Approximate Online Inference for Dynamic Markov Logic Networks

Thomas Geier, Susanne Biundo
Institute of Artificial Intelligence
Ulm University
Ulm, Germany
forename.surname@uni-ulm.de

***Abstract*—We examine the problem of filtering for dynamic probabilistic systems using Markov Logic Networks. We propose a method to approximately compute the marginal probabilities for the current state variables that is suitable for online inference. Contrary to existing algorithms, our approach does not work on the level of belief propagation, but can be used with every algorithm suitable for inference in Markov Logic Networks, such as MCSAT. We present an evaluation of its performance on two dynamic domains.**

***Keywords*-markov logic networks; dynamic probabilistic inference; online inference**

I. Introduction

One of the goals of Artificial Intelligence research is to provide technical systems with the ability to take on tasks in a real-world setting. A popular and very apparent example is robot navigation and acting; but the problem stretches further to any sensory platform, like a PC equipped with a microphone or a video camera, or even a PC with only mouse and keyboard input.

Any such system contains an explicit or, most of the time, implicit model of its environment. For example, even the saved user preferences of some software product are a model of the product's user. Most current technical systems rely on deterministic models. But in order to gain flexibility and robustness, the incorporation of uncertainty is sometimes beneficial to accommodate for partial observability or non-deterministic behavior. Incorporating statistical knowledge to make "best guesses" about a user's preferences can enhance situations where the total time of interaction is not long enough to justify letting the user explicitly state any preferences. For example, a ticket vending machine at a train station can categorize a user into beginner or expert in order to decide between a very simple or a more efficient user interface on the fly.

One tool for creating such probabilistic models can be Markov Logic Networks (MLN) [1]. They combine first-order predicate logic with probabilistic semantics and allow a more abstract and convenient way of modeling probabilistic systems than working at a propositional level like it is necessary when crafting Bayesian networks directly. Probabilistic weights can be provided by experts, taken from literature, or they can be retrieved from data sets by parameter learning techniques.

In addition, to be able to model more complex systems it is often necessary to incorporate time for systems that change dynamically and information from a past state shall be carried over by means of the probabilistic model. Such problems can range from object tracking to the development of markets or social networks.

In two experimental setups, dynamic MLNs have already been used to model and recognize events from pre-processed video data of a parking lot surveillance scene [2] and from GPS data recorded during a game of capture the flag [3]. In both settings, the MLNs have been applied in an offline mode, where the inference is done over a defined time frame, usually the whole experiment, after the experiment is over. For real applications this solution is often insufficient and a method is needed that can run online and in real-time.

Recent publications by Nath and Domingos address the need for efficient online inference methods for MLNs [4], [5]. Their approach of Expanding Frontier Belief Propagation (EFBP) describes a message computation schedule for belief propagation. The method aims to reuse past computation results when making incremental changes to the model.

Although belief propagation is usually very fast, it can fail to converge to the correct solution or even enter into oscillation when applied to loopy graphs. For this reason, we propose an alternative approach of reusing past inference results, which is not limited to belief propagation, but can also be applied to MCMC methods and inference algorithms special to MLNs, like MCSAT. This is achieved by constructing a new MLN for each time step which is then augmented with additional information that is taken from the marginal probabilities obtained during the computation for the last time step. The disadvantage of not working on the level of message passing is a loss in flexibility. While EFBP can be tuned to weigh accuracy against speed by the use of a parameter, the approach presented in this paper is fixed to compute only approximate results.

The rest of the paper is structured as follows. First we describe MLNs and our approximation method. Then we give an overview of the relevant related work. Finally we provide an evaluation of the method using two dynamic domains. The paper closes with a conclusion and a perspective on future applications.

II. Markov Logic Networks

In the following paragraphs, we describe MLNs and their probabilistic semantics. After introducing dynamic Markov Logic Networks (DMLN), we define slice networks as an approximation to a DMLN.

A *Markov logic network* $L = \{(f_1, w_1), \dots, (f_n, w_n)\}$ for $n \in \mathbb{N}$ is a set of first-order formulas $f_1, \dots, f_n$ that are given weights $w_1, \dots, w_n \in \mathbb{R}$. Together with a finite set of constants C, they define a probability distribution over all interpretations (or possible worlds). An interpretation maps each grounding of each predicate to a truth value. Let $g(f)$ be the set of groundings of formula f obtained by replacing the free variables in f by all combinations of constants from C. Given an interpretation x, then $n_i(x) \overset{\text{def}}{=} |\{g \mid g \in g(f_i) \text{ and } x \models g\}|$ are the number of groundings of formula f_i that are true under x. Then the probability distribution P_L that is defined by the MLN L is given as

$$P_L(X = x) \overset{\text{def}}{=} \frac{1}{Z} \prod_i \exp\big(w_i n_i(x)\big) \tag{1}$$

where i ranges over all formulas in L and Z is the normalization constant.

