsubseteq s'$ in the obvious way. In MRSs, multisets of entities are used to represent the state of a dynamic system. Thus, in the following, we use the terms *state* and *multiset of entities* interchangeably.

Typically, MRSs consider only *flat* (unstructured) entities. Here, we use *structured* entities: Each entity is a map of property names $\mathcal{K}$ to values $\mathcal{V}$, i.e. a partial function $\mathcal{E} = \mathcal{K} \nrightarrow \mathcal{V}$. Structured entities are necessary for the scenarios we are considering, as they contain entities with multiple, possibly continuous, properties.

For example, consider the following multiset, that describes a situation in a predator-prey model, with ten predators and six prey, each entity having a specific age[1]:

$$[\![\, 6\langle \mathrm{T}\colon \mathrm{Prey}, \mathrm{A}\colon 2\rangle,\ 3\langle \mathrm{T}\colon \mathrm{Pred}, \mathrm{A}\colon 3\rangle,\ 7\langle \mathrm{T}\colon \mathrm{Pred}, \mathrm{A}\colon 5\rangle \,]\!] \tag{1}$$

In [12], a factorized representation of such states is devised, that allows to represent state distributions more compactly. We note that the concepts presented in the following also apply to the factorized representation, but we omit it here for readability.

The general concept of a multiset rewriting system (MRS) is to model the system dynamics by **actions** (also known as rewriting rules) that describe preconditions and effects of the possible behaviors of the entities. An action is a triple (c, e, w) consisting of a precondition list $c \in \mathcal{C}$, an effect function $e \in \mathcal{F}$ and a weight $w \in \mathbb{R}$. In conventional MRSs (e.g. in the context of P Systems [1, 16, 5]), the preconditions are typically a multiset or a list of (flat) entities. However, when using structured entities, preconditions can be described much more concisely as *constraints* on entities, i.e. as a list of boolean functions: $\mathcal{C} = [\mathcal{E} \to \{\top, \bot\}]$. For example, consider an action *reproduce*, that can be performed by any entity with Age > 3, regardless of other properties of the entity, which is naturally and concisely represented as a constraint.

The idea of applying an action to a state is to *bind* entities to the preconditions. Specifically, one entity is bound to each element in the precondition list,

[1] we use $\langle\cdot\rangle$ to denote partial functions

and entities can only be bound when they satisfy the corresponding constraint. The effect function then manipulates the state based on the bound entities (by inserting, removing, or manipulating entities). We call such a binding **action instance** $(a, i) \in \mathcal{I}$, i.e. a pair consisting of an action and a list of entities. We write $a(i)$ for an action instance consisting of an action a and bound entities i. Note that we use *positional* preconditions, i.e. the action instances eat(x,y) and eat(y,x) are different – either x or y is eaten.

A Compound Action $k \in \mathcal{K}$ is a multiset of action istances. It is applied to a state by composing the effects of the individual action instances. The compound action k is *applicable* in a state s if all of the bound entities are present in s, and it is *maximal* with respect to s if all entities in s are bound in k. Thus, a compound action is applicable and maximal when the multiset of all the bound entities is exactly the state s, i.e. $\biguplus_{a(x)\in k} x = s$. In the following, we are only concerned with applicable maximal compound action (AMCAs), which define the transition model. Scenarios where agents can also choose to not participate in any action can be modelled by introducing explicit "no-op" actions.

Compound Action Probabilities: Our system is probabilistic, which means that each AMCA is assigned a probability. In general, any function from the AMCAs to the positive real numbers which integrates to one is a valid definition of these probabilities, that might be plausible for different domains. Here, we use the probabilities that arise when each entity independently chooses which action to participate in (which is the intended semantics for the scenarios we are concerned with). To calculate this probability, we count the number of ways specific entities from a state s can be chosen to be assigned to the action instances in the compound action. This concept to calculate probabilities is closesly related to [1] – except that due to the fact that we use positional preconditions, the counting process is slightly different.

The *multiplicity* $\mu_s(k)$ of a compound action k with respect to a state s is the number of ways the entities in k can be chosen from s. See Example 1 below for an illustration of the calculation of the multiplicity.

