Lifted Temporal Maximum Expected Utility*

Marcel Gehrke[0000−0001−9056−7673], Tanya Braun[0000−0003−0282−4284], and Ralf Möller

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

Abstract. The lifted dynamic junction tree algorithm (LDJT) efficiently answers exact filtering and prediction queries for temporal probabilistic relational models by building and then reusing a first-order cluster representation of a knowledge base for multiple queries and time steps. To also support sequential online decision making, we extend the underling model of LDJT with action and utility nodes, resulting in parameterised probabilistic dynamic decision models, and introduce meuLDJT to efficiently solve the exact lifted temporal maximum expected utility problem, while also answering marginal queries efficiently.

1 Introduction

Areas such as healthcare and logistics deal with probabilistic data including relational and temporal aspects and need efficient exact inference algorithms. These areas involve many objects in relation to each other with changes over time and uncertainties about object existence, attribute value assignments, or relations between objects. More specifically, healthcare systems involve electronic health records (relational) for many patients (objects), streams of measurements over time (temporal), and uncertainties due to, e.g., missing information. In this paper, we study the problem of supporting exact decision making and query answering in temporal probabilistic relational models.

We [3] present parameterised probabilistic dynamic models (PDMs) to represent temporal probabilistic relational behaviour and propose the lifted dynamic junction tree algorithm (LDJT) to efficiently answer multiple *filtering* and *prediction* queries. LDJT combines the advantages of the interface algorithm [5] and the lifted junction tree algorithm (LJT) [2]. Specifically, this paper contributes (i) parameterised probabilistic dynamic decision models (PDDecMs) by adding action and utility nodes to PDMs and (ii) meuLDJT to efficiently solve the temporal maximum expected utility (MEU) problem using PDDecMs, while also answering marginal queries efficiently.

Nath and Domingos [6] perform first steps to formally define action and utility nodes for static probabilistic relational models. Further, Apsel and Brafman [1] propose an exact lifted static solution to the MEU problem based on [6]. We [4] propose a solution based on LJT to combine decision support and efficient marginal query answering. However, in this paper, we propose to include sequential decision making. Research on sequential decision making relates to first-order (partially observable) Markov decision

* This research originated from the Big Data project being part of Joint Lab 1, funded by Cisco Systems Germany, at the centre COPICOH, University of Lübeck

processes (FO (PO)MDPs) [7]. In constrast to FO POMDPs, which support *offline* decision making, we propose to support probabilistic *online* decision making, which allows for reacting to observations as well as for query answering.

In the following, we present PDDecMs including the MEU problem. Afterwards, we introduce meuLDJT to solve the lifted temporal MEU problem efficiently.

2 Lifted Temporal Maximum Expected Utility

We recapitulate PDMs [2,3] and then define PDDecMs with action and utility nodes to support decision making. Finally, we define the temporal MEU problem for PDDecMs.

2.1 Parameterised Probabilistic Models

A parameterised probabilistic model (PM) combines first-order logic with probabilistic models, representing first-order constructs using logical variables (logvars) as parameters. We would like to remotely infer the condition of patients with regard to retaining water. To determine the condition of patients, we use the change of their weights and additionally use the change of weights of people living with a patient to reduce the uncertainty for inferring conditions. The cause of an increase in weight could either be overeating or retaining water. In case both persons gain weight, overeating is more likely. Otherwise, only one person gains weight and retaining water is more likely.

Definition 1. *Let* $\mathbf{L}$ *be a set of logvar names,* Φ *a set of factor names, and* $\mathbf{R}$ *a set of random variable (randvar) names. A parameterised randvar (PRV)* $A = P(X^1, ..., X^n)$ *represents a set of randvars behaving identically by combining a randvar* $P \in \mathbf{R}$ *with logvars* $X^1, ..., X^n \in \mathbf{L}$*. If* $n = 0$*, the PRV is parameterless. The domain of a logvar* L *is denoted by* $\mathcal{D}(L)$*. The term* $range(A)$ *provides possible values of a PRV* A*. Constraint* $(\mathbf{X}, C_{\mathbf{X}})$ *allows for restricting logvars to certain domain values and is a tuple with a sequence of logvars* $\mathbf{X} = (X^1, ..., X^n)$ *and a set* $C_{\mathbf{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X^i)$*. The symbol* $\top$ *denotes that no restrictions apply and may be omitted. The term* $lv(Y)$ *refers to the logvars,* $rv(Y)$ *to the randvars,* $gr(Y|C)$ *denotes the set of instances of* Y *with all logvars in* Y *grounded w.r.t. constraint* C*.*

To model the example, we use the randvar names C, LT, S, and W for Condition, LivingTogether, ScaleWorks, and Weight, respectively, and the logvar names X and X'. From the names, we build PRVs $C(X)$, $LT(X, X')$, $S(X)$, and $W(X)$. The domain of X and X' is $\{alice, bob, eve\}$. The range of $C(X)$ is $\{ok, bad\}$, $LT(X, X')$ and $S(X)$ have range $\{true, false\}$, and $W(X)$ has range $\{ok, high\}$.

