





Lifted Maximum Expected Utility

Marcel Gehrke¹ , Tanya Braun¹ , Ralf Möller¹, Alexander Waschkau²,
Christoph Strumann², and Jost Steinhäuser²

¹ Institute of Information Systems, University of Lübeck, Lübeck, Germany
{gehrke, braun, moeller}@ifis.uni-luebeck.de

² Institute of Family Medicine, University Medical Center Schleswig-Holstein,
Campus Lübeck, Lübeck, Germany
{alexander.waschkau, christoph.strumann, jost.steinhaeuser}@uksh.de

Abstract. The lifted junction tree algorithm (LJT) answers multiple queries efficiently for relational models under uncertainties by building and then reusing a first-order cluster representation. We extend the underlying model representation of LJT, which is called parameterised probabilistic model, to calculate a lifted solution to the maximum expected utility (MEU) problem. Specifically, this paper contributes (i) action and utility nodes for parameterised probabilistic models, resulting in parameterised probabilistic decision models and (ii) meuLJT, an algorithm to solve the MEU problem using parameterised probabilistic decision models efficiently, while also being able to answer multiple marginal queries.

1 Introduction

Areas such as health care and logistics involve probabilistic data with relational aspects and need efficient exact inference algorithms, which allow for decision support. These areas involve many objects in relation to each other with uncertainties about object existence, attribute value assignments, or relations between objects. More specifically, health care systems involve electronic health records (EHRs) (the relational part) for many patients (the objects) and uncertainties [18] due to, e.g., missing information caused by data integration from different hospitals or faulty sensors. Automatically analysing EHRs can improve the care of patients and save time. In this paper, we study the problem of exact decision making under uncertainty in lifted probabilistic models.

Braun and Möller [2] investigate parameterised probabilistic models (PMs) to represent probabilistic relational behaviour, and furthermore introduce the lifted junction tree algorithm (LJT), an exact inference algorithm to answer multiple queries efficiently. Specifically, this paper contributes (i) action and utility nodes for parameterised probabilistic models, resulting in parameterised probabilistic decision models (PDecMs) and (ii) meuLJT, an algorithm to solve the maximum

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.

expected utility (MEU) problem using PDecMs efficiently, while also being able to answer multiple marginal queries.

Action nodes are well-motivated candidates to model, e.g., treatments, while utility nodes can represent, e.g., the well being of patients, risk scores, or treatment costs. With utilities modelling is not restricted to a single particular area, but one can also model a combination of areas, such as well being of patients and risk scores.

Health care needs exact results as approximations might not be good enough [19]. Further, the lifting approach exploits symmetries in the model to reduce the number of instances or patients to perform inference on. Additionally, LJT clusters a model into submodels to efficiently answer queries, like the condition of each patient. Therefore, LJT is suitable to handle health care related data.

In the following, we recapitulate PMs as a representation for relational probabilistic models and introduce PDecMs, by adding actions and utilities to the representation. Afterwards, we formalise the MEU problem and discuss different modelling possibilities, also from an ethical point of view. Lastly, we introduce `meuLJT` to reuse computations and answer multiple queries efficiently.

2 Related Work

We take a look at inference under uncertainty in relational models as well as relational decision support.

First-order probabilistic inference leverages the relational aspect of a static model. For models with known domain size, it exploits symmetries in a model by combining instances to reason with representatives, known as lifting [11]. Poole [11] introduces parametric factor graphs as relational models and proposes lifted variable elimination (LVE) as an exact inference algorithm on relational models. Further, de Salvo Braz [12], Milch et al. [7], and Taghipour et al. [17] extend LVE to its current form. Lauritzen and Spiegelhalter [6] introduce the junction tree algorithm. To benefit from the ideas of the junction tree algorithm and LVE, Braun and Möller [2] present LJT, which efficiently performs exact first-order probabilistic inference on relational models given a set of queries.