The weight $v_s(k)$ of a compound action is the product of its multiplicity and the actions' weights:

$$v_s(k) = \mu_s(k) * \Pi_i w_i^{n_i} \tag{2}$$

Here, n_i is the number of action instances ai_i present in k. The probability of a compound action in a state s is its normalized weight:

$$p(k|s) = v_s(k) / \sum_{k_i} v_s(k_i) \tag{3}$$

Transition Model: The distribution of the AMCAs define the distribution of successor states, i.e. the *transition model*. The successor states of s are obtained by applying all AMCAs to s. The probability of each successor state s' is the sum of the probabilities of all AMCAs leading to s':

$$p(S'{=}s'|S{=}s) = \sum_{\{k|apply(k,s)=s'\}} p(k|s) \tag{4}$$

Finally, the posterior state distribution is obtained by applying the transition model to the prior state distribution, and marginalizing s (this is the standard predict step of Bayesian filtering):

$$p(S'{=}s') = \sum_s p(S{=}s)\, p(S'{=}s'|S{=}s) \tag{5}$$

Example 1: *In a simplified population model, two types of entities exist: Prey* $x = \langle Type = X \rangle$ *and predators* $y = \langle Type = Y \rangle$*. Predators can eat other animals (prey or other predators, action* e*), and all animals can reproduce (action* r*). Reproduction is 4 times as likely as consumption, i.e. action* e *has weight* 1*, and* r *has weight* 4*.*

For a state $s = [\![1x, 2y]\!]$*, the following applicable action instances exist:* $r(y)$*,* $r(x)$*,* $e(y, x)$*,* $e(y, y)$*. The resulting applicable maximal compound actions are:* $k_1 = [\![2r(y), 1r(x)]\!]$*,* $k_2 = [\![1e(y, y), 1r(x)]\!]$ *and* $k_3 = [\![1e(y, x), 1r(y)]\!]$*. Applying these compound actions (assuming that they have the obvious effects) to the initial state* s *yields the three successor states* $s'_1 = [\![4y, 2x]\!]$*,* $s'_2 = [\![1y, 2x]\!]$ *and* $s'_3 = [\![2y]\!]$*. The multiplicities of the compound actions are* $\mu_s(k_1) = 1$*,* $\mu_s(k_2) = 2$*,* $\mu_s(k_3) = 2$ *and their weights are* $v_s(k_1) = 1 * 4^3 = 64$*,* $v_s(k_2) = 2 * 1 * 4 = 8$ *and* $v_s(k_3) = 2 * 1 * 4 = 8$*.*

3 Efficient Implementation

In this section, we present the main contribution of this paper: An efficient approximate algorithm for computing the posterior state distribution (Equation 5).

Given a prior state distribution $p(S)$ and a set of actions A, the following steps need to be performed for each s with $p(S{=}s) > 0$ to obtain the posterior state distribution:

i Compute all action instances of each action $a \in A$, given s.
ii Compute all AMCAs and their probabilities (Equation 3).
iii Calculate the probabilities of the resulting successor states s', i.e. $p(s'|s)$, by applying all AMCAs to s (Equation 4).

Afterwards, the posterior state distribution $p(s')$ is obtained by weighting $p(s'|s)$ with the prior $p(s)$ and marginalizing s (Equation 5). In the following, we discuss efficient implementations for each of these steps.

Step (i) requires, for each action $(c, e, w) = a \in A$, to enumerate all bindings (lists of entities) that satisfy the precondition list $c = [c_1, \ldots, c_n]$ of this action, i.e. the set $\{[e_1, \ldots, e_n] \mid c_1(e_1) \wedge \ldots \wedge c_n(e_n)\}$. This is straightforward, as for each constraint, we can enumerate the satisfying entities independently. In the scenarios we are considering, the number of actions, as well as the number of *different* entities in each state is small (see Example 1). Furthermore, we only consider constraints that can be decided in constant time (e.g. comparisons with constants). Thus, we expect this step to be sufficiently fast.

Steps (ii) and (iii) are, however, computationally demanding, due to the large number of compound actions: Given a state s, let n be the total number

of entities in s and i be the number of action instances. The number of possible compound actions is at most the multiset coefficient $\left(\!\binom{i}{n}\!\right) = \frac{(i+n-1)!}{n!\,(i-1)!}$.