Definition 2. *We denote a parametric factor (parfactor) g with* $\forall \mathbf{X} : \phi(\mathcal{A}) \mid C$*,* $\mathbf{X} \subseteq \mathbf{L}$ *being a set of logvars over which the factor generalises, and* $\mathcal{A} = (A^1, ..., A^n)$ *a sequence of PRVs. We omit* $(\forall \mathbf{X} :)$ *if* $\mathbf{X} = lv(\mathcal{A})$*. Function* $\phi : \times_{i=1}^{n} range(A^i) \mapsto \mathbb{R}^+$ *with name* $\phi \in \Phi$ *is identical for all grounded instances of* $\mathcal{A}$*. A list of all input-output values is the complete specification for* ϕ*. C is a constraint on* $\mathbf{X}$*. A PM* $G := \{g^i\}_{i=0}^{n-1}$ *is a set of parfactors and semantically represents the full joint probability distribution* $P_G = \frac{1}{Z} \prod_{f \in gr(G)} f$ *with* Z *as a normalisation constant.*

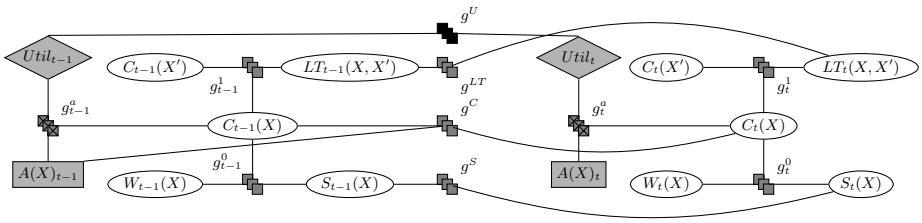

Fig. 1: Retaining water example with action and utility nodes

Now, we build the PM G^{ex} with: $g^0 = \phi^0(C(X), S(X), W(X))|\top$ and $g^1 = \phi^1(C(X), C(X'), LT(X, X'))|\kappa^1$. The constraint κ^1 of g^1 ensures that $X \neq X'$ holds.

We define PDMs based on the first-order Markov assumption and a stationary process. A PDM is composed of a PM G_0 for the initial time step and $G_{\rightarrow}$, which connects two PMs with inter-slice parfactors to model the temporal behaviour. Figure 1 shows a PDM with PRVs connected to parfactors and inter-slice parfactors g^{LT}, g^C, and g^S.

2.2 Parameterised Probabilistic Decision Models

Let us extend PDMs with action and utility nodes, resulting in PDDecMs.

Definition 3. *We represent actions and utilities by PRVs. Let Φ^u be a set of utility factor names. The range of action PRVs is disjoint actions and the range of utility PRVs is $\mathbb{R}$. A parfactor with a utility PRV U is a* utility parfactor. *We denote a utility parfactor u with $\forall \mathbf{X} : \mu(\mathcal{A}) \mid C$, where $U \in \mathcal{A}$ and C a constraint on $\mathbf{X}$. Function $\mu : \times_{i=1}^{n} range(A^i) \mapsto \mathbb{R}$, $A^i \in \mathcal{A}$, with name $\mu \in \Phi^u$ is defined identically for all grounded instances of $\mathcal{A}$ and its output is the additive change of U's value. Therefore, after the evaluation of a μ function, the initial value of U, i, is changed by the output value, j, resulting in the new value of U, which is $i + j$. The default initial utility value is 0. A* parameterised probabilistic decision model (PDecM) *G extends a PM with an additional set G^u of utility parfactors. Let $rv(G^u)$ refer to all probability randvars in G^u. Semantically G^u represents the combination of all utilities $U_G = \sum_{f \in gr(G^u)} f$.*

The μ functions output a utility, i.e., a scalar, which makes comparing utility values easy. Hence, we can easily test how discriminable actions are.

For a PDDecMs, which extends a PDecM to the temporal case similar to a PDM with a PM, to connect two utility PRVs, we define a utility transfer function.

