PETS: Predicting Efficiently using Temporal Symmetries in Temporal PGMs

Florian Andreas Marwitz[0000−0002−9683−5250], Ralf Möller[0000−0002−1174−3323], and Marcel Gehrke[0000−0001−9056−7673]

Institute of Information Systems, University of Lübeck, Germany
{marwitz,moeller,gehrke}@ifis.uni-luebeck.de

Abstract. Time in Bayesian Networks is concrete: In medical applications, a timestep can correspond to one second. To proceed in time, temporal inference algorithms answer conditional queries. But the interface algorithm simulates iteratively into the future making predictions costly and intractable for applications. We present an exact, GPU-optimizable approach exploiting symmetries over time during answering prediction queries by constructing a matrix for the underlying temporal process. Additionally, we construct a vector capturing the probability distribution at the current timestep. Then, we can time-warp into the future by matrix exponentiation. We show an order of magnitude speedup over the interface algorithm. The work-heavy preprocessing step can be done offline, and the runtime of prediction queries is significantly reduced. Now, we can handle application problems that could not be handled before.

Keywords: Dynamic Bayesian Network · Prediction · Probabilistic graphical models.

1 Introduction

Probabilistic Graphical Models (PGMs) encode probability distributions, which can be used to perform inference [8]. They can be extended to dynamic PGMs to take temporal behavior into account. In temporal settings, one task is predicting the probability of a random variable. In general, queries are costly and so are prediction queries. In, e.g., medical applications, a timestep can correspond to one second and for the selection of the correct treatment we need to predict into the future. To exploit factorizations, temporal inference algorithms compute a set of random variables, called interface, that render two timesteps independent of each other. Hence, to proceed to the next timestep, these algorithms calculate a conditional query, which is costly. Thus, to answer prediction queries, a temporal inference algorithm has to possibly eliminate many random variables in addition to answering the conditional query for each timestep during predictions. However, given a stationary process, the temporal behavior is actually independent of the current timestep and, for prediction queries, models are not manipulated with new events. Therefore, we propose to compute the temporal behavior of the model in an offline step and store it in a matrix. We can then

query the distribution of our interface variables for our current timestep and jump to the timestep we are interested in to answer the query. In medical applications, the requested timestep can change, so we cannot modify our model, but rather need to fast forward to the timestep. In doing so, we replace expensive inference computations with cheap matrix-vector multiplications, while still getting exact results for answering the query. Our approach is applicable to all temporal inference algorithms using temporal conditional independences. In this paper, for illustrative purposes, we restrict ourselves to Dynamic Bayesian Networks (DBNs) and the interface algorithm (IA) [10].

Judea Pearl introduces Bayesian Networks (BNs) [11]. BNs store a probability distribution in a factored way using conditional independences. BNs are extended by Paul Dagum to DBNs [3]. DBNs generalize Hidden Markov Models and Kalman Filters [13]. Our approach works with discrete random variables rather than with normal distributions a Kalman Filter requires [6]. Variable elimination (VE) can be used to infer probabilities in BNs or in unrolled DBNs [15]. Kevin Murphy develops the IA for more efficient inference in DBNs [10] using the junction tree by Lauritzen and Spiegelhalter [9], which is ideal for answering multiple queries [9]. The concept of lifting involves identifying similar random variables and operating with a single representation [7, 14]. Our approach utilizes time-induced symmetries in the behavior of DBNs. Besides exact inference, we can do approximate inference, e.g., by representing the belief state as a product of marginals [2]. Kevin Murphy gives an overview over approximate inference [10]. These approaches all require to perform costly inference for predictions. We propose an algorithm to skip almost all inference computations during prediction.

The huge cost for prediction queries in inference algorithms is because the probability distributions are simulated iteratively. We present a new algorithm *PETS* for logarithmic time-warping to a requested timestep and illustrate it using DBNs as focus. When advancing in time during prediction, we always perform the same calculations. PETS exploits these symmetries and constructs a matrix-vector representation for them. The matrix representation enables GPU-optimization of our algorithm and fast exponential squaring leads to logarithmic time-warping. The construction of the matrix can be done offline, so the cost per timestep is reduced significantly from costly inference to a cheap matrix-vector multiplication. The horizon t of a query is the number of timesteps between the current timestep and the queried timestep. The runtime for large horizons t of PETS is exponential only in the number of interface nodes, while the IA is in the worst-case exponential in the number of nodes in the current timestep. PETS can be used as a submodule in existing inference algorithms to speedup prediction queries: If we are at timestep t and want to predict the probability distribution in timestep $t + h$, the inference algorithm could utilize our matrix-vector formulation. Notably, we can run our algorithm on edge devices, when the offline step is done beforehand, or on GPUs for more complex models or queries.