Therefore, in the following, we focus on the efficient computation of $p(K|s)$. We start with an exact algorithm that enumerates all AMCAs, and, based on that, derive a sampling-based algorithm that approximates $p(K|s)$.

In the context of other PPMRSs, efficient implementations for computing $p(K|s)$ have not been discussed. Either, they use a semantics that allows to sample a compound action by sequentially sampling the individual actions[2] [16], or they use a semantics similar to ours (requiring to enumerate all compound actions), but are not concerned with an efficient implementation [1, 5].

3.1 Exact Algorithm

The task we have to solve is the following: Given a set of action instances $(a, i) \in I$ and a state s, compute the distribution $p(K|s)$ of the compound actions that are applicable and maximal with respect to s (the AMCAs), as shown in Equation 3. To compute the partition function of this distribution exactly, it is necessary enumerate all AMCAs and their weights. Thus, the exact algorithm works as follows: First, all AMCAs are enumerated, which then allows to compute the partition function and thus $p(K|s)$.

In the following, we show how the AMCA calculation problem can be transformed into a constraint satisfaction problem (CSP) Γ, such that each solution of the CSP is an AMCA, and vice versa. Then, we only need to compute all solutions of Γ, e.g. by exhaustive search.

A CSP Γ is a triple (X, D, C) where X is a set of variables, D is a set of domains (one for each variable), and C is a set of constraints, i.e. boolean functions of subsets of X. Given action instances I and a state s, a CSP Γ is constructed as follows:

- For each action instance $(a, i) \in I$, there is a variable $x \in X$. The domain of x is $\{0, \ldots, \min_{e \in i}(n_e)\}$, where n_e is the multiplicity of entity e in s.
- For each entity $e \in s$ with multiplicity n_e in s, there is a constraint $c \in C$ on all variables x_i whose corresponding action instances a_i bind e. Let $m_{i,e}$ the number of times the action instance a_i binds e. The constraint then is $\sum_i m_{e,i} = n_e$. This models the applicability and maximality of the compound actions.

Note that the constraint language consists only of summation and equality, independently of the constraint language of action preconditions (which have been resolved before, when computing action instances).

A solution σ of Γ is an assignment of all variables in X that satisfies all constraints. Each solution σ of Γ corresponds to a compound action k: The value $\sigma(x)$ of a variable x indicates the multiplicity of the corresponding action instance (a, i) in k. Each solution σ corresponds to an *applicable* and *maximal*

[2] Due to the sequential sampling process, the probability of a compound action is higher when there are more possible permutations of the individual actions, which is explicitly avoided by our approach.

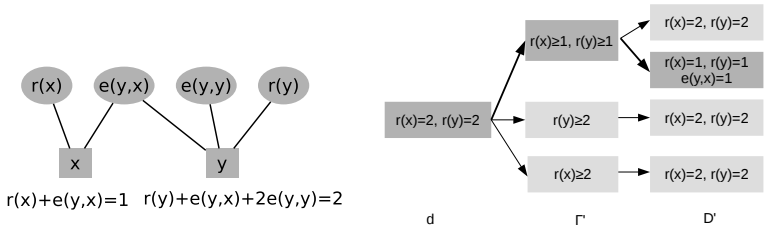

Fig. 1: Left: The CSP for Example 1. Circles represent variables, rectangles represent constraints. Right: Illustration of the proposal function, using the CSP of Figure 1 and the solution $d = (r(x) = 2, r(y) = 2)$. Equalities represent assignments in the solution, and inequalities represent constraints. Assignments and constraints with 0 are not shown.

compound action (as this is directly enforced by the constraints of Γ), and each AMCA is a solution of Γ. Figure 1 (left) shows the CSP corresponding to Example 1.

We use standard backtracking to enumerate all solutions of the CSP[3]. Afterwards, the weight of each solution (and thus the partition function) can be calculated.

Note that the CSP we are considering is *not* an instance of a *valued* (or *weighted*) CSP [6, 17]: They assume that each satisfied constraint has a value, and the goal is to find the *optimal* variable assignment, whereas in our proposal, only *solutions* have a value, and we are interested the *distribution* of solutions.

3.2 Approximate Algorithm

The exact algorithm has a linear time complexity in the number of AMCAs (i.e. solutions of Γ). However, due to the potentially very large number of AMCAs, enumerating all solutions of Γ is infeasible in many scenarios.