Definition 4. *A utility transfer function λ has utility PRVs $\mathbf{U}$ as input and one utility PRV U_o as output. Additionally, λ can have non utility PRVs as input. λ specifies how the value of U_o is additively changed, possibly depending on non utility PRVs. In this fashion, PDDecMs transfer utility values to the next time step and allow for discounting.*

Figure 1 shows one action (square) and one utility (diamond) PRV in grey, utility parfactors (crosses), and a utility transfer parfactor g^U in black. Assume actions are: A^1 is *visit patient* and A^2 is *do nothing*. For example, patients with a chronic heart

failure might tend to retain water. In case water retention is detected early on, treatment is easier. However, if this water retention remains undetected, water can also retain in the lung, which can lead to a pulmonary edema, making a treatment more costly. More importantly, pulmonary edema is an acute life-threatening condition. A^1 also influences the utility as for a doctor, with limited time, visiting a patient is expensive. Thus, one always needs to consider that alerting a doctor too early generates unnecessary appointments and alerting a doctor too late can have serious consequences for patients.

2.3 Maximum Expected Utility in PDDecMs

PDDecMs encode trade-offs in utility parfactors. Connecting $Util_{t-1}$ and $Util_t$ with a utility transfer parfactor g^U makes utility PRVs time-dependent and allows for discounting. For example, g^U specifies that the value of $Util_{t-1}$ is reduced by 5 and then added to $Util_t$. To select an action, we begin by defining probability and utility queries.

Definition 5. *Given a PDecM G, a ground PRV Q and grounded PRVs with fixed range values $\mathbf{E}$, the expression $P(Q|\mathbf{E})$ denotes a probability query w.r.t. P_G and the expression $U(Q, \mathbf{E})$ refers to a utility w.r.t. U_G.*

Semantically, the expected utility of a PDecM or a PDDecM G, with utility part G^u, under assignment to action PRVs $\mathbf{a}$ is given by:

$$eu(G|\mathbf{a}) = \sum_{r \in range(rv(G^u))} P(r|\mathbf{a}) \cdot U(r, \mathbf{a}) \tag{1}$$

We calculate a belief state and combine the belief state with corresponding utilities. By eliminating all randvars, one obtains the expected utility. We define the MEU problem as:

$$meu[G] = (\arg\max_{\mathbf{a}} eu(G|\mathbf{a}), \max_{\mathbf{a}} eu(G|\mathbf{a})) \tag{2}$$

Equation (2) defines how to calculate the MEU for a PDecM and a PDDecM. Using a utility transfer function, we see the problem as an iterative filtering problem. The expected utility is calculated for one time step and then the utility value is transfered to the next time step. Therefore, the utility value of the latest time step, is the overall utility value. Due to the inherent uncertainty of PDDecMs, calculating the best actions is only feasible for a finite horizon, as one needs to iterate over all possible action assignments.

3 Solving the MEU Problem with meuLDJT

We illustrate how meuLDJT incorporates utilities and solves the MEU problem.

Including Utilities Alg. 1 outlines how meuLDJT includes utilities. Similar to LDJT with PDMs (c.f. [3]), meuLDJT first builds an first-order junction tree (FO jtree) from a PDDecM. Allowing utility parfactors in parclusters is straight forward. While constructing the FO jtree, meuLDJT treats the utility parfactor in the same way as probability parfactors. With the parclusters, meuLDJT distributes local information by message

Algorithm 1 meuLDJT for a PDDecM G, Queries $\{\mathbf{Q}\}_{t=0}^{T}$, and Evidence $\{\mathbf{E}\}_{t=0}^{T}$

procedure MEULDJT($G_0, G_\rightarrow, \{\mathbf{Q}\}_{t=0}^{T}, \{\mathbf{E}\}_{t=0}^{T}$)
  $(J_0, J_t, \mathbf{I}_t) :=$ DFO-JTREE$(G_0, G_\rightarrow)$
  while $t \neq T + 1$ **do**
    $J_t :=$ LJT.EnterEvidence$(J_t, \mathbf{E}_t)$
    $J_t :=$ LJT.PassProbMessages(J_t)
    $J_t :=$ LJT.PassUtilMessages(J_t)
    AnswerQueries$(J_t, \mathbf{Q}_t)$
    $(J_t, t, \alpha[t-1]) :=$ ForwardPass$J_t, t)$

passing. To calculate the probability messages, meuLDJT excludes utility parfactors as they do not influence the probability distributions. Using the probability messages, meuLDJT can calculate the current utility value and distribute the value as long as the utility PRV is in the separator. To calculate utilities, utility parclusters need to know the probability distributions, which is distributed during message passing to each parcluster. Using the probability distributions, meuLDJT calculates for each group a utility value and multiplies the value by the number of groundings. The new utility value is then added to the old utility value. Further, the utility transfer function ensure the transfer of utility values, while LDJT's behaviour ensures preserving the current state.

