Towards Tractable Dynamic Decision Making With Circuits

Gabriele Venturato[1] **Vincent Derkinderen**[1] **Pedro Zuidberg Dos Martires**[2] **Luc De Raedt**[1,2]

[1]KU Leuven, Belgium
[2]Örebro University, Sweden

Abstract

A fundamental problem tackled by artificial intelligence is decision making under uncertainty in dynamic environments. For example, a robot may need to autonomously reason on where to move (decision) at each time step (dynamic) while maximising the expected utility of the performed actions, and taking into account the inherent noisiness of the world (uncertainty). *Decision circuits* have been shown to be a useful modelling tool in such settings, with the caveat that they do not treat time as a first-class citizen. We repair this omission by introducing *dynamic decision circuits* (DDCs). More specifically, we show how to obtain DDCs from dynamic decision-theoretic Bayesian networks via knowledge compilation and how to perform inference in DDCs using the algebraic model counting framework — a generalisation of weighted model counting.

1 INTRODUCTION

Bayesian networks (BNs) have widely been adopted as the go-to formalism to model real-world processes under uncertainty [Koller and Friedman, 2009, Russell and Norvig, 2020]. Since their inception [Pearl, 1988], they have been extended in various ways. In this paper we focus on their dynamic (*i.e.*, temporal) [Dean and Kanazawa, 1989, Murphy, 2002] and decision-theoretic [Howard and Matheson, 1984] extensions. BNs that are both dynamic and decision-theoretic have been dubbed *dynamic decision networks* (DDNs) and are capable of modelling a wide range of problems, including *partially observable Markov decision processes* [Russell and Norvig, 2020].

Unfortunately, inference (and by extension learning) in DDNs is computationally hard. We will mitigate this hardness by means of *knowledge compilation* [Darwiche and Marquis, 2002], a state-of-the-art technique to perform inference in BNs. The idea is to compile a BN into a so-called *arithmetic circuit* (a computationally hard step), that can then be evaluated cheaply to compute probabilities.

Our main contribution is the introduction of *dynamic decision circuits* (DDCs). In the same way arithmetic circuits represent Bayesian networks, our newly-introduced DDCs represent dynamic decision networks for the full observability setting. In order to evaluate DDCs, *i.e.*, to compute expected utilities, we leverage the *algebraic model counting* framework [Kimmig et al., 2017].

Furthermore, we provide an online planning algorithm for *fully observable Markov decision processes* (MDPs) that reduces reasoning to inference on DDCs. We also provide preliminary experiments and discuss limitations.

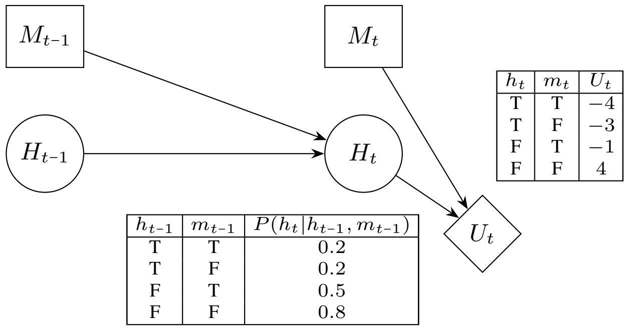
2 RELATED WORK

This work takes inspiration from Derkinderen and De Raedt [2020], who used algebraic model counting with arithmetic circuits in order to compute expected utilities and maximise a decision. While their work is restricted to one-shot problems, we instead consider a setting with multiple decisions spanning over time, thus tackling MDPs.

In Hoey et al. [1999], compiled structures similar to those we introduce, were used to solve MDPs. Feng and Hansen [2002] later extended this approach using forward search to avoid exploring the whole state space, as we do in our online algorithm. In contrast to our work, they iteratively manipulate (smaller) circuits while we create one larger circuit that we evaluate through bottom-up passes.