Nath and Domingos [8] introduce Markov logic decision networks (MLDNs), which are relational static models with action and utility nodes. Nath and Domingos calculate approximate solutions to the static MEU problem in a completely grounded way [10] based on MLDNs. Another approach of Nath and Domingos include unnecessary groundings [9]. Further, Apsel and Brafman [1] propose an exact lifted solution to the MEU problem based on [8]. These approaches are designed to handle single queries. However, we propose to answer multiple queries efficiently.

Additional research focuses on sequential decision making by investigating first-order (partially observable) Markov decision processes (FO (PO)MDPs) [5, 14, 15], which use lifting techniques from de Salvo Braz, Amir, and Roth [13]. In contrast to FO POMDPs, which perform offline policy iteration, we propose to support probabilistic online planning.

3 Parameterised Probabilistic Models

Based on [4], we recapitulate PMs for relational probabilistic models. PMs combine first-order logic with probabilistic models, representing first-order constructs using logical variables (logvars) as parameters. Let us assume, we would like to remotely infer the condition of patients with regards to water retaining. To determine the condition of patients, we use the change of their weights. An increase in weight could either be caused by overeating or retaining water. Additionally, we use the change of weights of people living with the patient to reduce the uncertainty to infer conditions. In case both persons gain weight, overeating is more likely, while otherwise retaining water is more likely. If a water retention is undetected, it can be an acute life-threatening condition.

People behave in the same way w.r.t. gaining weight if we are interested whether a person retains water. For a water retention, persons gain weight over a few days in a way which would be hard to achieve by overeating each day. Thus, if we are interested whether they retain water, having information about the weight gain of persons is independent of the actual person. Hence, we can have a random variable (randvar) for each person about their current condition. As persons behave the same w.r.t. gaining weight and PMs allow for using logvars as parameters, we can construct a parameterised randvar (PRV) with the persons as logvar for our randvar.

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

To model our scenario, 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 $\{\text{alice}, \text{bob}, \text{eve}\}$. The range of $C(X)$ is $\{\text{normal}, \text{deviation}, \text{retains water}\}$. $LT(X, X')$ and $S(X)$ have range $\{\text{true}, \text{false}\}$ and $W(X)$ has range $\{\text{steady}, \text{falling}, \text{rising}\}$. A constraint $C = (X, \{\text{alice}, \text{bob}\})$ for X allows for restricting X to a subset of its domain, in this case to alice and bob . Using the constraint, the expression $\text{gr}(W(X)|C)$ evaluates to $\{W(\text{alice}), W(\text{bob})\}$. The expression $\text{gr}(W(X)|\top)$ also contains $W(\text{eve})$. Now, we define parametric factors (parfactors), to set PRVs into relation to each other.

Definition 2. We denote a parfactor g with $\forall \mathbf{X} : \phi(\mathcal{A}) | C$. $\mathbf{X} \subseteq \mathbf{L}$ being a set of logvars over which the factor generalises and $\mathcal{A} = (A^1, \dots, A^n)$ a sequence of

PRVs. We omit $(\forall \mathbf{X} :)$ if $\mathbf{X} = lv(\mathcal{A})$. A function $\phi : \times_{i=1}^n range(A^i) \mapsto \mathbb{R}^+$ with name $\phi \in \Phi$ is defined identically 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 $G := \{g^i\}_{i=0}^n$ is a set of parfactors and semantically represents the full joint probability distribution $P_G = \frac{1}{Z} \prod_{f \in gr(G)} f$ where Z is a normalisation constant.