We propose to solve this problem by *sampling* CSP solutions instead of enumerating all of them. However, sampling directly is difficult: To compute the probability of a solution (Equation 3), we first need to compute the partition function, which requires a complete enumeration of the solutions.

Metropolis-Hastings-Algorithm: Markov chain Monte Carlo (MCMC) algorithms like the Metropolis-Hastings algorithm provide an efficient sampling mechanism for such cases, where we can directly calculate a value $v(k)$ that is proportional to the probability of k, but obtaining the normalization factor (the partition function) is difficult. The Metropolis-Hastings algorithm works by constructing a Markov chain of samples $\mathcal{M} = k_0, k_1, \ldots$ that has $p(K)$ as its stationary distribution. The samples are produced iteratively by employing a *proposal*

[3] This is sufficient, as the problem here is not that *finding* each solution is difficult, but that there are factorially many solutions.

distribution $g(k'|k)$ that proposes a move to the next sample k', given the current sample k. The proposed sample is either *accepted* and used as the current sample for the next iteration, or rejected and the previous sample is kept. The *acceptance probability* is calculated as $A(k,k') = \min\{1, (v(k')\, g(k|k'))/(v(k)\, g(k'|k))\}$. It can be shown that the Markov chain constructed this way does indeed have the target distribution $p(K)$ (Equation 3) as its stationary distribution [10]. The Metropolis-Hastings algorithm thus is a random walk in the sample space (in our case, the space of AMCAs, or equivalently, solutions of Γ) with the property that each sample is visited with a frequency relative to its probability. The Metropolis-Hastings sampler performs the following steps at time $t+1$:

1. Propose a new sample k' by sampling from $g(k'|k_t)$.
2. Let $k_{t+1} = k'$ with probability $A(k', k_t)$.
3. Otherwise, let $k_{t+1} = k_t$.

Proposal Function: In the following, we present a proposal function of compound actions. The idea is to perform *local* moves in the space of the compound actions as follows: The proposal function $g(k'|k)$ proposes k' by randomly selecting n action instances to delete from k, and sample one of the possible completions of the remaining (non-maximal) compound action. This means the proposal makes small changes to k for proposing k', while ensuring that k' is applicable and maximal.

The proposal function can be formulated equivalently when viewing compound actions as CSP solutions. For a CSP solution σ, "removing" a single action instance is done by removing the assignment of the corresponding variable in σ, and "remembering" the previous value of the variable as a constraint, relaxed by 1: Suppose that we want to remove an action instance corresponding to the CSP variable x, and the solution contains the assignment $x = v$. We do this by removing the assignment, and adding $x \geq v - 1$ as a constraint. This is done randomly for n variables of the CSP. Similarly, for all other variables, we add constraints $x \geq v$ to capture the fact that the remaining CSP can have solutions where these variables have a higher value. In Algorithm 1, a procedure is shown that enumerates all CSPs that can be obtained this way. From the resulting CSPs, one CSP Γ' is sampled uniformly, and then a solution σ' of Γ' is sampled (also uniformly). Notice that each of these CSPs is much easier to solve by backtracking search than the original CSP, as the solution space is much smaller. The proposal function is shown in Algorithm 1.

For example, consider the CSP corresponding to Example 1 (Figure 1 left) and the solution $d = (r(x) = 2, r(y) = 2, e(y,y) = 0, e(y,x) = 0)$. Suppose we want to remove $n = 2$ action instances. This results in three possible reduced CSPs: Either two r(x), two r(y) or one r(x) and one r(y) are removed. The CSPs, and the possible solutions of each CSP are shown in Figure 1 (right).