We start in Section 2 with the required preliminaries. After that, we present PETS in Section 3. In Section 4, we evaluate PETS. We end with a conclusion.

2 Preliminaries

In this section, we define BNs and DBNs. BNs model probability distributions exploiting conditional independences. Additionally, DBNs take temporal behavior into account. We build on the definitions of Pearl and Murphy [12, 10].

A BN is a directed acyclic graph. For a probability distribution $P(X_1, \ldots, X_n)$ over random variables $X_1, \ldots, X_n$, the graph consists of n nodes, one for each random variable. The edges in the network model influences. Each node X_i has a conditional probability distribution (CPD) $P(X_i \mid Pa(X_i))$ assigned, where $Pa(X_i)$ stands for the parents of X_i in the network. The semantics of a BN is

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa(X_i)). \tag{1}$$

While a BN can represent any probability distribution, it lacks the ability to take temporal effects on the random variables into account. Figure 1 shows a DBN. The umbrella network consists of two random variables R, indicating that it is raining, and U, indicating that we take the umbrella. Both are repeated for every day. A BN would need to include all random variables for all timesteps. A DBN consists of two BNs $(B_0, B_\rightarrow)$ splitting the definition of a temporal process into two parts: B_0 defines a prior over the variables $X_1 = \{X_1^i \mid i = 0, \ldots, n\}$ and $B_\rightarrow$ defines the temporal behavior. That is, the probability distribution of all random variables in timestep t given their parents. The parents of a node X_t^i, $t > 0$, can be in the same timestep t or in the previous timestep $t-1$. Figure 1 gives an example for defining B_0 and $B_\rightarrow$ for the umbrella network by Russell and Norvig [13]: The initial distribution defines $P(R_0)$ and $P(U_0 \mid R_0)$ and the transition distribution defines $P(R_t \mid R_{t-1})$ and $P(U_t \mid R_t)$. The semantics can be defined by unrolling the network, i.e., instantiating the model for T timesteps:

$$P(X_{0:T}) = \prod_{t=0}^{T} \prod_{i=1}^{n} P(X_t^i \mid Pa(X_t^i)). \tag{2}$$

The prediction task is to compute $P(Y_{t+\pi} \mid E_t)$ for $t, \pi \in \mathbb{N}_0$ with $\pi > 0$ for some $Y, E \subseteq X$ with $Y \cap E = \emptyset$. Figure 1 gives an example for a DBN modeling the probability of rainy days and whether we take an umbrella [13]. When it is raining (R_i = true), we take the umbrella (U_i = true) with a probability of 0.9. Otherwise, we take the umbrella with a probability of 0.1. If it is raining today, it rains tomorrow with a probability of 0.7. Otherwise, it rains with a probability of 0.3. On the first day, it rains with a probability of 0.7. Now, we could be interested in the probability of $P(U_7 \mid R_0)$, the probability that we take the umbrella next week. In general, the probability distribution over variables X_t can be computed given X_{t-1}. The key observation is, that there is a set $I_{t-1} \subseteq X_{t-1}$ sufficient for computing $P(X_t \mid I_{t-1}) = P(X_t \mid X_{t-1})$. This set is called *interface* and consists of all variables in B_{t-1} with successors in B_t [10]. The interface in Figure 1 is $\{R_{t-1}\}$. In the next section, we show how we can exploit temporal symmetries to perform efficient predictions by matrix-vector multiplication.

(a) B_0 for the umbrella network (b) $B_{\rightarrow}$ for the umbrella network

Fig. 1: Graphical representation of B_0 and B_t for the umbrella network [13].

