Gaussian Lifted Marginal Filtering

Stefan Lüdtke[1], Alejandro Molina[2], Kristian Kersting[2], and Thomas Kirste[1]

[1]Institute of Visual and Analytic Computing, University of Rostock, Germany
{stefan.luedtke2, thomas.kirste}@uni-rostock.de
[2]Computer Science Department, TU Darmstadt, Germany
{molina, kersting}@cs.tu-darmstadt.de

Abstract. Recently, *Lifted Marginal Filtering* has been proposed, an efficient Bayesian filtering algorithm for stochastic systems consisting of multiple, (inter-)acting agents and objects (entities). The algorithm achieves its efficiency by performing inference jointly over groups of *similar* entities (i.e. their properties follow the same distribution).
In this paper, we explore the case where there are no entities that are directly suitable for grouping, which is typical for many real-world scenarios. We devise a mechanism to identify entity groups, by formulating the distribution that is described by the grouped representation as a *mixture* distribution, such that the parameters can be fitted by Expectation Maximization. Specifically, in this paper, we investigate the Gaussian mixture case. Furthermore, we show how Gaussian mixture merging methods can be used to prevent the number of groups from growing indefinitely over time. We evaluate our approach on an activity prediction task in an online multiplayer game. The results suggest that compared to the conventional approach, where all entities are handled individually, decrease in prediction accuracy is small, while inference runtime decreases significantly.

Keywords: Lifted Inference · Probabilistic Inference · Bayesian filtering · Probabilistic Multiset Rewriting System · Gaussian Mixture · Activity Recognition

1 Introduction

Many AI tasks like Human Activity Recognition (HAR) or network analysis involve modeling stochastic systems that consist of multiple, interacting agents or objects. Recently, a novel inference approach (Lifted Marginal Filtering [7]) has been devised, that performs efficient inference in such systems. It models system states as *multisets*, and the system dynamics as a *Probabilistic Multiset Rewriting System* (PMRS) [2]. The algorithm is efficient when the system consists of groups of entities that are *similar*, i.e. whose properties follow the same distribution, and that have the same capabilities to interact. In this case, inference complexity only depends on the number of groups, but not on the number of entities per group. Many Bayesian filtering problems consist of such interacting entities, but often, it is not immediately clear which entities are suitable for grouping

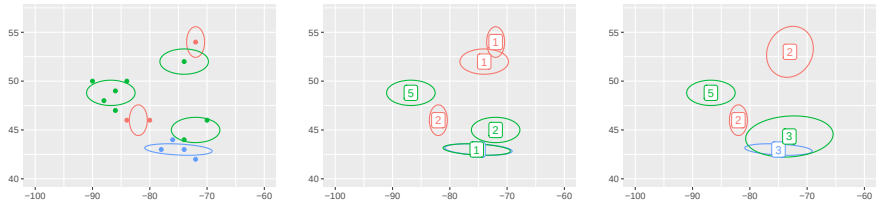

Fig. 1: Proposed approach for efficient inference multi-entity scenarios, illustrated by the multiplayer game domain from Example 1. Different colors denote different players. Left: Actual village positions are shown by dots. We do not use the actual positions, but Gaussian mixture components of village locations, shown by ellipses. Center: Example of state after several conquer actions occurred. Right: New mixture model, after merging of mixture components (to avoid growing the state space indefinitely).

– because all of them may have distinct properties. For example, consider the online multiplayer game we are concerned with in this paper (Figure 1), whose system dynamics can be conveniently described by a PMRS: Players own villages arranged on a grid map, and can perform actions like conquering other players' villages. The task is to estimate the distribution over future game states (i.e. to perform Bayesian filtering). However, initially, all entities (villages) have distinct properties (the position on the map), and thus Bayesian filtering in this system quickly leads to a very large number of possible states.

In this paper, we investigate methods to find groups of entities that can be handled jointly. The basic observation is that in many cases, some of the entities show almost the same behavior, despite having different properties. The idea is to approximate their properties by a single parametric distribution, such that the entities can be handled as a group. We call the state representation where entities are grouped *lifted state*[1]. Ideally, the lifted state still contains the relevant information to compute the posterior state distribution. Furthermore, we show how methods from Gaussian mixture merging [12] can be used to prevent the number of groups (and thus inference complexity) from growing indefinitely over time.

The empirical results in the online game domain show that our approach is much faster than the conventional approach of working with distinct entities (as done for the same game by [15]). Additionally, in this domain, the approximation that is performed to arrive at a lifted state representation has only marginal influence on the prediction accuracy.