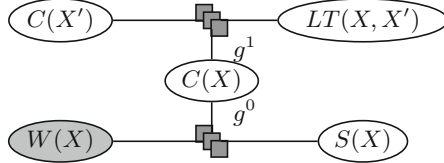


Fig. 1. Parfactor graph for G^{ex} , the weight is observable

Now, we build the model G^{ex} of our example with the parfactors:

$$g^0 = \phi^0(C(X), S(X), W(X)) | \top \text{ and } g^1 = \phi^1(C(X), C(X'), LT(X, X')) | \kappa^1$$

We omit the concrete mappings of ϕ^0 and ϕ^1 . Parfactor g^0 has the constraint \top , meaning it holds for *alice*, *bob*, and *eve*. The constraint κ^1 of g^1 ensures that $X \neq X'$ holds. Figure 1 depicts G^{ex} as a parfactor graph and shows PRVs, which are connected via undirected edges to parfactors, with $W(X)$ being observable. We can observe the weight of patients. The remaining PRVs are latent.

The semantics of a model is given by grounding and building a full joint distribution. In general, queries ask for a probability distribution of a randvar using a model’s full joint distribution and fixed events as evidence.

Definition 3. Given a PM G , a query term Q (ground PRV), and events $\mathbf{E} = \{E^i = e^i\}_i$ (ground PRVs with fixed range values), the expression $P(Q|\mathbf{E})$ denotes a query w.r.t. P_G .

In our example, a query is $P(C(bob)|W(bob) = steady)$, asking for the probability distribution of *bob*’s condition given information about his weight.

4 Lifted Maximum Expected Utility

In this section, we introduce actions and utilities to PMs and show how to solve the MEU problem, by formalising the problem. Further, we discuss different modelling possibilities with PDecMs.

4.1 Parameterised Probabilistic Decision Models

Let us extend PMs with action and utility nodes, resulting in PDecMs.

Definition 4. 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 that maps to a utility PRV U is a utility parfactor. We denote a utility parfactor u with $\forall \mathbf{X} : \mu(\mathcal{A}) \mid C$, where C a constraint on \mathbf{X} . Function $\mu : \times_{i=1}^{n-1} \text{range}(A^i) \mapsto \mathbb{R}$, $A^i \in \mathcal{A}$, with name $\mu \in \Phi^u$ is defined identically for all grounded instances of \mathcal{A} and its output is the value of U . A PDecM G is a PM with an additional set G^u of utility parfactors. Let $rv(G^u)$ refer to all probability randvars in G^u . Then, G^u semantically 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. Further, a scalar allows for testing whether utilities are within an ϵ margin of each other, making them hardly discriminable. With utilities incorporated, we look at actions. To model actions, we introduce an action PRV with the actions in its range. Hence, we have one PRV, which models disjoint actions. To execute an action, we set the value of the action PRV to the action, which we want to perform, similar to providing evidence for marginal queries. Thus, the range of an action PRV $A(X)$ consists of different actions, lets say A^1, \dots, A^n , and by setting $A(X)$ to the action, lets say A^1 ($A(X) = A^1$), we can select the action we would like to perform

Let us now extend the example with action and utility nodes. In Fig. 2, one can see one action node (square), one utility node (diamond), and one utility parfactor (crosses). In our example, the action PRV $A(X)$ has two actions in its range, namely A^1 is visit patient and action A^2 is do nothing. Obviously, other actions could also be included in the model, e.g., diet related actions or obtaining a more accurate scale.

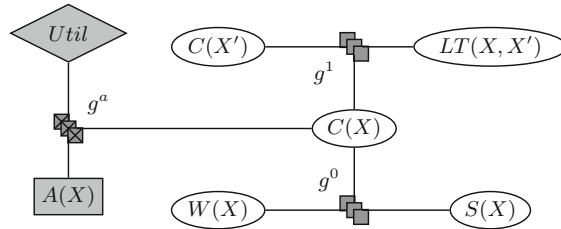


Fig. 2. Retaining water example with action and utility nodes in grey

In our example, the condition of patients and A^1 influence the utility. 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. In addition to the condition of patients, A^1 also influences the utility as a doctor, with limited time, visiting a patient is expensive. Thus, one always needs to consider that alerting the doctor too early generates unnecessary costs and alerting the doctor too late can have serious consequences for the patient.

4.2 Maximum Expected Utility

To select the best action, we define queries and expected utility on a PDecM.