Probability of a Step: We do not only need to sample a value from g, given σ (as implemented in Algorithm 1), but for the acceptance probability, we also need to calculate the probability of $g(\sigma'|\sigma)$, given σ' and σ. This is implemented by Algorithm 2. The general idea is to follow all possible choices of removed action instances, and count the number of choices that lead to σ'. In Algorithm

Algorithm 1 Proposal function

function G(Γ,σ,n)
  $\Gamma' \leftarrow$ uniform(REDUCEDCSPS(Γ,σ,n))
  $\sigma' \leftarrow$ uniform(ENUMSOLUTIONS(Γ')) ▷ Enumerate solutions of Γ', sample one
  return σ'
end function
function REDUCEDCSPS($\Gamma = (X, D, C)$,σ,n)
  for each x_i, add constraint $x_i \geq d_i$ to C
  $R \leftarrow$ set of all combinations with repetitions of variables in X with exactly n elements, where x_i occurs at most $\sigma(x_i)$ times
  for $r \in R$ **do**
    $C' \leftarrow$ same constraints as in C, but $\forall\, x \in X$: replace $x \geq v$ by $x \geq v - x\#r$[4]
    $G \leftarrow G \cup (X, D, C')$ ▷ Collect all reduced CSPs
  end for
  return (G)
end function

Algorithm 2 Probability of a step of the proposal function

function GPROB(σ',σ,Γ,n)
  $\forall\, x : rem(x) \leftarrow \max(0, \sigma'(x) - \sigma(x))$ ▷ Variable assignments that need to be reduced to get from σ to σ'.
  $G \leftarrow \{\Gamma' \in$ REDUCEDCSPS$(\Gamma, \sigma, n) \mid$ reductions that reduce each variable x at least $rem(x)$ times$\}$ ▷ Reduced CSPs that have d' as a solution
  $\forall\, \Gamma' \in G : n_{\Gamma'} \leftarrow \mid$ ENUMCSP$(\Gamma') \mid$ ▷ Number of CSP solutions for each Γ'
  $t \leftarrow \mid$ REDUCEDCSPS$(\Gamma, \sigma, n) \mid$ ▷ Total number of ways to reduce the CSP
  $p \leftarrow 1/t \sum_{\Gamma' \in R} 1/n_{\Gamma'}$ ▷ Calculate probability that σ' is sampled
  return p
end function

1, two random choices are performed: (i) Choosing one of the reduced CSPs Γ', and (ii) choosing one of the solutions of Γ'. In both cases, a uniform distribution is used. Therefore, it is sufficient to know the *number* of elements to choose from. Furthermore, only need to compute the solutions for those CSPs Γ' where σ' can be reached. Both considerations are exploited by Algorithm 2, leading to an increased efficiency.

Figure 1 (right) illustrates these ideas. Suppose the dark grey path has been chosen by the proposal function. The function $gProb(\sigma', \sigma, \Gamma, 2)$ then only has to compute the solutions of the single CSP Γ' in the dark grey path, as it is the only CSP that has σ' as a solution. The probability is calculated as $gProb(\sigma', \sigma, \Gamma, 2) = 1/3 * 1/2 = 1/6$.

[4] $x\#r$ denotes the number occurences of x in r

4 Experimental Evaluation

In this section, we investigate the performance of the approximate compound action computation algorithm in terms of computation time and accuracy by simulating a variant of a probabilistic Lotka-Volterra model that has a compound action semantics.

4.1 Experimental Design

The Lotka-Volterra model is originally a pair of nonlinear differential equations describing the population dynamics of predator (y) and prey (x) populations [11]. Such predator-prey systems can be modeled as a MRS [16, 8].

In contrast to previous approaches, we use a *maximally parallel* MRS to model the system, i.e. in our approach, multiple actions (reproduction and consumption) can occur between consecutive time steps. We introduce explicit no-op actions to allow entities to not participate in any action. Modeling the system like this can, for example, be beneficial for Bayesian filtering, where between observations (e.g. a survey of population numbers), a large number of actions can occur, but their order is not relevant. Figure 2 (left) shows an example of the development of the system over time, as modeled by our approach. It shows the expected behavior of stochastic Lotka-Volterra systems: Oscillations that become larger over time [14].

We compare the exact and approximate algorithms by computing the compound action distribution for a single state s of the predator-prey model, i.e. $p(K|s)$. We vary the number of predator and prey entities in s (2, 3, 5, 7, 15, 20, 25, 30, 40, 50, 60, 70) as well as the number of samples drawn by the approximate algorithm (1, ..., 30000).