2 Lifted Marginal Filtering

In this section, we briefly introduce LiMa, a lifted recursive Bayesian estimation algorithm based on Multiset Rewriting. We show how distributions over multi-

[1] Based on the connections to our approach to lifted probabilistic inference [5], as outlined in [7].

sets can be factorized, such that (i) entities in the multisets whose properties follow the same distribution can be grouped, and (ii) the distributions of those properties are represented on the parametric rather than on the instance level. We then show how recursive Bayesian estimation can be performed directly on this lifted representation, without computing the original, much larger ground representation first.

2.1 Problem Statement

We are concerned with a *recursive Bayesian estimation* task (also called *Bayesian filtering*). That is, the goal is to estimate the posterior distribution $p(X_t|y_{1:t})$ of the *hidden* system state x_t at time t from the previous posterior $p(X_{t-1}|y_{1:t-1})$ and an observation y_t. We assume that the system dynamics is a first-order Markov chain (i.e. it can be described by a *transition model* $p(X_{t+1}|x_t)$), and that the observation y_t only depends on the state x_t. Then, the computation can be decomposed into two steps: The *prediction* step calculates the distribution after applying the transition model, i.e. $p(X_{t+1}|y_{1:t}) = \int_{x_t} p(X_{t+1}|X_t{=}x_t)\, p(X_t{=}x_t|y_{1:t})\, dx_t$. Afterwards, the posterior distribution is computed by employing the *observation model* $p(y_{t+1}|X_{t+1})$:

$$p(X_{t+1}|y_{1:t+1}) = \frac{p(y_{t+1}|X_{t+1})\, p(X_{t+1}|y_{1:t})}{p(y_{t+1}|y_{1:t})} \tag{1}$$

In LiMa, the states x_t are *multisets*, i.e. functions of *entities* to *multiplicities* (natural numbers). We use *structured* entities here, i.e. entities are key-value maps. The transition model $p(X_{t+1}|x_t)$ is modeled as a *Probabilistic Multiset Rewriting System*, introduced in more detail in Section 2.3. The following example illustrates how system states can be modeled as multisets.

Example 1 Consider the situation depicted in Figure 1 (left): There are three players, each one having multiple villages. Each village is represented as a separate entity, containing owner and position information, for example:

$$\begin{aligned} c_1 &= \langle \text{Player: } \mathit{Red}, \text{Pos: } (-72, 54) \rangle \\ c_2 &= \langle \text{Player: } \mathit{Red}, \text{Pos: } (-84, 46) \rangle \\ &\dots \\ c_{15} &= \langle \text{Player: } \mathit{Green}, \text{Pos: } (-74, 43) \rangle \end{aligned}$$

The state is a multiset of those entities, each entity having a multiplicity of one:

$$x = [\![\, 1c_1, 1c_2, \dots, 1c_{15} \,]\!]$$

Modeling a distribution over such multisets naively as a categorical distribution (i.e. a set of tuples (x_i, p_i)) quickly becomes infeasible, due to the combinatorial explosion in the number of tuples with respect to the number of entities. When m different types of entities (species) can exist in the system, and n entities are present at a given time, the maximum number of states is

$\binom{m+n-1}{n} = \frac{(m+n-1)!}{n!\,(m-1)!}$, i.e. the number of multisets of cardinality n, with elements chosen from a set of cardinality m. Note that enumeration becomes impossible when entities can contain continuous values, e.g. from $\mathbb{R}$.

2.2 Lifted Representation

Lifted Marginal Filtering (LiMa) uses a more efficient representation for such distributions over structured multisets. Specifically, LiMa can efficiently represent situations where multiple entities are similar, i.e. their properties follow the same distribution, and efficiently perform inference in such domains. The idea is that in these cases, the distribution of properties can be represented *parametrically*, and the similar entities can then be grouped in the multiset. To make this more clear, consider the following example.

Example 2 For the online game domain introduced in Example 1, consider the situation where the *Red* player owns two villages, and we are uncertain about whether they are at location 1 or location 2 (both villages could be either at 1 or 2 with uniform probability). Using the representation above, this requires 3 distinct ground states

$$
\begin{aligned}
x_1 &= [\![\, 2\langle \text{P: } \mathit{Red}, \text{Pos: } 1\rangle \,]\!] \\
x_2 &= [\![\, 1\langle \text{P: } \mathit{Red}, \text{Pos: } 1\rangle, 1\langle \text{P: } \mathit{Red}, \text{Pos: } 2\rangle \,]\!] \\
x_3 &= [\![\, 2\langle \text{P: } \mathit{Red}, \text{Pos: } 2\rangle \,]\!]
\end{aligned}
$$

with probabilities $p(x_1) = p(x_3) = 0.25$ and $p(x_2) = 0.5$. However, by using a *parametric distribution* to represent the location of both entities, we can represent the same situation as:

$$
s = [\![\, 2\langle \text{P: } \delta_{\text{Red}}, \text{Pos: } \mathcal{C}(0.5{:}1, 0.5{:}2)\rangle \,]\!]
$$

Here, $\mathcal{C}(0.5{:}1, 0.5{:}2)$ represents the categorical distribution, where both 1 and 2 are drawn with probability 0.5, and δ_{Red} represents the singleton distribution with value *Red*.