Note that we can consider an interpretation to be an assignment to a multivariate probability distribution where the random variables are the truth values of the elements of the Herbrand base (the atoms formed by the grounding of the predicates). We can thus compute the marginal probability of a ground atom p being true as

$$P_L(p) \overset{\text{def}}{=} \frac{1}{Z} \sum_x P_L(X = x) I_x(p), \tag{2}$$

where the function I_x maps p to 1 if $x \models p$ and to 0, otherwise.

For practical reasons a sorted (or typed) logical language is used for MLNs. In order to model dynamic domains we assign a dedicated time sort T whose constants are elements from the natural numbers, i.e., $C_T \subseteq \mathbb{N}$. We demand that the time parameter appears as the first argument in every time-dependent predicate. A *dynamic Markov logic network* is a MLN for which such a dedicated time sort T exists. We call a DMLN *pure*, if all predicates are time-dependent.

A. Slice Networks

The main idea of the paper is to reuse old computations when doing inference over a DMLN progressing in time. When employing a belief propagation algorithm, this can be achieved by selectively updating only newer nodes while reusing the messages emitted by older nodes, like it is done by EFBP. Contrary, reusing past results when performing inference with MCMC methods is not as simple, because new evidence usually invalidates already obtained samples. Also, MCSAT does not lend itself very well to tweaking, because it requires that factors are given as weighted logical formulas. Thus one cannot simply add arbitrary factors to an existing network.

Table I: Listing of the MLNs, used for evaluation. The upper one, which is an adaption of the classic smokers example to a dynamic domain, is taken from Kersting et. al [6]. The lower one is inspired by the social force model for pedestrian movement [7].

(a) Dynamic Smokers Domain

```
//smoking causes cancer
1.0 smokes(t,x)⇒cancer(t,x)

//friends share smoking habits
1.0 friends(t,x,y)⇒(smokes(t,x)⇔smokes(t,y))

//friendship persists over time
3.0 friends(t,x,y)⇔friends(t+1,x,y)

//smoking persists over time
3.0 smokes(t,x)⇔smokes(t+1,x)
```

(b) Social Force Domain

```
//predicate is functional in Location
at(Time, Agent, Location!)

//only move one step at a time
2 at(t+1,a,x) ⇒ at(t,a,x-1) ∨ at(t,a,x)
   ∨ at(t,a,x+1)

//two agents do not occupy the same location
1.5 !(at(t,a1,x) ∧ at(t,a2,x))
```

We overcome these problems by using the marginal probabilities of ground atoms of a past time step to construct weighted formulas to be included in the network for the next time step. These formulas capture the summary over the removed messages from the older time slice. Obtaining simply a new MLN, we can then run inference with every algorithm that is suited to compute marginal probabilities for MLNs. We are now going to introduce the necessary vocabulary.

If L is a pure DMLN, then $L[t]$ is a DMLN, for which the time sort has only two constants $t-1$ and t, i.e., $C_T = \{t-1, t\}$ and all formulas that contain only atoms of a single time step are fixed to the time t. We call such a temporal fragment a *slice network*.

The Herbrand base of a slice for time step t thus contains only the ground atoms of time t and $t-1$. The intra-time formulas that relate between variables at time $t-1$ are removed. Figure 1 illustrates which components of an unrolled network are instantiated for a certain slice when the ground MLN is seen as a factor graph. Note that ground formulas are factor nodes and ground atoms are variable nodes.