Definition 5. *Given a PDecM G , a query term Q , and events \mathbf{E} , the expression $P(Q|\mathbf{E}, \mathbf{s})$ denotes a probability query w.r.t. P_G . Given an assignment \mathbf{a} for actions, the expression $U(Q, \mathbf{E}, \mathbf{a})$ refers to a utility w.r.t. U_G . The expected utility of G is defined by*

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

The inner part of the summation in Eq. (1) calculates a belief state $P(v|\mathbf{a})$ and combines it with corresponding utilities $U(v, \mathbf{a})$. By summing over all randvars from G^u , one obtains a scalar representing the expected utility. LVE allows for exactly computing an expected utility. Based on expected utility, we define the MEU as follows.

Definition 6. *Given a PDecM G , the MEU problem is given by*

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

Equation (2) suggest a naive algorithm defining how to calculate the MEU, namely by iterating over all possible action configurations, computing an expected utility for each configuration using LVE, an iteration that one cannot avoid if asking for an exact solution. The action assignment that maximises the expected utility is selected. As the utility value is a scalar, the expected utility w.r.t. configurations can be easily compared. Therefore, we also can easily determine configurations whose expected utility lie within an ϵ margin. In case different actions lie within an ϵ margin, the actions are hardly discriminable w.r.t. utilities.

The action PRV in G^{ex} has two possible actions. By setting $A(X) = A1$, we turn on A^1 . By setting $A(X) = A2$, we turn on A^2 . Thus, in our example to calculate the MEU, we need to iterate over two action assignments. For each expected utility, we obtain a scalar, allowing us to easily compare them and return the action with the MEU and the actual expected utility value. If all patients behave the same, we only need to iterate over two actions. In case we obtain different evidence for lets say two groups of patients, X^1 and X^2 , we need to iterate over the actions for both groups. Hence, we would need to iterate

over $\{A(X^1) = A1, A(X^2) = A1\}$, $\{A(X^1) = A2, A(X^2) = A1\}$, $\{A(X^1) = A1, A(X^2) = A2\}$, and $\{A(X^1) = A2, A(X^2) = A2\}$. In general, we need to iterate over r^n actions, where r is the number of actions in the range of an action PRV and n the number of different groups. Assuming, we have ten patients in two groups and two possible actions. Solving the MEU in a lifted way, we need to iterate over $2^2 = 4$ actions. Without the lifting idea, we would need to iterate over $2^{10} = 1024$ actions. Therefore, solving the MEU problem in a lifted way makes the problem manageable.

4.3 How to Model Utilities in a Medical Context

For decision support in a medical context, the model has to take into account the prevalence, i.e., the probability, of the diseases or health related problems to be identified. The prevalence does not only depend on the value of data but also on the source of data. For example, to identify a coronary heart diseases the prevalence is higher if the data comes from a chest pain unit compared to examinations from general practice [16]. In this context, the knowledge of the sensitivity and specificity of the analytical model and the prevalence is very important. Ideally, the model should inform the physician about its sensitivity and specificity to clarify the probability of a false positive result for each patient regarding the pre/post test probability. These information can help to plan further treatment and diagnostic decisions. The aim of the model should be to avoid unnecessary examinations and thus costs. Further, decision making should not unsettle the patient on the one hand, but on the other hand detect serious conditions timely.

As PDecMs can model different influences, we can take prevalence into account. Thus, we need to model different PRVs for different sources, which then depending on the value of the test results, having different impact on the condition of a patient. Further, there are two different kinds of queries for PDecMs, namely utility and probability queries. Thus, we can also state marginal queries. Having marginal queries, we can also query the current belief of the condition of a patient as well as the condition of a patient after an action, i.e., treatment or test, is performed.

5 Solving the MEU Problem and Answer Multiple Marginal Queries Efficiently

In this section, we recapitulate LJT [3] to answer queries for PMs and introduce meuLJT to solve the MEU problem and answer multiple marginal queries using PDecMs efficiently.