We call such a multiset, which contains entitites that map properties to representations of a distribution, a *lifted state*. The number of lifted states that are required to represent a distribution can be much lower than the number of ground states, as each lifted state already describes a distribution of *multiple* ground states. In general, it is also possible to maintain factors that represent joint distributions over values of multiple entities, see [7] for more details.

A lifted state s represents a distribution of ground states x by the following generative process: Fix an order of the entities in s (e.g. lexicographically), denote the i-th entity as s_i. For simplicity of the definitions, identical entities are repeated here. Denote the overall number of entities in s and x as n. The k-th distribution in s_i is called $p_{s_i}^{(k)}$. Then, for each distribution $p_{s_i}^{(k)}$, sample a value and replace the representation of $p_{s_i}^{(k)}$ in s by the sampled value. Finally, all entities created this way are collected in a multiset x.

For example, the lifted state $[\![2\langle \text{P: } \delta_{\text{Red}}, \text{Pos: } \mathcal{C}(0.5{:}1, 0.5{:}2)\rangle]\!]$ represents the distributions $p^{(1)}_{s_1} = p^{(1)}_{s_2} \sim \delta_{\text{Red}}$ and $p^{(2)}_{s_1} = p^{(2)}_{s_2} \sim \mathcal{C}(0.5{:}1, 0.5{:}2)$. Suppose we sample the value *Red* from both $p^{(1)}_{s_1}$ and $p^{(1)}_{s_2}$, 2 from $p^{(2)}_{s_1}$ and 1 from $p^{(2)}_{s_2}$. The resulting ground state is $x = [\![1\langle \text{P: } \mathit{Red}, \text{Pos: } 1\rangle, 1\langle \text{P: } \mathit{Red}, \text{Pos: } 2\rangle]\!]$. Note that the same ground state is also obtained by sampling 1 from $p^{(2)}_{s_1}$ and 2 from $p^{(2)}_{s_2}$. This is due to the fact that both x and s are multisets, where the entities do not have an order.

Next, we give a closed-form expression of the ground distribution $p(x|s)$ that is described by a lifted state. This will allow us to describe the relationship of the distribution to mixture models in more detail in Section 3. Assume we can fix an order of the entities in x (e.g. lexicographically), and call the j-th entity x_j. First, each lifted entity s_i describes a distribution $p(x_j|s_i)$ of ground entities. According to the sampling process, this is simply

$$p(x_j|s_i) = \prod_{k=1}^{K} p^{(k)}(x_j^{(k)}|s_i), \tag{2}$$

i.e. all distributions contained in a lifted entity s_i are independent. There are multiple ways how a given ground state x can be sampled from a lifted state s (i.e. which entity in x has been sampled from which entity in s). Thus, we need to consider all possible associations. Each association between entities from x and entities from s can be described by a permutation σ of $1, \ldots, n$: Given a permutation σ, the probability of the entity x can be defined as:

$$p(x|s, \sigma) = p(x_1, \ldots, x_n|s_1, \ldots, s_n, \sigma) = \prod_{j=1}^{n} p(x_j|s_{\sigma(j)}) \tag{3}$$

Finally, the distribution of ground states x is obtained by marginalizing over all permutations:

$$p(x|s) = \sum_{\sigma} p(x|s, \sigma)\, p(\sigma), \tag{4}$$

where $p(\sigma) = 1/n!$, i.e. a uniform distribution over permutations.

2.3 System Dynamics

The system dynamics, i.e. the transition model $p(s_t|s_{t-1})$ is described by a *Probabilistic Multiset Rewriting System* (PMRS). Notably, the transition model can be applied directly to a distribution of *lifted states*, without generating all ground states first: This is possible when the actions are *homogeneous* with respect to the lifted states s, i.e. all ground states x described by s are manipulated in the same way by the action, such that the posterior can again be represented by a lifted state s'.

An *action* a is a triple (c, f, w), where c is a list of preconditions, f is an effect function and w is a weight. It is useful to formulate the preconditions as *constraints*, i.e. boolean functions on the lifted entities. An action a is compatible

to a sequence $i = (i_1, \ldots, i_n)$ of entities when each constraint is satisfied by the corresponding entity. We call a pair of action and a compatible sequence of entities an *action instance*. An action instance can be applied to a lifted state s when all entities are contained in the state. The posterior state s' of applying an action instance is obtained by removing the entities from s, and inserting the results of the effect function f (applied to the entities): $s' = \text{apply}(s, (a, i)) = (s \uplus i) \uplus f(i)$.