As a textual example, we look at the smokers domain from Table I. To create the slice for time 3, we ground all

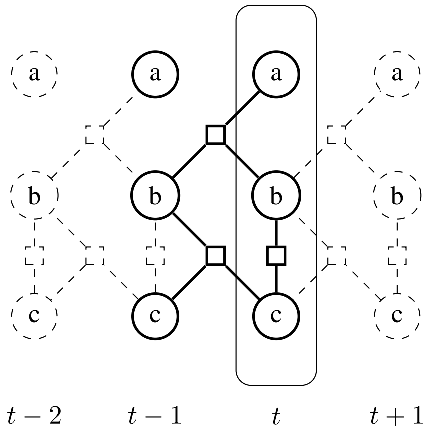


Figure 1: The figure shows a factor graph representation of a ground slice network for time t in bold, while the complete network over the whole time span is indicated dashed. The box surrounds the ground atoms associated with time t. The square nodes are the factor nodes, which are induced by ground formulas. The circles are variable nodes representing ground atoms. Notice the slice network at time t contains only the intra-time formula connecting variable b and c for time t, although both variables are also instantiated for time $t-1$.

inter-time formulas to the time frame between 2 and 3 and ground all intra-time formulas to the time 3. The resulting slice network is given as follows:

```
//intra-time formulas
1.0 smokes(3,x)⇒cancer(3,x)
1.0 friends(3,x,y)
      ⇒(smokes(3,x)⇔smokes(3,y))

//inter-time formulas
3.0 friends(2,x,y)⇔friends(3,x,y)
3.0 smokes(2,x)⇔smokes(3,x)
```

What is still left to describe is the process of transferring information from one slice to the next. This is done by adding formulas that capture the marginal distribution for each ground atom as it was calculated during the last slice; we call this process *augmentation*.

Given a MLN L, a set of ground atoms V, and a function $p : V \to [0,1]$, that maps atoms to their marginal probabilities, we define the *augmented MLN* by

$$L \uparrow (V,p) \overset{\text{def}}{=} L \cup \left\{ \left(f, \ln \frac{p(f)}{1-p(f)} \right) \mid f \in V \right\}. \quad (3)$$

The augmentation preserves only some information from a previous slice. To stay exact, it would be necessary to carry over the joint distribution over the random variables in the last time step. We approximate the joint distribution by the marginal distributions over the atoms inside the last time step, assuming marginal independence. This approximation is also done in the factored frontier algorithm for dynamic Bayesian networks [8].

Given a pure DMLN L, we define the sequence of *augmented slice networks* as the augmented MLNs L_i, with $0 \leq i$ in the following way:

$$L_0 \overset{\text{def}}{=} L[0] \quad (4)$$

$$L_i \overset{\text{def}}{=} L[i] \uparrow (V_{i-1}, p_{i-1}) \quad (5)$$

The second definition uses the set V_{i-1} to refer to the ground atoms of time $i-1$. Also, we have abbreviated the marginal probabilities described by the MLN L_{i-1} with p_{i-1}.

The reasoning why the augmentation is defined as described goes as follows. We assume that we are performing belief propagation on the ground factor graph, where each ground formula corresponds to a factor node and is adjacent to the variable nodes of the ground atoms that appear inside the formula. We want to create a new factor f_v for each variable node v of time $t-1$ in slice t that summarizes the incoming messages to v originating from the removed factors during the instantiation of slice $t-1$.

Since *all* factors that were present during slice $t-1$ are removed in slice t, we must summarize *all* incoming messages for v. Fortunately, the marginal distribution of v during slice $t-1$ is exactly the summary over those messages. And this can be obtained even if we do not have access to the messages in the first place, e.g., when running an MCMC algorithm. Thus by adding a factor that emits the marginal distribution of v during the last slice, i.e., $p_{t-1}(v)$, as its constant message to v, we capture the "frozen" messages from slice $t-1$ and can put them into the model for slice t. This construction is sound and exact as long as the factor graph contains no cycles spanning over two time steps. Because then, the messages coming from the older part will not change in the light of information coming from the newer network and can be safely frozen.

We have now defined a series of MLNs, we call slice MLNs, that each range over two time steps. The marginal probabilities that are defined by them are an approximation to the marginal probabilities defined by the DMLN that ranges over the complete time span. The definition indicates the intended way of inferring those probabilities. This is done by successively constructing the slice networks, inferring their marginal probabilities, constructing the next slice using the output of the last, and so forth. For computing the marginal probabilities of a slice, every inference algorithm that works on MLNs and computes marginal probabilities can be used.

III. Related Work

Nath and Domingos have made two contributions to online inference for MLNs in 2010. In "Efficient Lifting for Online Probabilistic Inference" [4] they describe a way to update lifted networks in order to reduce the cost of the lifting procedure when dealing with incremental changes. As a dynamic application, they apply their algorithm to the computer vision task of video background segmentation over short snippets (about ten frames) of motion data.

In "Efficient Belief Propagation for Utility Maximization and Repeated Inference" [5], they describe an algorithm called *Expanding Frontier Belief Propagation*, whose purpose is to reuse as much information from an earlier run of belief propagation as possible. This approach is applicable to changes of evidence and network structure, and can thus be useful for online inference. We describe this approach in more detail, as it is related to the idea of slice networks.