3 PETS Algorithm: Predicting Efficiently using Temporal Symmetries

In this section, we develop PETS for fast prediction in temporal models assuming a stationary process and first order Markov assumption. As stated earlier, we focus on DBNs. When advancing in time, we multiply always the same temporal behavior, encoded by $P(I_t \mid I_{t-1})$, on the model. The transition matrix A captures these symmetries and models the change $P(I_t \mid I_{t-1})$. The state vector s_t contains the current distribution $P(I_t)$ over the interface. Together, we have a matrix-vector representation of the interface and it's change over time. Now, we can proceed in time during prediction using cheap matrix-vector multiplication: Intuitively, we get $P(I_t) = \prod_{i=1}^{t} P(I_i \mid I_{i-1}) \cdot P(I_0) = A^t \cdot s_0$. The outline of PETS is as follows: First, PETS builds the transition matrix and the current state vector. Then, PETS can calculate the distribution over the interface variables for each timestep by matrix-vector multiplication. In the end, PETS materializes s_{t-1} and answers queries on B_t using VE. Materializing is the process of updating the CPDs such that the interface variables occur with the probability given by s_{t-1}. Thus, PETS performs the following five steps, which we describe in the same order: (i) identify interface, (ii) identify basis probabilities, (iii) build transition matrix, (iv) build current state vector, and (v) query answering. Algorithm 1 shows pseudocode for PETS.

Identify Interface In Section 2, we define the interface. Thus, we can loop over all nodes in B_0 and check if they have a successor in B_1.

Identify Basis Probabilities The interface renders two adjacent timesteps independent of each other. Moreover, given an assignment for the interface variables, the probability distribution for all other variables is deterministic. In fact, we can rewrite $P(X_t^i)$ in terms of interface variables. Because we may need joint probabilities, we cannot formulate this as a linear combination. We call the set of distributions over (possibly joint) random variables required to compute $P(I_{t+1})$ through $P(I_{t+1} \mid I_t)$ from $P(I_t)$ *basis probabilities*. Then, we can write $P(i_{t-1})$, $i_{t-1} \in I_{t-1}$, as a linear combination of basis probabilities. The coefficients of

Algorithm 1 PETS. *index* maps an assignment to an index, $relevant(x)$ maps to the relevant basis probabilities to compute $P(x)$

Require: DBN $(B_0, B_\rightarrow)$, query $P(Y_{t+\pi} \mid E_{0:t})$ with $\pi > 0$
Ensure: $p = P(Y_{t+\pi} \mid E_{0:t})$
  identify interface and basis probabilities
  for all assignments b for all basis probabilities B **do**
    $R \leftarrow relevant(B)$
    for all assignments r of R **do**
      $A[index(b), index(e)] \leftarrow P(B = b \mid R = r)$ ▷ fill transition matrix
    end for
  end for
  for all assignments b for all basis probabilities B **do**
    $s_t[index(b)] \leftarrow P(B_t = b \mid E_{0:t})$ ▷ fill state vector
  end for
  $s_{t+\pi-1} \leftarrow A^{\pi-1} \cdot s_t$ ▷ advance in time
  unroll B_0, B_1
  materialize $s_{t+\pi-1}$ into B_0 ▷ ensure CPDs to match s_t
  $p \leftarrow P(Y_1)$ on B_0, B_1 ▷ answer query

the linear combination constitute the transition matrix modeling the temporal behavior. We could also use the joint probability distribution over the interface as basis probabilities. Exploiting conditional independences, this is not always needed and in this subsection, we describe a method for finding only the necessary joint probabilities.

Assume we want to compute $P(U_t)$ in Figure 1. Unrolling only the current timestep, we get $P(U_t) = P(U_t \mid R_t) \cdot P(R_t \mid R_{t-1}) \cdot P(R_{t-1})$ by Equation 2. The conditional probabilities $P(U_1 \mid R_1)$ and $P(R_1 \mid R_0)$ are part of the model and thus known. We call $P(R_{t-1})$ a *basis probability*, because we can write $P(U_t)$ as a linear combination of $P(R_{t-1})$. For some variables, we may have to calculate a joint probability. We cannot write this as a linear combination of single variables, so we include the joint variables as an additional basis probability. This enables us to further use the matrix formulation.