Answering MEU Queries meuLDJT can answer MEU queries for a finite horizon. The horizon defines how far meuLDJT predicts into the future. For a given horizon, meuLDJT tests all action sequences to find the best action sequence. Hence, meuLDJT constructs all action sequences for a horizon. In general, meuLDJT constructs r^{h+1} action sequences, where r is the number of actions and h the horizon. For each action sequence, meuLDJT enters the sequence as evidence and answers the expected utility query for that sequence. Finally, after having tested all action sequences, meuLDJT returns the best action sequence and the expected utility value.

Assume that $t = 3$ and we only have a horizon of 1 to answer an MEU query, then meuLDJT constructs the action sequences. In this case, there are four action sequences. For example, the first action sequence is $A_3(X) = A^1$ and $A_4(X) = A^1$. Now, meuLDJT enters $A_3(X) = A^1$ in the FO jtree for time step 3. Message passing on the FO jtrees is now performed in two steps. The first step is to calculate probability messages. The second step is to calculate utility messages. meuLDJT uses the probability messages and the evidence, which includes the current action, and distributes the utility through the FO jtree. After message passing, each parcluster can answer queries about its PRVs. Thus, meuLDJT can answer multiple probability and utility queries efficiently, as it can reason over representatives. To proceed in time, meuLDJT uses the *out-cluster* to calculate an α message over the interface PRVs, which now includes $Util_t$. Hence, the current belief state and utility value is stored in the α message and then added to the *in-cluster* of the next time step, in this case $\mathbf{C}_4^3$. Using the utility transfer function, the current FO jtree contains the overall utility value. Now, meuLDJT enters $A_4(X) = A^1$ in the FO jtree for time step 4 and performs the two message passes. Finally, meuLDJT calculates the expected utility value and proceeds with the next action sequence, until meuLDJT has the expected utility values for all four sequences.

Theorem 1. *meuLDJT is sound, i.e., it produces the same result as a ground algorithm.*

Proof sketch. LDJT is sound [3] and meuLDJT uses LDJT for the probability calculations. Thus, marginal *filtering* and *prediction* queries are still sound. To calculate the new utility value, meuLDJT efficiently calculates the utility value and adds it to the old utility value. While calculating the utility value, meuLDJT accounts for the groundings, namely, it calculates the utility value for one representative and multiplies it by the number of groundings. As all instances behave the same, they instances each would contribute the same utility value in a ground model. Hence, by calculating a utility for one representative and multiplying the utility by the number of groundings, meuLDJT obtains the same result a ground algorithm would obtain. Additionally, the message passing inside of a FO jtree ensures that the current utility value is known at all relevant parclusters and the transfer function preserves the value over time.

4 Conclusion

We present PDDecMs, an extension to PDMs, and meuLDJT for sequential probabilistic online decision suport by calculating a solution to the lifted temporal MEU problem. Areas such as healthcare could benefit from the lifting idea for many patients in combination with the efficient handling of temporal aspects of meuLDJT and the support of different kinds of queries. By extending the underlying model with action and utility nodes, complete healthcare processes including treatments can be modelled. Additionally, by maximising the expected utility, meuLDJT can calculate a best action. Further, meuLDJT can efficiently answer a combination of *expected utility*, *filtering*, and *prediction* queries. Thus, meuLDJT can support decision support as well as help to understand the suggested decision by also efficiently answering multiple marginal queries.

We currently check whether meuLDJT can reuse computations from previous expected utility calculations by, e.g., identifying dominant actions for belief state regions.

References

1. Apsel, U., Brafman, R.I.: Extended Lifted Inference with Joint Formulas. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence. pp. 11–18. AUAI Press (2011)
2. Braun, T., Möller, R.: Lifted Junction Tree Algorithm. In: Proceedings of KI 2016: Advances in Artificial Intelligence. pp. 30–42. Springer (2016)
3. Gehrke, M., Braun, T., Möller, R.: Lifted Dynamic Junction Tree Algorithm. In: Proceedings of the 23rd International Conference on Conceptual Structures. pp. 55–69. Springer (2018)
4. Gehrke, M., Braun, T., Möller, R., Waschkau, A., Strumann, C., Steinhäuser, J.: Lifted Maximum Expected Utility. In: Artificial Intelligence in Health. pp. 131–141. Springer International Publishing (2019)
5. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. Ph.D. thesis, University of California, Berkeley (2002)
6. Nath, A., Domingos, P.: A language for relational decision theory. In: Proceedings of the International Workshop on Statistical Relational Learning (2009)
7. Sanner, S., Kersting, K.: Symbolic Dynamic Programming for First-order POMDPs. In: Proceedings of the Twenty-Fourth AAAI. pp. 1140–1146. AAAI Press (2010)