5.1 Lifted Junction Tree Algorithm

LJT provides efficient means to answer queries $P(Q^i|\mathbf{E})$, with $Q^i \in \mathbf{Q}$ a set of query terms, given a PM G and evidence \mathbf{E} , by performing the following steps:

- (i) Construct an first-order junction tree (FO jtree) J for G .
- (ii) Enter \mathbf{E} in J .
- (iii) Pass messages.
- (iv) Compute answer for each query $Q^i \in \mathbf{Q}$.

We first define an FO jtree and then go through each step. To define an FO jtree, we define parameterised clusters (parclusters), nodes of an FO jtree.

Definition 7. A parcluster \mathbf{C} is defined by $\forall \mathbf{L} : \mathbf{A} | \mathbf{C}$. \mathbf{L} is a set of logvars, \mathbf{A} is a set of PRVs with $lv(\mathbf{A}) \subseteq \mathbf{L}$, and C a constraint on \mathbf{L} . We omit $(\forall \mathbf{L} :)$ if $\mathbf{L} = lv(\mathbf{A})$. A parcluster \mathbf{C}^i can have parfactors $\phi(\mathcal{A}) | C^\phi$ assigned given that (i) $\mathcal{A}^\phi \subseteq \mathbf{A}$, (ii) $lv(\mathcal{A}^\phi) \subseteq \mathbf{L}$, and (iii) $C^\phi \subseteq C$ holds. We call the set of assigned parfactors a local model G^i .

An FO jtree for a PM G is $J = (\mathbf{V}, \mathbf{P})$ where J is a cycle-free graph, the nodes \mathbf{V} denote a set of parclusters, and \mathbf{P} is a set of edges between parclusters. J must satisfy the following properties: (i) A parcluster \mathbf{C}^i is a set of PRVs from G . (ii) For each parfactor $\phi(\mathcal{A}) | C$ in G , \mathcal{A} must appear in some parcluster \mathbf{C}^i . (iii) If a PRV from G appears in two parclusters \mathbf{C}^i and \mathbf{C}^j , it must also appear in every parcluster \mathbf{C}^k on the path connecting nodes i and j in J (running intersection). The separator \mathbf{S}^{ij} of edge $i - j$ is given by $\mathbf{C}^i \cap \mathbf{C}^j$ containing shared PRVs.

LJT constructs an FO jtree using a first-order decomposition tree, enters evidence in the FO jtree, and to distribute local information of the nodes through the FO jtree, passes messages through an *inbound* and an *outbound* pass. To compute a message, LJT eliminates all non-separator PRVs from the parcluster’s local model and received messages. After message passing, LJT answers queries. For each query, LJT finds a parcluster containing the query term and sums out all non-query terms in its local model and received messages.

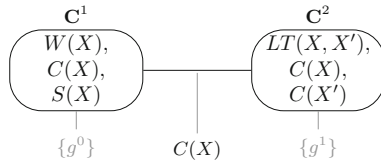


Fig. 3. FO jtree for G^{ex} (local models under the parclusters)

Figure 3 shows an FO jtree of G^{ex} with the local models of the parclusters and the separators as labels of edges. During the *inbound* phase of message passing, LJT sends messages from \mathbf{C}^1 to \mathbf{C}^2 and for the *outbound* phase a message from \mathbf{C}^2 to \mathbf{C}^1 . If we would like to know whether $S(bob)$ holds, we query $P(S(bob))$ for which LJT can use parcluster \mathbf{C}^1 . LJT sums out $C(X)$, $W(X)$, and $S(X)$ where $X \neq bob$ from \mathbf{C}^1 ’s local model G^1 , $\{g^0\}$, combined with the received messages.

5.2 meuLJT

Now, we introduce meuLJT to solve the MEU problem. For now, we restrict a PDecM G to have at most one utility PRV and one utility parfactor. The basic step of meuLJT are similar to LJT, namely:

- (i) Construct an FO jtree J for G .
- (ii) Enter evidence and actions in J .
- (iii) Pass messages.
- (iv) Compute answer queries.