EFBP begins by performing normal belief propagation on a Markov network. The final messages after convergence are stored. Then the network gets changed, which in our case means the addition of a new temporal step and the according evidence. All nodes that are directly affected by the change are considered *active*. Then belief propagation is performed with only the active nodes recomputing their messages. The new messages are constantly compared with the stored messages from the computation before the network update. If a non-active node receives a message that differs more than a predefined constant from the old message, then it gets activated and participates in belief propagation for the updated network.

The nodes that are activated directly after a network update in EFBP are the nodes that are part of the current slice in our approach. The difference to the slice network approach is that we cannot activate additional nodes and thus our method is a special case of EFBP where the message threshold is infinity and no nodes are activated. But in contrast to EFBP, slice networks allow using different inference algorithms than EFBP, which requires belief propagation.

IV. Evaluation

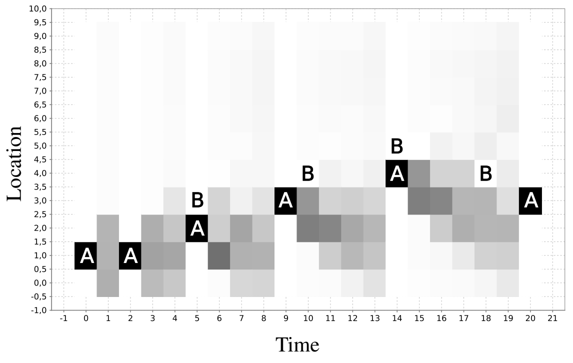
We have implemented the slice method for MLNs and a set of inference algorithms based on a factor graph representation using the Scala programming language. The Alchemy system[1] and pyMLNs[2] have been used as reference implementations. The system, including the experiments, is available for download at our website[3]. The experiments were run on an Intel Core2 CPU with 2.8 GHz and 4 GB of RAM.

We have evaluated the approach using two domains. The listing for both is given in Table I. The first domain is a simple model of pedestrian movement. Agents may wander in a one-dimensional space and they are repelled by other agents. We have simulated this problem for 20 time steps and two agents. Evidence was added as indicated in the figure. We have used loopy belief propagation with a

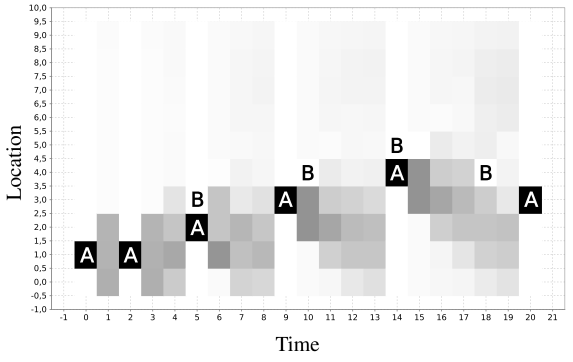
[1]http://alchemy.cs.washington.edu

[2]http://www9-old.in.tum.de/people/jain/mlns

[3]http://www.uni-ulm.de/in/ki/forschung/mln/scb



(a) Slice Filtering



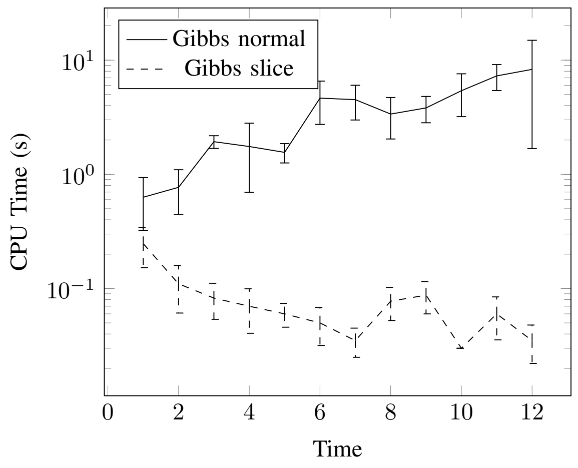
(b) Normal Filtering

Figure 2: Two plots of the probability distribution for the location of only agent `A` against time. We have observed the positions of both agents `A` and `B` at several time steps, annotated by letters. For example at time 5, agent `A` is at location 2 and agent `B` is at location 3. Darker cells represent higher probability of agent `A` occupying the location. We have plotted the slice approximation (a) and inferring over the complete unrolled network up to the current time step (b). Notice, the approximation does not differ much from the exact solution.