For finding the required joint probabilities to include as basis probabilities, as we want to keep the set as small as possible, PETS runs a depth-first search for each node $v \in I$ following edges in reversed direction and terminating a branch once it reaches a node in the interface. This cannot be substituted by a simple lookup of the predecessors, because an interface variable may depend on a non-interface variable, which itself depends on an interface variable. The set of needed basis probabilities $relevant(X^i)$ for a variable X^i are all nodes in the interface visited during the depth-first search for that node and added to the set of basis probabilities. Because some variable $v \in relevant(X^i)$ may need the probability of other variables in the interface, we augment the basis probabilities iteratively: For each set $relevant(X^i)$, PETS adds the set $\bigcup_{v \in relevant(X^i)} relevant(v)$ to the set of basis probabilities and sets $relevant(relevant(X^i))$ to the newly added basis probability. Then, PETS unions all basis probabilities sharing some variable

and sets *relevant* accordingly to account for joint treatment of same variables. By the end of this step, we have identified all basis probabilities M. For storing them in a matrix and vector, we fix any order on them. In the umbrella network, the basis probabilities consist of only R_{t-1} and thus match the interface.

Build Transition Matrix We can calculate all probability distributions in B_t with the help of the basis probabilities. We can encode the calculation of the basis probabilities in a matrix, because we argue earlier that we can write the probabilities as linear combinations of basis probabilities. Thus, the transition matrix A models $P(M_t \mid M_{t-1})$. The nodes $m_{t-1} \in M_{t-1}$ and $m_t \in M_t$ do not have to be neighbors. Therefore, we need to calculate $P(m_t = i \mid M_{t-1})$ for all $m \in M$ and store that in the corresponding row in A for $m_t = i$. In fact, PETS calculates this probability by running VE with all evidences $M_{t-1} = j$ over all domains to obtain the linear combination of $m_t = i$ on basis M_{t-1}. In the umbrella network, the transition matrix is $A = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$. Note that, in general, the transition matrix is not simply composed of transition probabilities.

Build Current State Vector The current state vector s_t captures $P(M_t)$. Therefore, PETS runs VE to calculate $P(m_t)$ for all $m_t \in M_t$. With the transition matrix A, PETS can then calculate the state at the requested timestep $t+\pi$ by $s_{t+\pi} = A^{\pi} \cdot s_t$. In the umbrella network, the initial state vector is $s_0 = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$.

Query Answering Assume we want to know $P(Y_{t+\pi} \mid E_t)$ for some $Y_{t+\pi} \subseteq X_{t+\pi}, E_t \subseteq X_t$. In short, PETS answers queries in three steps: First, PETS calculates the state vector $s_{t+\pi-1}$. Then, PETS materializes $s_{t+\pi-1}$ forcing the distribution over the interface variables to match the state vector. Finally, PETS runs VE to answer the query. When an observation is added at some timestep, we can update the probability distributions over the interface accordingly and update the state vector. Afterward, our algorithm can be used further.

For calculating $s_{t+\pi-1}$, PETS computes the current state vector s_t regarding E_t and fast forwards to $s_{t+\pi-1} = A^{\pi-1} \cdot s_t$. Materializing $s_{t+\pi-1}$ means forcing the CPDs of the interface variables to match the state vector. For basis probabilities containing only one random variable, we can just update the CPD of that variable and remove all ingoing edges. For joint basis probabilities, we must ensure that these random variables are treated jointly and not independently: We add a new node for new variable J_j for all joint basis probabilities $j \in M$. We connect J_j to all random variables included in the joint basis probability j. We update the CPD of variable v included in basis probability j to pass through its value assigned by J_j with probability one. Analogously, speaking in terms of IA, we construct the ingoing message for the junction tree for timestep t.

Integrating Query Variables Often we are interested in the probability distribution of the same query variable $y_{t+\pi}$ for many t. In this case, PETS calls