The convergence of the approximate algorithm is assessed using the *total variation distance* (TVD). Let p be the true distribution, and let q_n be the distribution of the approximate algorithm after drawing n samples. The TVD is then

$$\Delta(n) = 1/2 \sum_s \mid p(s) - q_n(s) \mid$$

The *mixing time* $\tau(\epsilon)$ measures how many samples need to be drawn until the TVD falls below a threshold ϵ:

$$\tau(\epsilon) = \min\{t \mid \Delta(n) \leq \epsilon \text{ for all } n \geq t\}$$

We assess the TVD and mixing time of (i) the compound action distribution, and (ii) of the state transition distribution. The rationale here is that ultimately, only the successor state distribution is relevant, but assessing the TVD and mixing time of the compound action distribution allows further insight into the algorithm.

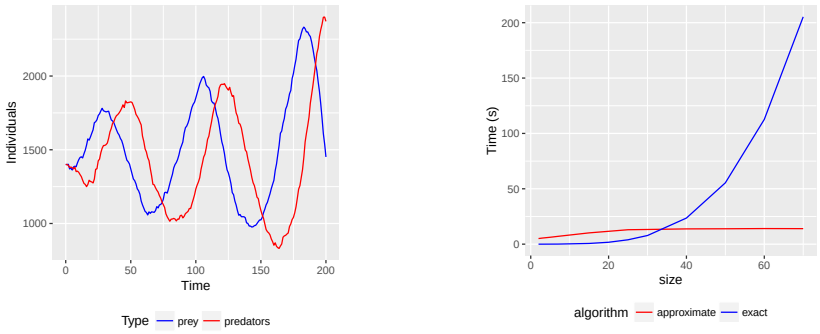

Fig. 2: Left: Sample trajectory, each state transition is obtained by calculating the compound action distribution using the approximate algorithm with 10,000 samples, and then sampling and executing one of the compound actions. Right: Runtime of the algorithms, using constant number of 10,000 samples.

4.2 Results

Figure 2 (right) shows the runtime of the exact and approximate algorithm (with a fixed number of 10,000 samples) for different numbers of entities in s. The exact algorithm is faster for states with only few entities, as solutions of only a single CSP are enumerated, whereas the approximate algorithm enumerates solutions for 10,000 CSPs (although each of those CSPs has only few solutions). However, the runtime of the approximate algorithm does not depend on the number of entities in s at all, as long as the number of samples stays constant (but approximation quality will decrease, as investigated later). In our scenario, the approximate algorithm is faster for states consisting of 40 or more predator and prey entities.

The difference between the exact and approximate compound action distribution $p(K|s)$ in terms of TVD is shown in Figure 3 (left). When more samples are drawn by the approximate algorithm, the TVD converges to zero, as expected (implying that the approximate algorithm works correctly). Naturally, the TVD converges slower for states with more entities (due to the much larger number of compound actions).

Eventually, we are interested in an accurate approximation of the distribution $p(S'|s)$. Figure 3 (middle) shows that this distribution can be approximated more accurately than $p(K|s)$: For a state with 70 predator and prey entities (with more than 9 million compound actions), the approximate transition model is reasonably accurate (successor state $TVD < 0.1$) after drawing 10,000 samples. Even more, Figure 2 (left) suggests that this approximation is still reasonable for states with more than 2,000 entities – as we still observe the expected qualitative behavior.

Figure 3 (right) shows the empirical mixing time of $p(S'|s)$. The mixing time grows approximately linear in the number of entities in the state. This suggests that to achieve the same accuracy of the approximation, the runtime of the

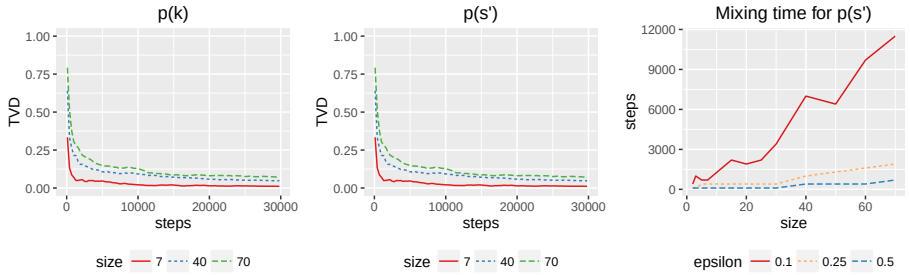

Fig. 3: TVD of $p(K|s)$ (left) and $p(S'|s)$ (middle) for different numbers of samples and for states with different number of entities. Right: Empirical mixing time of $p(S'|s)$, indicating that a linear increase in samples (and thus, runtime) of the approximate algorithm is sufficient to achieve the same approximation quality.

approximate algorithm only has to grow linearly – as compared to the exact algorithm, which has a factorial runtime.