Recurrent sum-product-max networks [Tatavarti et al., 2021] have recently been introduced. Importantly, these circuits are learned approximately — both the structure and their parameters — from fully observed data. In our case the DDC structure is not learned but compiled from a given model. In addition, we do emphasise the power of algebraic model

Accepted for the 5th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2022).



h_t	m_t	U_t
T	T	-4
T	F	-3
F	T	-1
F	F	4

$h_{t\text{-}1}$	$m_{t\text{-}1}$	$P(h_t \mid h_{t\text{-}1}, m_{t\text{-}1})$
T	T	0.2
T	F	0.2
F	T	0.5
F	F	0.8

Figure 1: Dynamic decision network (DDN) for Example 1.

counting, which we envision will allow learning parameters from partially observed data with the gradient semiring.

3 PRELIMINARIES

We use upper case symbols to denote random variables, lower case for their instantiation, and bold for sets of variables. Moreover, we focus on discrete random variables and assume *w.l.o.g.* that they are Boolean, *i.e.*, X is x or $\neg x$.

3.1 DYNAMIC DECISION NETWORKS

Decision networks [Howard and Matheson, 1984] are a class of probabilistic graphical models for decision making[1]. They are Bayesian networks enriched with additional node types for decisions (rectangles) and utilities (diamonds).

Dynamic decision networks (DDNs) [Kanazawa and Dean, 1989] are decision networks that explicitly involve time and that are usually represented as a transition from time $t-1$ to time t where the Markovian property is assumed — *i.e.*, the future is independent from the past given the present.

Example 1 (Monkey) *The DDN depicted in Figure 1 involves a monkey named Abu who is trying to hit (H) you with some suspicious mud. You can decide to move (M) or not. Your decision and your state — that is, if you have been hit previously or not — influence the next state. If you get hit, Abu celebrates and it is less likely you will get hit in the next time step. On the other hand, if he misses he gets angrier and focus more on his task, increasing the probability to hit you. Of course moving decreases the probability of being hit. There is a utility (U) associated with every state and decision. For example, moving requires energy even if you did not get hit, while the best reward is when you both do not move and do not get hit.*

Like in Bayesian networks, each probabilistic node has a *conditional probability table* (CPT). In addition, we associate with each utility node a *conditional utility table* (CUT). We assume *w.l.o.g.* that there is only one utility node and hence one CUT. Finally, to fully express a dynamic decision problem we need a starting state. This is the *prior network*, which is the transition network without the previous time step, *e.g.*, in this case we can start with $P(H_0{=}\neg h) = 1$.

[1]They are often also called "influence diagrams". We follow Russell and Norvig [2020]'s choice of using decision networks because it is a more descriptive term.

3.2 ALGEBRAIC MODEL COUNTING

Algebraic model counting [Kimmig et al., 2017] is a generalisation of weighted model counting (Appendix A.1).

Definition 1 (Algebraic model counting (AMC)) *Given a commutative semiring $S = (\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, a propositional logic theory T over a set of variables $\mathcal{V}$, and a labelling function $\alpha : \mathcal{L} \mapsto \mathcal{A}$, mapping literals $\mathcal{L}$ of the variables in $\mathcal{V}$ to values from the semiring domain $\mathcal{A}$, the task of algebraic model counting is to compute:*

$$AMC(S, T, \alpha) = \bigoplus_{m \in \mathcal{M}(T)} \bigotimes_{\ell \in m} \alpha(\ell). \qquad (1)$$

where $\mathcal{M}(T)$ is the set of models of T.

Many problems, such as sensitivity analysis and gradient computation, can be tackled defining the appropriate semiring and labelling function [Kimmig et al., 2017]. We exploit this in Section 4.

Algebraic model counting is in general #P-complete. However, via knowledge compilation, the logical theory can first be compiled into an appropriate circuit such that counting becomes tractable — in the size of the circuit.

4 DYNAMIC DECISION CIRCUITS

We devise dynamic decision circuits with two principles in mind. First, we want to exploit the repetition over time: because the transition function is stationary, we can compile it into a circuit only once, and re-use it at each time step. This avoids the circuit exponentially exploding over the future lookahead [Vlasselaer et al., 2016]. Second, the evaluation of the circuit should correspond to a Bellman update. This means that an infinite lookahead leads to an optimal policy [Bellman, 1957].