VE for every t. However, we can integrate y_t into the transition matrix and state vector to compute the probability distribution of y_t on the fly skipping all VE calls except for initialization. The basic idea is to treat y_t as a basis probability and integrate y_t into the transition matrix and the state vector. Consequently, we have to redo all mentioned steps regarding basis probabilities for the newly added one. In the end, we do not need VE to calculate $P(y_{t+\pi})$ as opposed to general queries of the form $P(Y_{t+\pi} \mid E_t)$ and the inference step collapses into matrix-vector multiplication. Suppose we want to know $P(U_t)$ for all $t \in [1, T]$ in the umbrella network. We then have

$$A = \begin{pmatrix} 0.7 & 0.3 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 \\ 0.59 & 0.31 & 0 & 0 \\ 0.41 & 0.69 & 0 & 0 \end{pmatrix}, s_0 = \begin{pmatrix} 0.3 \\ 0.7 \\ 0.31 \\ 0.69 \end{pmatrix}, \tag{3}$$

with the first two components in referring to R and the last two to U. The last two rows in A represent $P(U_t \mid R_{t_1}) = P(U_t \mid R_t) \cdot P(R_t \mid R_{t-1})$. The last two entries in s_0 are the probability distribution $P(U_0) = P(U_0 \mid R_0) \cdot P(R_0)$. Please note that the umbrella network is a very small DBN, leading to a simple transition matrix and current state vector.

Correctness and Runtime Unrolling a DBN yields a BN, and Equations 1 and 2 coincide. For calculating $P(X_t^i)$, we need to calculate the probability distribution of the parents of X^i, going back to B_0. By the first-order Markov assumption, it is sufficient to extend Equation 1 only up to the interface to the previous timestep. Then, we have a linear combination in the basis probabilities, which include the interface variables and their required joints. By construction, our transition matrix stores the coefficients of the linear combination and the state vector the probability distribution over the basis probabilities. Therefore, one matrix-vector multiplication advances the probability distribution over the basis probabilities exactly one timestep.

The runtime of PETS is at most $\mathcal{O}(t \cdot q^3 \cdot i^6 \cdot d^{3k} + q^2 \cdot i^4 \cdot d^{2k} \cdot n \cdot 2^n)$ for horizon t, number of query variables q integrated into state vector, interface size i, maximum domain size d, maximum number of reachable nodes in interface k and n nodes in B_0. We can speed the first summand up by replacing t by $\log t$ when using fast exponential squaring instead of iterative matrix-vector multiplication. We can split the runtime in an offline preprocessing and online prediction part. The offline runtime is $\mathcal{O}(q^2 \cdot i^4 \cdot d^{2k} \cdot n \cdot 2^n)$, mainly because of the matrix construction. The online runtime is $\mathcal{O}(t \cdot q^3 \cdot i^6 \cdot d^{3k})$, when the current state vector is given. The construction of the current state vector is in $\mathcal{O}(q \cdot i^2 \cdot n \cdot 2^n + q \cdot i^2 \cdot d^k)$ and the IA has to compute it anyhow.

Summing Up In this section, we develop PETS, a new prediction algorithm capable of answering $P(Y_{t+\pi} \mid E_t)$ for some $Y_{t+\pi} \in X_{t+\pi}, E_t \subseteq X_t$. First, PETS identifies the basis probabilities required for computing the probability distribution over the interface variables. Then, PETS constructs a transition

matrix, which models the temporal behavior of the basis probabilities, and a state vector, containing the probability distribution for the basis probabilities in the current timestep. Finally, PETS uses matrix-vector multiplication to advance in time. The horizon-dependent runtime is $\mathcal{O}(t \cdot q^3 \cdot i^6 \cdot d^{3k})$. In particular, the cost per timestep is only exponential in the number of interface variables.

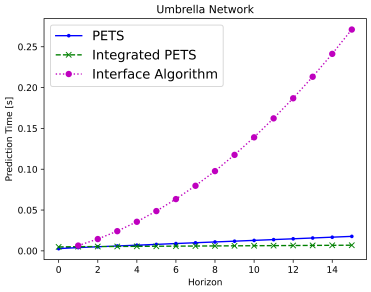
4 Evaluation

The main motivation behind PETS is to reduce costly VE calls to advance in time and replace them with cheap matrix-vector multiplications. In this section, we evaluate the runtime of PETS compared to IA. We use *pgmpy* to implement PETS and use its implementation of the IA [1]. We evaluate two variants of PETS: The first is PETS without integrating the query variables into the transition matrix and state vector, and the second is with integrating the query variables. We call the second variant *integrated PETS*. When measuring the runtime of (integrated) PETS, we include the construction of the transition matrix and state vector. In particular, this means that PETS is never slower when only the online runtime is measured. The evaluation is divided into three parts: First, we investigate the effect of saving VE calls. Second, the effects of a growing interface, and third, a theoretical view of when PETS outperforms IA.