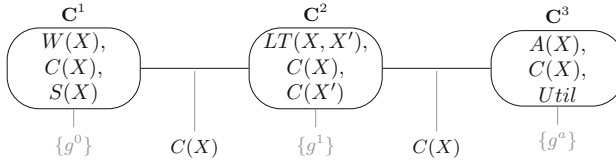


Fig. 4. FO jtree for the PDecM G^{ex} (local models under the parclusters)

Let us now develop an idea about how meuLJT solves the lifted MEU problem. Figure 4 shows an FO jtree for the PDecM G^{ex} . Compared to the FO jtree from Fig. 3, we see an additional parcluster C^3 with the utility parfactor. To construct the FO jtree, meuLJT treats the utility parfactor as any other parfactor. Including utility parfactored in the parcluster definition is straight forward. Using the FO jtree, meuLJT distributes local information by message passing. To calculate the probability messages, meuLJT also performs a message pass. During the message pass, meuLJT excludes utility parfactored as they do not influence the probability distributions and we only have one utility parfactor and one utility PRV. Hence, during the *inbound* pass, C^2 receives a message over $C(X)$ from C^1 and an empty message from C^3 . For the *outbound* pass C^2 sends messages over $C(X)$ to C^1 and C^3 . To calculate utilities, utility parfactored need to know the probability distributions, which is distributed by message passing also to parclusters with utility parfactored. Now, meuLJT can use C^1 and C^2 to answer marginal queries and C^3 to answer *expected utility* queries. Given new evidence or a new action assignment meuLJT has to recompute messages. Hence, for each action assignment meuLJT can answer the *expected utility* query and efficiently answer multiple marginal queries, e.g., of the condition of patients.

In our example, we have two action sequences $\{A(X) = A1\}$ and $\{A(X) = A2\}$, if all patients behave the same. To calculate the MEU, meuLJT has to iterate over all action sequences and calculate the corresponding expected utility. For the first action sequence, meuLJT enters $\{A(X) = A1\}$ as evidence in the FO jtree from Fig. 4. After the message pass, meuLJT uses C^3 to answer the *expected utility* query for action $\{A(X) = A1\}$. C^3 received the current belief

state during message passing and has the current action due to the evidence. Thus, all required information to calculate the expected utility are present.

For the second action sequence, `meuLJT` enters $\{A(X) = A2\}$ as evidence in the FO jtree from Fig. 4. Normally `meuLJT` would need to perform a new message pass, but the evidence does not change any calculations of the probability messages in this case. Thus, `meuLJT` can reuse the already performed message pass. Hence, `meuLJT` can directly use C^3 to answer the *expected utility* query for action $\{A(X) = A2\}$. C^3 received the current belief state during message passing and has the current action due to the evidence. Thus, all required information to calculate the expected utility are present. Having the expected utility for both actions, `meuLJT` selects the action with the MEU. In case, we have more actions or have more groups of patients, `meuLJT` has more action sequences to iterate over. In general, as long as we only have one action PRV and one utility PRV, and both occur only in a utility parfactor, `meuLJT` can reuse the message pass and thereby, prevent redundant calculations.

All in all, `meuLJT` directly reasons over all patients instead of reason over each patient individually. Additionally, `meuLJT` can provide alerts based on observations of each patient. Apsel and Brafman [1] extend C-FOVE to solve MEU queries, which significantly outperforms the propositional case. Braun and Möller [2] show that LJT outperforms GC-FOVE, an extension to C-FOVE, for multiple queries. Therefore, `meuLJT` is well-suited to support lifted decision making and answering multiple marginal queries.

6 Conclusion

We present `meuLJT` to support lifted decision making by calculating a solution to the MEU problem efficiently. Areas like health care benefit from the lifting idea for many patients and the support of different kinds of queries. By extending the underlying model with action and utility nodes, complete health care processes including treatments can be modelled. Additionally, by maximising the expected utility, `meuLJT` can calculate the best action.