An example of such an action in the multiplayer domain is the conquer(c_1, c_2) action, describing that a village c_2 is conquered by an attack starting from a village c_1. The precondition of this action is that the owners of c_1 and c_2 are different (a player cannot conquer a city of himself). Note that the preconditions are *positional* – the action instance conquer(c_1, c_2) is different from conquer(c_2, c_1).

Probabilistic MRSs assign a *weight* w_{ai} to each action instance ai. This weight is based on the property values of the bound entities. For example, the weight of a specific instance of the conquer action depends on the distance of the locations of c_1 and c_2. In the scenarios we are concerned with, all entities can act simultaneously between observations. For example, multiple players can conquer villages in a single time step. Formally, this is expressed by a *maximally parallel* MRS [2] (MPMRS). In such a system, each state transition is described by a parallel executed of a multiset of action instances, called *compound action*.

We use a generalization of MPMRSs to model this behavior: We distinguish between *exclusive* constraints, where the entity that is bound to such a constraint cannot participate in any other action, and *non-exclusive* constrains. For example, in the multiplayer domain, the same entity can conquer multiple other entities, but can only be conquered once per timestep. A compound action is applied to a state s by applying all of its action instances to s. A compound action is *valid* in a state s when all exclusiveness conditions are satisfied.

Given a state s, each valid compound action k can be assigned a weight, based on its *multiplicity* $\mu_s(k)$ (the number of ways the entities in k can be chosen from s), and the weights of its action instances: $w_s(k) = \mu_s(k) \prod_{ai \in k} w_{ai}$. The distribution of valid compound actions is the normalized weight: $p(k|s) = w_s(k) / \sum_{k_i \in K_s} w_s(k_i)$, where K_s is the set of all valid compound actions in s. As the number of valid compound actions can quickly become very large, we use the MCMC method proposed in [8] to approximate $p(k|s)$ here. Finally, the transition model (i.e. the distribution of successor states of a given state s_t) is obtained as:

$$p(s_{t+1}|s_t) = \sum_{\{k|s_{t+1}=apply(s_t,k)\}} p(k|s_t) \tag{5}$$

The transition semantics is illustrated by the following example.

Example 3 Consider the entities $e_1 = \langle \text{P: } \mathit{Red}, \text{Pos: } \mathcal{N}(\mu_1, \Sigma_1) \rangle$, $e_2 = \langle \text{P: } \mathit{Red}, \text{Pos: } \mathcal{N}(\mu_2, \Sigma_2) \rangle$ and $e_3 = \langle \text{P: } \mathit{Green}, \text{Pos: } \mathcal{N}(\mu_3, \Sigma_3) \rangle$, and the lifted state $s = [\![2e_1, 1e_2, 5e_3]\!]$. The only action is the conquer action described above. There are four action instances: $a_1 = \text{conquer}(e_1, e_3)$, $a_2 = \text{conquer}(e_3, e_1)$, $a_3 = \text{conquer}(e_2, e_3)$ and $a_4 = \text{conquer}(e_3, e_2)$. This results in a number of valid compound actions. For illustration, consider the compound action $k = [\![2a_1, 1a_4]\!]$.

The resulting successor state is

$$s = [\![\, 2\langle \text{P: } \mathit{Red}, \text{Pos: } \mathcal{N}(\mu_1, \Sigma_1)\rangle, 1\langle \text{P: } \mathit{Green}, \text{Pos: } \mathcal{N}(\mu_2, \Sigma_2)\rangle, \\ 3\langle \text{P: } \mathit{Green}, \text{Pos: } \mathcal{N}(\mu_3, \Sigma_3\rangle, 2\langle \text{P: } \mathit{Red}, \text{Pos: } \mathcal{N}(\mu_3, \Sigma_3\rangle \,]\!],$$

and its probability can be computed as shown in Equation 5 (specific values are not given in this example).

Afterwards, the probability $p(s_{t+1}|y_{1:t})$ of each posterior state s_{t+1} is reweighed by the observation likelihood $p(y_{t+1}|s_{t+1})$. The observations y can be continuous or discrete random variables, and $p(y_{t+1}|s_{t+1})$ can be any distribution, as long as we are able to compute its value, given a state s_{t+1} and an observation y_{t+1}. Note that this operation does not increase the representational complexity (as it is simply reweighing the probability of each state s_{t+1}. In this paper, we are only concerned with the *prediction* step (Equation 5), which can increase the representational complexity.