In order to make optimal decisions we must compute the maximum expected utility (MEU). In the dynamic context, the utility is given not only by the reward at the current time step t, but also by the reward expected from the future. Namely, for a given state $\mathbf{x}$ and decisions $\mathbf{d}$ taken in $\mathbf{x}$, we have the expected utility:

$$U_t(\mathbf{x}, \mathbf{d}) = \mathsf{CUT}_{u|\mathbf{x},\mathbf{d}} + MEU_{t+1}(\mathbf{x}, \mathbf{d}) \qquad (2)$$

where $\mathsf{CUT}_{u|\mathbf{x},\mathbf{d}}$ is the value in the corresponding node's conditional utility table, and

$$MEU_t(\mathbf{x},\mathbf{d}) = \max_{\mathbf{d}'} \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x},\mathbf{d}) U_t(\mathbf{x}',\mathbf{d}') \qquad (3)$$

is the maximum expected utility achievable from taking the decision $\mathbf{d}$ on state $\mathbf{x}$, and in which $P(\mathbf{x}'|\mathbf{x},\mathbf{d})$ stands for $P(\mathbf{X}_t{=}\mathbf{x}'|\mathbf{X}_{t-1}{=}\mathbf{x},\mathbf{D}_{t-1}{=}\mathbf{d})$. These equations correspond to the Bellman equations. Specifically, Equation 3 corresponds to the Q-function, where $\mathsf{CUT}_{u|\mathbf{x},\mathbf{d}}$ is the immediate reward, and the recursive call MEU_{t+1} is the maximum expected reward that can be obtained from the future.

Given a dynamic decision network, we want to produce a dynamic decision circuit which exploits algebraic model counting in order to make decisions in a tractable way. Namely, such that Equation 3 can be computed as follows.

$$MEU_t(\mathbf{x},\mathbf{d}) = AMC(S_{meu}, \Delta_{|\mathbf{x},\mathbf{d}}, \alpha_t) \qquad (4)$$

In this equation, S_{meu} is the semiring used by Derkinderen and De Raedt [2020] to compute the maximum expected utility (see Appendix A.2). However, because we consider multiple decisions spanning over time, we have to define the circuit $\Delta_{|\mathbf{x},\mathbf{d}}$ representing the transition, and the labelling function α_t. We do this in the following two sections.

4.1 KNOWLEDGE COMPILATION

The second element of the AMC call in Equation 4 is the circuit $\Delta_{|\mathbf{x},\mathbf{d}}$, short for $\Delta_{|\mathbf{X_{t-1}}=\mathbf{x},\mathbf{D_{t-1}}=\mathbf{d}}$. Specifically, it is the circuit representing the transition from time step $t-1$ to time step t, where we fix the previous state $\mathbf{X}_{t-1}{=}\mathbf{x}$ and the previous decision $\mathbf{D}_{t-1}{=}\mathbf{d}$.

In order to obtain such a circuit from the dynamic decision network (input), we first *encode* the network into a logic theory T. Afterwards, T is *(knowledge) compiled* into a circuit Δ. For the encoding, we take inspiration from Darwiche [2003] and adapt it to a DDN. The adaptation is straightforward, therefore we will explain it encoding Example 1.

Example 2 (Encoding for the Monkey DDN) *We use the subscript "-1" to indicate variables from the previous time step ($t-1$), and no subscript to indicate those in the current time step (t). Moreover, we use indicator variables λs and network parameters θs as in the original work.*

$$\lambda_{m_{-1}} \leftrightarrow \theta_{m_{-1}}, \qquad \lambda_{h_{-1}} \leftrightarrow \theta_{h_{-1}}, \qquad \lambda_m \leftrightarrow \theta_m$$

$$\left.\begin{aligned} \lambda_h \leftrightarrow \; & (\lambda_{h_{-1}} \wedge \lambda_{m_{-1}} \wedge \theta_{h|h_{-1},m_{-1}}) \\ & \vee(\lambda_{h_{-1}} \wedge \lambda_{\neg m_{-1}} \wedge \theta_{h|h_{-1},\neg m_{-1}}) \\ & \vee(\lambda_{\neg h_{-1}} \wedge \lambda_{m_{-1}} \wedge \theta_{h|\neg h_{-1},m_{-1}}) \\ & \vee(\lambda_{\neg h_{-1}} \wedge \lambda_{\neg m_{-1}} \wedge \theta_{h|\neg h_{-1},\neg m_{-1}}) \end{aligned}\right\} \textit{Trans.}$$

$$\left.\begin{aligned} \theta_{u|h,m} &\leftrightarrow \lambda_h \wedge \lambda_m \\ \theta_{u|h,\neg m} &\leftrightarrow \lambda_h \wedge \lambda_{\neg m} \\ \theta_{u|\neg h,m} &\leftrightarrow \lambda_{\neg h} \wedge \lambda_m \\ \theta_{u|\neg h,\neg m} &\leftrightarrow \lambda_{\neg h} \wedge \lambda_{\neg m} \end{aligned}\right\} \textit{Utilities}$$