Thus, using the approximate algorithm, it is possible to accurately calculate the successor state distribution, for situations with a large number of entities, even when the exact algorithm is infeasible.

5 Conclusion

In this paper, we investigated the problem of efficiently computing the compound action distribution (and thus, the state transition distribution, or transition model) of a probabilistic parallel Multiset Rewriting System (PPMRS) – which is required when performing Bayesian filtering (BF) in PPMRSs. We showed that computing the transition model exactly is infeasible in general (due to the factorial number of compound actions), and provided an approximation algorithm based on MCMC methods. This strategy allows to *sample* from the compound action distribution, and is therefore also useful for simulation studies that employ PPMRSs. Our empirical results show that the approach allows BF in cases where computing the exact transition model is infeasible – where the state contains thousands of entities.

Future work includes applying the approach to BF tasks with real-world sensor data, e.g. for human activity recognition. It may also be worthwhile to further investigate the general framework developed in this paper – approximating the solution distribution of a CSP that has probabilistic (or weighted) solutions – and see whether it is useful for other problems beyond compound action computation.

References

1. Barbuti, R., Levi, F., Milazzo, P., Scatena, G.: Maximally Parallel Probabilistic Semantics for Multiset Rewriting. Fundamenta Informaticae 112(1), 1–17 (2011)
2. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Science 96(1), 217–248 (1992), http://portal.acm.org/citation.cfm?doid=96709.96717
3. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: Distribution-aware sampling and weighted model counting for sat. In: AAAI. vol. 14, pp. 1722–1730 (2014)
4. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artificial Intelligence 172(6-7), 772–799 (2008)
5. Ciobanu, G., Cornacel, L.: Probabilistic transitions for P systems. Progress in Natural Science 17(4), 432–441 (2007)
6. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174, 449–478 (2010), http://linkinghub.elsevier.com/retrieve/pii/S0004370210000147
7. Ermon, S., Gomes, C., Sabharwal, A., Selman, B.: Taming the curse of dimensionality: Discrete integration by hashing and optimization. In: International Conference on Machine Learning. pp. 334–342 (2013)
8. Giavitto, J.L., Michel, O.: Mgs: A rule-based programming language for complex objects and collections. Electronic Notes in Theoretical Computer Science 59(4), 286–304 (2001)
9. Gogate, V., Dechter, R.: Samplesearch: Importance sampling in presence of determinism. Artificial Intelligence 175(2), 694–729 (2011)
10. Häggström, O.: Finite Markov chains and algorithmic applications, vol. 52. Cambridge University Press (2002)
11. Lotka, A.J.: Analytical Theory of Biological Populations. Springer Science & Business Media (1998)
12. Lüdtke, S., Schröder, M., Bader, S., Kersting, K., Kirste, T.: Lifted Filtering via Exchangeable Decomposition. ArXiv e-prints (2018), https://arxiv.org/abs/1801.10495
13. Oury, N., Plotkin, G.: Multi-level modelling via stochastic multi-level multiset rewriting. Mathematical Structures in Computer Science 23, 471–503 (2013)
14. Parker, M., Kamenev, A.: Extinction in the Lotka-Volterra model. Physical Review E 80(2) (2009), https://link.aps.org/doi/10.1103/PhysRevE.80.021129
15. Paun, G.: Membrane Computing: An Introduction. Springer Science & Business Media (2012)
16. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic P systems. International Journal of Foundations of Computer Science 17(01), 183–204 (2006)
17. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: Procedings of the International Joint Conference on Artificial Intelligence (1995)
18. Schröder, M., Lüdtke, S., Bader, S., Krüger, F., Kirste, T.: LiMa: Sequential Lifted Marginal Filtering on Multiset State Descriptions. In: KI 2017: Advances in Artificial Intelligence. Springer International Publishing AG (2017)
19. Wei, W., Selman, B.: A new approach to model counting. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 324–339. Springer (2005)