An important aspect to note here is that the number of compound actions (and thus the cardinality of the posterior distribution) depends on the number of action instances, and thus on the number of *different* entities. Thus, the lifted state representation is more efficient for two reasons: (i) The number of states is lower (each lifted state represents a distribution of ground states), and (ii) the number of successor states per lifted state is lower (as fewer compound actions can be applied).

3 Gaussian Lifted Marginal Filtering

In the following, we show how a lifted state representation can be obtained, given data of ground entities. Furthermore, we discuss how the representational size of the distribution can be prevented from growing indefinitely over time.

3.1 Lifting by Mixture Fitting

Given a ground state, the goal is to find a lifted state that approximates the ground states as close as possible, but has fewer different entities. Specifically, the goal is to find s such that the likelihood of s for the ground state x, i.e. $p(x|s)$, is maximized. Thus, we need to devise a maximum likelihood estimator for $p(x|s)$.

Consider again how $p(x|s)$ has been defined in Section 2.2: Overall, $p(x|s)$ is a sum over permutations, i.e. ways to associate the entities in x and in s. This is due to the fact that given a sample x (a ground state) from a lifted state s, we do not know which entity has been sampled from which entity in s. Instead of attempting to directly perform maximum likelihood estimation for the distribution $p(x|s)$, we instead make the following simplification: Instead of considering all *permutations* of $1, \ldots, n$, we consider all *tuples* $\tau \in T_n$ of length n, with elements chosen from $1, \ldots, n$. In other words, we omit the constraint

that we sample exactly once from each distribution in s. This way, the likelihood becomes

$$\tilde{p}(x|s) = \sum_{\tau} p(\tau) \prod_{j=1}^{n} p(x_j|s_{\tau(j)}) \tag{6}$$

i.e. it sums over even more terms. Again, $p(\tau)$ is a uniform distribution over all $\tau \in T_n$, i.e. $p(\tau) = 1/n^n = \prod_{j=1}^{n} 1/n$. Then, by using the identity $\sum_{\tau} \prod_{j=1}^{n} a_{j,\tau(j)} = \prod_{j=1}^{n} \sum_{i=1}^{n} a_{ij}$, we obtain

$$\tilde{p}(x|s) = p(x_1, \ldots, x_n|s) = \prod_{j=1}^{n} \sum_{i=1}^{n} \frac{1}{n} p(x_i|s_j). \tag{7}$$

Finally, remember that in s, there are multiple identical entities. Denote the multiplicity of entity s_j by n_j, and the number of different entities by m. Thus, the distribution can be written as

$$\tilde{p}(x|s) = p(x_1, \ldots, x_n|s) = \prod_{j=1}^{n} \sum_{i=1}^{m} \frac{n_j}{n} p(x_i|s_j). \tag{8}$$

This distribution describes a sequence of i.i.d. samples from a mixture, where $p(\cdot|s_j)$ are the mixture components and n_j/n is the weight of each mixture. The likelihood $\tilde{p}(x|s)$ can be maximized by standard mechanisms to fit mixture distributions. Specifically, for the online game domain, each distribution $p(\cdot|s_j)$ is a product of a singleton distribution (to encode the owner of a village), and a normal distribution (to encode the location of the distribution). Thus, for each value of the singleton distributions (i.e. for each player), a Gaussian mixture needs to be fitted, which can be done efficiently using Expectation Maximization (EM) [4].

The lifted state is constructed by generating one entity for each mixture component, with a value distribution according to that component. The multiplicity of each of those entities is the number of ground entities associated with this mixture component. As an example, consider Figure 1, showing the original village locations, and the estimated mixture components per player (and thus entities in the lifted state).

3.2 Merging Entities

Over time, the lifted representation that has been obtained can degenerate in multiple ways. First, the number of *states* typically increases over time due to the system dynamics, as each state can have multiple successor states, and some operations can require *splitting* a lifted states into multiple partitions (see [7] for more details). To prevent the representational size of the distribution from growing indefinitely, an operation that reduced the number of lifted states is required (e.g. by identifying multiple lifted states that can be represented by a single lifted state). Solving this problem in general is a topic for future work.

Here, we consider a closely related problem: The number of *distinct entities* (species) can also be increased due to the system dynamics, which in turn also leads to an increased number of posterior states. This is illustrated in Example 3. In other contexts, this problem is well-known: In a mixture Kalman filter [1], the number of mixture components grows exponential over time. For this method, *merging* procedures have been developed: Given a Gaussian mixture, they compute a mixture with fewer components that has a minimal distance to the original mixture.