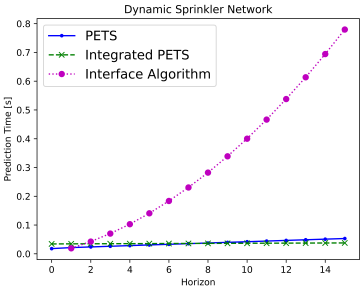
Faster Runtime We use two DBNs to evaluate PETS: the umbrella network as given in Figure 1 and a dynamic sprinkler network [4]. The task for all algorithms is to answer $\{P(Y_\pi)\}_{\pi=1}^{15}$ for all random variables Y_π in B_π. We plot the runtime in seconds against growing horizon $\pi \in [1, 15]$. The runtime of the offline step is plotted for $\pi = 0$. Figure 2 shows the results. The runtime of the IAs grows quadratic for both DBNs, while the runtimes of the PETS variants are linear. For the umbrella network and a horizon of 15, PETS outperforms the IA by a factor larger than 15. Integrated PETS outperforms the IA by a factor of more than 39. The offline step accounts for 15 % of the runtime in timestep 15 for PETS, and almost 70 % for integrated PETS. For the dynamic sprinkler network and a horizon of 15, PETS outperforms the IA by a factor larger than 14. Integrated PETS outperforms the IA by a factor of more than 20.

Growing Interface Size To test the effect of increasing interface size on the runtime of PETS, we construct a sink network consisting of n interface variables connected to a sink, e.g., the umbrella network with n interface variables all pointing to the sink umbrella.

Figure 3a shows the runtime for the three algorithms on the sink network with interface sizes starting at two and going up to nine. PETS is faster than IA because PETS does not store the full joint probability distribution over the interface variables by default. PETS performs a depth-first search to find only the necessary basis probabilities and stores a small transition matrix exploiting conditional independence. However, integrated PETS does, for this network,



(a) Test results for the umbrella network as given in Figure 1 [13].

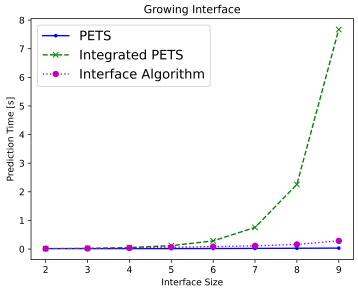


(b) Test results for a dynamic variation of the sprinkler network [4].

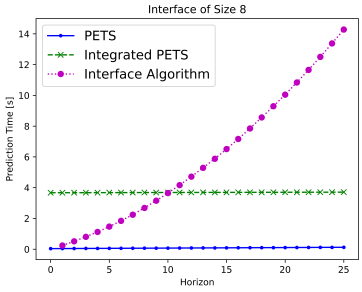
Fig. 2: Prediction times for two DBNs.

store the full joint to answer queries about non interface variables, and therefore the runtime is exponential in the interface size. Figure 3b shows the runtime of the three algorithms for the sink network with an interface size of eight, querying $\{P(Y_\pi)\}_{\pi=1}^{25}$. In this figure, we can see the labor-intensive preprocessing when integrating query variables: Integrated PETS is about 30 times slower than PETS. In spite of this, the IA is slower than both variants of PETS from horizon 11 on. This shows that even some heavy preprocessing pays off. The difference in runtime between PETS and integrated PETS is because integrated PETS makes many VE calls to integrate query variables into the matrix-vector representation. As Figure 3b shows, this does not pay off when querying only one horizon at a time, but it can be beneficial when querying in many timesteps. Figure 3b shows that integrating query variables does not pay off when querying only once. However, it can be beneficial when querying in many timesteps. PETS requires one VE call per prediction query of non-interface variables, so integrated PETS is faster when the number of prediction queries is greater than the number of VE calls required to integrate the query variables.