The next step is to extend `meuLJT` and the underlying problem to the temporal case. Further, we investigate whether, for our application, evidence can reduce the MEU problem roughly from a POMDP to an MDP.

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: Friedrich, G., Helmert, M., Wotawa, F. (eds.) KI 2016. LNCS (LNAI), vol. 9904, pp. 30–42. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46073-4_3
3. Braun, T., Möller, R.: Parameterised queries and lifted query answering. In: IJCAI, pp. 4980–4986 (2018)

4. Gehrke, M., Braun, T., Möller, R.: Lifted dynamic junction tree algorithm. In: Chapman, P., Endres, D., Pernelle, N. (eds.) ICCS 2018. LNCS (LNAI), vol. 10872, pp. 55–69. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91379-7_5
5. Joshi, S., Kersting, K., Khardon, R.: Generalized first order decision diagrams for first order Markov decision processes. In: IJCAI, pp. 1916–1921 (2009)
6. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Stat. Soc. Ser. B (Methodol.)* **50**(2), 157–224 (1988)
7. Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., Kaelbling, L.P.: Lifted probabilistic inference with counting formulas. In: Proceedings of AAI, vol. 8, pp. 1062–1068 (2008)
8. Nath, A., Domingos, P.: A language for relational decision theory. In: Proceedings of the International Workshop on Statistical Relational Learning (2009)
9. Nath, A., Domingos, P.: Efficient lifting for online probabilistic inference. In: Proceedings of the Twenty-Fourth AAI Conference on Artificial Intelligence, pp. 1193–1198. AAI Press (2010)
10. Nath, A., Domingos, P.M.: Efficient belief propagation for utility maximization and repeated inference. In: Proceedings of the Twenty-Fourth AAI Conference on Artificial Intelligence, pp. 1187–1192. AAI Press (2010)
11. Poole, D.: First-order probabilistic inference. In: Proceedings of IJCAI, vol. 3, pp. 985–991 (2003)
12. de Salvo Braz, R.: Lifted first-order probabilistic inference. Ph.D. thesis, Ph. D. dissertation, University of Illinois at Urbana Champaign (2007)
13. de Salvo Braz, R., Amir, E., Roth, D.: MPE and partial inversion in lifted probabilistic variable elimination. In: AAI, vol. 6, pp. 1123–1130 (2006)
14. Sanner, S., Boutilier, C.: Approximate solution techniques for factored first-order MDPs. In: 17th International Conference on Automated Planning and Scheduling, ICAPS 2007, pp. 288–295. AAI Press (2007)
15. Sanner, S., Kersting, K.: Symbolic dynamic programming for first-order POMDPs. In: Proceedings of the Twenty-Fourth AAI Conference on Artificial Intelligence, pp. 1140–1146. AAI Press (2010)
16. Steinhäuser, J., Kühlein, T.: Role of the general practitioner. In: Gombotz, H., Zacharowski, K., Spahn, D.R. (eds.) Patient Blood Management, pp. 61–65. Thieme, Stuttgart (2015)
17. Taghipour, N., Fierens, D., Davis, J., Blockeel, H.: Lifted variable elimination: decoupling the operators from the constraint language. *J. Artif. Intell. Res.* **47**(1), 393–439 (2013)
18. Theodorsson, E.: Uncertainty in measurement and total error: tools for coping with diagnostic uncertainty. *Clin. Lab. Med.* **37**(1), 15–34 (2017)
19. Wemmenhove, B., Mooij, J.M., Wiegerinck, W., Leisink, M., Kappen, H.J., Neijt, J.P.: Inference in the promedas medical expert system. In: Bellazzi, R., Abu-Hanna, A., Hunter, J. (eds.) AIME 2007. LNCS (LNAI), vol. 4594, pp. 456–460. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73599-1_61