For our approach, we employ the same concepts: After each transition, we apply a merging procedure that reduces the number of different entities (i.e. mixture components). Specifically, we employ the procedure described in [12]: Let s_i and s_j be two entities in a state s, with multiplicities n_i and n_j. Assume that they have associated value distributions $p_i \propto \mathcal{N}(\mu_i, \Sigma_i)$ and $p_j \propto \mathcal{N}(\mu_j, \Sigma_j)$. The Gaussian mixture of p_i and p_j is $p_m = w_i\, p_i + w_j\, p_j$, where $w_i = n_i/n$ and $w_j = n_j/n$. The parameters of the normal $p_{ij} \propto \mathcal{N}(\mu_{ij}, \Sigma_{ij})$ with minimal Kullback-Leibler divergence to p_m are calculated as follows:

$$\begin{aligned} w_{ij} &= w_i + w_j, \qquad w_{i|ij} = \frac{w_i}{w_i + w_j}, \qquad w_{j|ij} = \frac{w_j}{w_i + w_j} \\ \mu_{ij} &= w_{i|ij}\mu_i + w_{j|ij}\mu_j \\ \Sigma_{ij} &= w_{i|ij}\Sigma_i + w_{j|ij}\Sigma_j + w_{i|ij}w_{j|ij}(\mu_i - \mu_j)(\mu_i - \mu_j)^T \end{aligned} \tag{9}$$

The weight of the new component p_{ij} is w_{ij}, and the multiplicity of the new entity is $n_i + n_j$. Given a mixture, it is possible to find the *optimal* components to merge, i.e. those producing the smallest change in Kullback-Leibler divergence of the mixtures before and after merging. This is done by calculating the bound $B(i,j) = \frac{1}{2}\left[w_{ij}\log|\Sigma_{ij}| - w_i\log|\Sigma_i| - w_j\log|\Sigma_j|\right]$ for all combinations of components p_i and p_j and selecting the two components for which $B(i,j)$ is minimal. This operation is repeated, until either the minimal bound $B(i,j)$ exceeds a threshold (meaning that merging the mixture further would lead to a larger error), or the desired number of mixture components is reached. An example of this procedure is shown in Figure 1 (right).

4 Experimental Evaluation

We evaluated the proposed approach on a prediction task in the large-scale massively multiplayer online strategy game Travian [2]. Travian is a strategy game, consisting of a grid map. Players own one or multiple villages located on the map. They harvest resources from the environment, improve their village, build military units and attack other players' villages (possibly conquering them). In the following, we will focus on the high-level aspects of the game, namely village ownership and attacks. The same application scenario has already been investigated by Thon et al. [15], using a ground state representation.

[2] www.travian.com

Table 1: Factors and levels of experimental design.

Factor	Levels	Description
Entities per player	1,3,5,∞ (ground representation)	Maximum number of entities per player, i.e. mixture components
Timesteps	2,. . . ,5	Prediction horizon
Players	3,. . . ,10	Number of players per state
Merging	yes, no	Gaussian mixture reduction
Dataset	1,. . . ,10	Random subset of complete data

We logged the state of a game server over 5 days and recorded high-level data (position and affiliation of villages). The data was collected once every 24 hours. The data contains more than 6400 villages and 1400 players.

We evaluated our approach on a subset of this data, that has been extracted in the same way as done by Thon et al. [15]: From the complete data, sequences of *local* game world states were sampled. Each sequence contains the game state for a small set of players (3 to 10 players).

The goal of the experiments is to evaluate the performance of the proposed approach, regarding prediction accuracy and runtime. Specifically, we compare the lifted state representation to the conventional ground state representation. Multiple prediction steps (up to 5) are performed, without incorporating observations. After each prediction step, either Gaussian mixture merging is performed to the maximally allowed number, or no merging is performed (to investigate the effect of the merging operation on runtime and accuracy). The prediction accuracy is assessed by computing receiver operating characteristic (ROC) curves, regarding whether each conquer event did or did not occur in the transition, and by calculating the area under the curve (AUC). In all experiments, we used the MCMC-based algorithm to compute compound actions described in [8] with 100 samples, and performed 10 runs for each factor configuration. We use a prototypical implementation of our approach in R, and use the R package `mclust` [14] for fitting Gaussian mixtures by Expectation Maximization.

5 Results

Results of the evaluation are shown in Figure 2. The upper left plot shows exemplary ROC curves for the subsets containing 3 players. AUC is substantially greater than 0.5, indicating that all models capture some of the characteristics of the true system dynamics. As expected, the AUC is lowest when only a single mixture component is allowed per player, as this is the broadest approximation of the true village locations. The more components are allowed, the more information about the true village locations is preserved, resulting in a more accurate prediction.