Theoretical Evaluation PETS includes preprocessing to construct the matrix-vector representation. In this subsection, we investigate at what point this preprocessing pays off compared to the IA. In the worst case, the matrix is constructed over a joint basis probability for the entire interface. Then we have d^k possible evidences leading to $\mathcal{O}(i^4 \cdot d^{2k})$ VE calls. For this estimation, we run VE once for each entry in the matrix. After that, we only have one more VE call for each prediction query. The IA calls VE once per timestep. Let n be the number of prediction queries and h_i the horizon for each prediction query. Then PETS outperforms IA in terms of VE calls once $\sum h_i > i^4 \cdot d^{2k} + n$. In the umbrella network, we have a variable with boolean cardinality in the interface. Thus, PETS performs 4 VE calls to construct its matrix. With only one prediction query, PETS is faster once $h > 5$ or overall two prediction queries with



(a) Prediction time for growing interface size in the sink network.



(b) Prediction time for the sink network with an interface size of eight.

Fig. 3: Prediction times for increasing interface sizes.

$h = 3$. In general, one is often interested in prediction from every time step into the future with a given horizon, so the initial offline costs pay off fast.

5 Conclusion

Temporal inference algorithms proceed iteratively in time for prediction queries. However, we multiply the same temporal behavior for each timestep to the model. We propose PETS, a new algorithm that exploits these temporal symmetries to time-warp to the requested timestep for efficient prediction. PETS stores the transition probabilities $P(I_t \mid I_{t-1})$ of the interface in a transition matrix. Next, a vector is constructed to capture the probabilities for the current state. After that, we can proceed in time by simple matrix-vector multiplication, as opposed to expensive inference. Moreover, the matrix-vector multiplication can be optimized on GPUs. The offline runtime of PETS is exponential in the number of random variables in the network, while the online runtime per timestep is only exponential in the size of the interface. Whereas, the runtime per timestep of the IA is exponential in the number of nodes in the network. Having a transition matrix A to go forward in time naturally raises the question whether we can go back in time with a backward transition matrix over the interface to answer hindsight queries. Moreover, we can investigate including lifting in the matrix construction leading to possible further speedups. Additionally, we can try to integrate the ideas presented in this paper to speed up planning in PGMs [5].

Acknowledgements The research for this paper was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2176 'Understanding Written Artefacts: Material, Interaction and Transmission in Manuscript Cultures', project no. 390893796. The research was conducted within the scope of the Centre for the Study of Manuscript Cultures (CSMC) at Universität Hamburg.

References

1. Ankan, A., Panda, A.: pgmpy: Probabilistic graphical models using python. In: Proceedings of the 14th Python in Science Conference (SCIPY 2015). Citeseer (2015)
2. Boyen, X., Koller, D.: Tractable inference for complex stochastic processes. In: Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. pp. 33–42 (1998)
3. Dagum, P., Galper, A., Horvitz, E.: Dynamic network models for forecasting. In: Uncertainty in artificial intelligence. pp. 41–48. Elsevier (1992)
4. Darwiche, A.: Modeling and reasoning with Bayesian networks. Cambridge university press (2009)
5. Gehrke, M., Braun, T., Möller, R.: Lifted temporal maximum expected utility. In: Advances in Artificial Intelligence: 32nd Canadian Conference on Artificial Intelligence, Canadian AI 2019, Kingston, ON, Canada, May 28–31, 2019, Proceedings 32. pp. 380–386. Springer (2019)
6. Hartwig, M.: New Methods for Efficient Query Answering in Gaussian Probabilistic Graphical Models. Ph.D. thesis, University of Lübeck (September 2022), phD thesis
7. Kersting, K., Ahmadi, B., Natarajan, S.: Counting belief propagation. arXiv preprint arXiv:1205.2637 (2012)
8. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
9. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society: Series B (Methodological) **50**(2), 157–194 (1988)
10. Murphy, K.P.: Dynamic bayesian networks: representation, inference and learning. University of California, Berkeley (2002)
11. Pearl, J.: Probabilistic reasoning using graphs. In: Uncertainty in Knowledge-Based Systems: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems Paris, France, June 30–July 4, 1986 Selected and Extended Contributions. pp. 200–202. Springer (1987)
12. Pearl, J.: Bayesian networks (2011)
13. Russell, S.J., Norvig, P.: Artificial intelligence a modern approach. Pearson Education, Inc. (2010)
14. Singla, P., Domingos, P.M.: Lifted first-order belief propagation. In: AAAI. vol. 8, pp. 1094–1099 (2008)
15. Zhang, N.L., Poole, D.: A simple approach to bayesian network computations. In: Proc. of the Tenth Canadian Conference on Artificial Intelligence (1994)