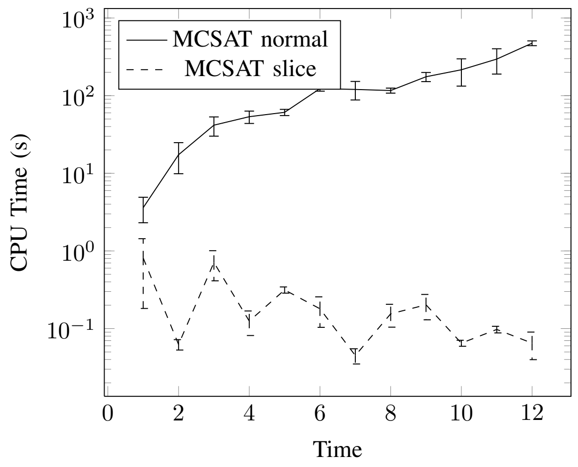
flooding schedule as the inference algorithm. We ran four parallel computations until the marginals have converged below a variance of 10^{-5}. The probability distributions are graphically visualized in Figure 2.

The difference between inference using slice networks (a) and normal inference (b) is very small for the given domain and problem. Whether the error produced by the approximation is feasible depends on the inference task and the requirements of the application.

For the second experiment, we have measured the running times of the slice version of Gibbs sampling and MCSAT against inference on the unrolled network. This was done using the dynamic smokers domain by Kersting et al. [6]. We have chosen a simpler setup than theirs with only four persons to reduce the inference times for the experiment. We have simulated the problem for 12 time steps. Again we ran



(a) Gibbs Sampling



(b) MCSAT

Figure 3: The plots list the running time of Gibbs sampling and MCSAT both on the unrolled network and incrementally using the slicing approach for 12 time steps.

four parallel computations until the marginal probabilities converged to a maximum variance below 0.003. To help the convergence of the Gibbs sampler, we have reduced the weights on some formulas in contrast to the original model. We have also added random evidence, observing a random truth value for each ground atom with a probability of 0.5, thus observing about one half of the random variables. We compare the computation time of unrolled inference for each increment of time against the incremental inference times for the slice network approach.

The CPU time for the unrolled computations increases with the time span over which to infer, while the computation times for the slicing approach roughly stay constant. As expected the effort for the slice approach is very low compared to redoing inference on the unrolled network on each time step.

V. Conclusion and Future Work

We have presented an approach for approximate computation of marginal probabilities for DMLNs that is suitable for online inference. The approximation ensures that at every time step only a limited amount of computation must be performed. The concept of slice MLNs is both a special case of the EFBP algorithm and a generalization. The EFBP algorithm is more flexible than using slice networks because it allows to trade in speed for improved accuracy. On the other hand EFBP is limited to belief propagation inference while slice networks allow to employ any inference algorithm for marginal probabilities of MLNs.

In the future, we intend to apply probabilistic models based on MLNs to the integration of sensory data with symbolic knowledge. Among the planned applications is the selection of output modalities for multi-modal user interfaces based on the environmental situation, like lighting and noise level. Properties of the user and personal preferences shall also be taken into account. For this application, a fast inference algorithm is particularly important in order to reduce the experienced lag of the resulting system.

Acknowledgment

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

[1] M. Richardson and P. Domingos, “Markov logic networks,” *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[2] S. Tran and L. Davis, “Event modeling and recognition using markov logic networks,” *Computer Vision–ECCV 2008*, pp. 610–623, 2008.

[3] A. Sadilek and H. Kautz, “Recognizing multi-agent activities from GPS data,” in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010, pp. 1134–1139.

[4] A. Nath and P. Domingos, “Efficient lifting for online probabilistic inference,” in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010, pp. 1194–1198.

[5] A. Nath and Domingos, “Efficient Belief Propagation for Utility Maximization and Repeated Inference,” in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010, pp. 1187–1192.

[6] K. Kersting, B. Ahmadi, and S. Natarajan, “Counting belief propagation,” in *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2009, pp. 277–284.

[7] D. Helbing and P. Molnár, “Social force model for pedestrian dynamics,” *Phys. Rev. E*, vol. 51, no. 5, pp. 4282–4286, 1995.

[8] K. Murphy and Y. Weiss, “The factored frontier algorithm for approximate inference in DBNs,” in *Proceedings of the 17th Conference on Uncertainty in AI*, 2001, pp. 378–385.