Figure 2 (top right) shows the mean AUC (aggregated over all experiments of 3 to 10 players) for different numbers of mixture components, and a prediction

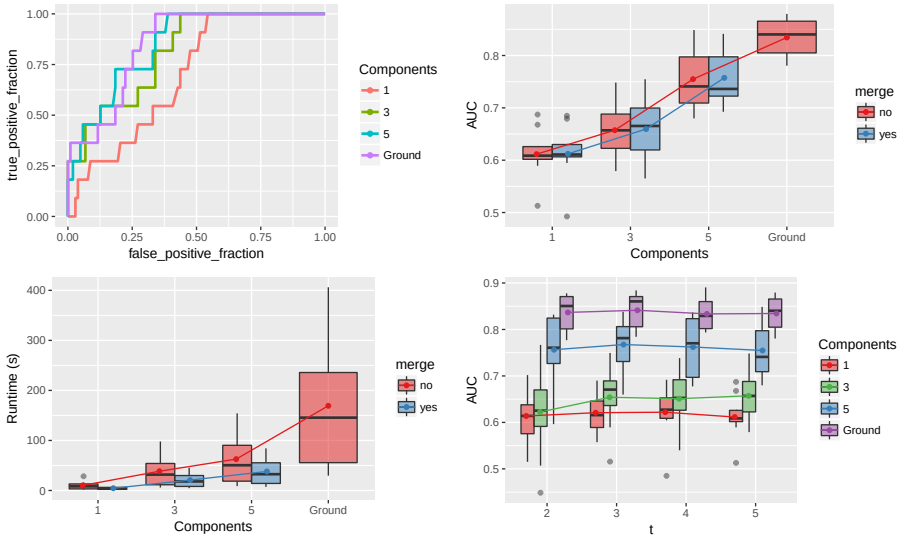

Fig. 2: Results for the Travian scenario for 5-timestep prediction. Top left: Exemplary ROC curves (3 players). Top right: AUC for different numbers of mixture components per player (aggregated over all runs and numbers of players). Bottom left: Runtime for different number of mixture components per player. Bottom right: AUC for different prediction horizons (aggregated over all runs and numbers of players).

horizon of 2. Allowing more components in general leads to a higher AUC, as discussed above. The obtained AUC values for the ground version of our approach are comparable to results reported by [15] (approximately 0.8 in both cases). Their approach is similar to the *ground* version of our approach, except that the transition model they use considers additional factors like history of players and alliances. Interestingly, the AUC for 5 components over all numbers of players is not significantly lower than the AUC of the ground model ($p > 0.12$ using Wilcoxon signed rank test, $n = 80$). Furthermore, for each number of mixture components, there is no significant difference in AUC between the runs where Gaussian mixture merging was performed or not performed.

Figure 2 (bottom left) shows the average runtime of computing the prediction. More components lead to an increased algorithm runtime (as the number of action instances increases). The ground model has by far the highest runtime. In contrast to the AUC, the runtime of the lifted models is significantly lower than runtime of the ground model ($p < 10^{-10}$ using Wilcoxon signed rank test, $n = 80$), and Gaussian mixture merging also leads to a significantly lower runtime.

To summarize, the results suggest that the lifted model can achieve an AUC that is not significantly lower than the AUC of the ground model (given a sufficiently large number of mixture components per entity), but needs a significantly lower runtime.

6 Related Work

The concept of factorizing the state distribution, and handling some factors analytically, is prominently and successfully used in the Rao-Blackwellized particle filter (RBPF) [3]. In the RBPF, the distribution of the variables that are not handled analytically is estimated by a particle filter. The distribution of these remaining variables has a lower dimensionality and smaller support. Thus fewer particles are necessary to provide a good approximation. In contrast to the RBPF, in our approach, the factorization has an additional advantage: The remaining object is a *multiset*, where more elements can be grouped, and inference on this multiset (using multiset rewriting) can be performed more efficiently.

There are a number of other approaches that use a computational language to describe system dynamics (just as we use a MRS), known as Computational State Space Models (CSSM) [11, 6]. We are not aware of any CSSM that uses a factorized state representation, as done by Lifted Marginal Filtering. Stochastic Relational Processes (SRP) [15] are a specific instance of CSSMs that describe states by logical interpretations, and state dynamics by a probabilistic logic. Their approach is conceptually similar to the *ground* version of our approach (although the semantics of parallel actions is different in detail).

The concept of grouping entities that are not exactly identical in a multiset is known as *super-individuals* [13, 10] in the modeling and simulation community. The idea is to maintain not all individual entities, but typical representatives, that each represent a large number of actual entities. In contrast to our approach, each of the super-individuals has *specific* property values (i.e. it its assumed that each of the entities represented by the super-individual has the same property values), whereas LiMa is able to represent (and manipulate) the *distribution* of property values. A further difference of LiMa to these approaches is that it directly estimates state *distributions*, instead of repeatedly drawing sample trajectories, which allows to incorporate observations and thus recursive Bayesian estimation.

7 Conclusion

In this paper, we investigated how Lifted Marginal Filtering (LiMa) can be applied to situations without exactly identical entities. The central idea is to formulate the distribution described by a lifted state as a mixture distribution, which allows to use Expectation Maximization to estimate the parameters of the mixture (i.e. of the lifted state). Inference can then be performed directly on this compact representation of the distribution, which is much more efficient than inference on the instance level. Specifically, in this paper we considered the special case where the value distributions are assumed to be normal distributions, and showed how methods for fitting Gaussian mixtures can be used to find a suitable grouping. The evaluation on an online multiplayer game suggests that using this approach, prediction accuracy decreases only marginally, while runtime is significantly lower.

The ideas developed in this paper can be extended in several directions: First, other parametric or non-parametric distributions (instead of normal distributions) could be used for the value distribution. Second, a more general approach to estimate the parameters of such mixtures – that include different types of distributions – is needed. Both of these requirements can potentially be satisfied by using the recently proposed *Mixed Sum-Product Networks* (MSPNs) [9]. An MSPN is a general density estimator, that allows to fit a hierarchical probabilistic model, consisting of layers of mixtures and factorization. Using MSPNs, we do not need any prior knowledge about which property values to approximate by distributions, or about the parametric form of the distribution.

A further direction for future work is concerned with keeping the representation of the posterior distribution small, by identifying *groups* of lifted states whose ground distribution can be represented (approximately) by a single lifted state. Finally, the experimental evaluation could be extended by integrating (partial) state observations (thus performing full Bayesian Filtering, the original task Lifted Marginal Filtering was developed for), a more elaborate transition model (for example based on the transition model used by [15]), or by investigating more complex application domains, like sensor-based human activity recognition.

References

1. Alspach, D., Sorenson, H.: Nonlinear bayesian estimation using gaussian sum approximations. IEEE transactions on automatic control **17**(4), 439–448 (1972)
2. Barbuti, R., Levi, F., Milazzo, P., Scatena, G.: Maximally Parallel Probabilistic Semantics for Multiset Rewriting. Fundamenta Informaticae **112**(1), 1–17 (2011). https://doi.org/10.3233/FI-2011-575
3. Doucet, A., De Freitas, N., Murphy, K., Russell, S.: Rao-Blackwellised particle filtering for dynamic Bayesian networks. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence. pp. 176–183. Morgan Kaufmann Publishers Inc. (2000)
4. Fraley, C., Raftery, A.E.: Model-based clustering, discriminant analysis, and density estimation. Journal of the American statistical Association **97**(458), 611–631 (2002)
5. Gogate, V., Domingos, P.: Probabilistic Theorem Proving. Commun. ACM **59**(7), 107–115 (2016). https://doi.org/10.1145/2936726
6. Krüger, F., Nyolt, M., Yordanova, K., Hein, A., Kirste, T.: Computational State Space Models for Activity and Intention Recognition. A Feasibility Study. PLOS ONE **9**(11), e109381 (2014). https://doi.org/10.1371/journal.pone.0109381
7. Lüdtke, S., Schröder, M., Bader, S., Kersting, K., Kirste, T.: Lifted Filtering via Exchangeable Decomposition. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (2018)
8. Lüdtke, S., Schröder, M., Kirste, T.: Approximate Probabilistic Parallel Multiset Rewriting using MCMC. In: KI 2018: Advances in Artificial Intelligence (2018)
9. Molina, A., Vergari, A., Di Mauro, N., Natarajan, S., Esposito, F., Kersting, K.: Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (2018)

10. Parry, H.R., Evans, A.J.: A comparative analysis of parallel processing and super-individual methods for improving the computational performance of a large individual-based model. Ecological Modelling **214**(2-4), 141–152 (2008)
11. Ramrez, M., Geffner, H.: Goal recognition over POMDPs: Inferring the intention of a POMDP agent. In: Proceedings of the 22nd IJCAI. pp. 2009–2014. AAAI Press (2011)
12. Runnalls, A.R.: Kullback-leibler approach to gaussian mixture reduction. IEEE Transactions on Aerospace and Electronic Systems **43**(3) (2007)
13. Scheffer, M., Baveco, J., DeAngelis, D., Rose, K., van Nes, E.: Super-individuals a simple solution for modelling large populations on an individual basis. Ecological modelling **80**(2-3), 161–170 (1995)
14. Scrucca, L., Fop, M., Murphy, T.B., Raftery, A.E.: mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. The R journal **8**(1), 289 (2016)
15. Thon, I., Landwehr, N., De Raedt, L.: Stochastic relational processes: Efficient inference and applications. Machine Learning **82**(2), 239–272 (2011). https://doi.org/10.1007/s10994-010-5213-8