The use of S_{meu} for decision making is only valid in a context where the variable ordering is constrained, as the associativity property does not hold in general [Derkinderen and De Raedt, 2020]. For the compilation process, we will therefore target circuits of the class $\mathbf{X}$-constrained SDD [Oztok et al., 2016], which allows us to enforce the ordering constraint.

4.2 LABELLING

The third and last element of the AMC call in Equation 4 is the labelling function α_t which is recursively defined on time as follows.

Definition 2 (Labelling function α_t) *Let us denote with $\mathcal{L}$ the set of literals of a circuit Δ representing the transition from one time step to the next, of a dynamic decision process. The labelling function $\alpha_t : \mathcal{L} \mapsto (p, eu, \mathcal{D})$, for the S_{meu} semiring — where p is a probability, eu is a utility value, and $\mathcal{D}$ is a set of decisions —is defined as follows for every possible instantiation $\mathbf{x}$ of $\mathbf{X}$, and $\mathbf{d}$ of $\mathbf{D}$.*

$$\begin{aligned} &\alpha_t(\lambda_\ell) = (1,0,\emptyset) && \forall \ell \in \mathcal{L} \\ &\alpha_t(\theta_d) = (1,0,\{d\}) && \forall d \in \mathbf{d} \\ &\alpha_t(\theta_{x|Pa(X)}) = (\mathit{CPT}_{x|Pa(X)},0,\emptyset) && \forall x \in X, X \in \mathbf{X} \\ &\alpha_t(\theta_{x|\mathbf{x}_{-1},\mathbf{d}_{-1}}) = (\mathit{CPT}_{x|\mathbf{x}_{-1},\mathbf{d}_{-1}},0,\emptyset) && \forall x \in \mathbf{x} \\ &\alpha_t(\theta_{u|\mathbf{x},\mathbf{d}}) = (1, U_t(\mathbf{x},\mathbf{d}),\emptyset) \end{aligned}$$

where $Pa(X)$ is the set of parents of X, and $U_t(\mathbf{x},\mathbf{d})$ from Equation 2 can be re-written as

$$U_t(\mathbf{x},\mathbf{d}) = \mathit{CUT}_{u|\mathbf{x},\mathbf{d}} + AMC(S_{meu}, \Delta_{|\mathbf{x},\mathbf{d}}, \alpha_{t+1})$$

The way the labelling function α assigns values is simple. The probabilities originate from the corresponding CPTs, and when there is no probability for that parameter (λ_ℓ, θ_d, and $\theta_{u|\mathbf{x},\mathbf{d}}$), the neutral value 1 is used as to not influence the probability calculations. For the utilities it is always 0, except for $\theta_{u|\mathbf{x},\mathbf{d}}$. Finally, for the decisions it is always $\emptyset$, except for the decision parameters θ_d. This allows the S_{meu} semiring to keep track of the best set of decisions in that (sub)circuit.

Note that, since AMC returns a triple, the final usage must select the proper component. For example, in Equation 4 we are interested in the expected utility, that is the second component of the result returned by the AMC call.

5 EXPERIMENTS

Dynamic decision circuits can be used in an online planning algorithm to retrieve the next best action — Algorithm 1 in Appendix A.4 provides the pseudocode for such integration.

Table 1: Compilation time, size of the circuit, and average time per action with respect to the horizon length.

Model	KC (s)	$\|\Delta\|$	Horizon (s)				
			0	1	2	3	4
monkey	0.007	637	$\mathbf{0.001 \pm 0}$	0.008 ± 0	0.036 ± 0	0.146 ± 0	0.589 ± 0
row	0.053	5 968	0.007 ± 0	0.061 ± 0	0.258 ± 0	$\mathbf{1.428 \pm 0.6}$	7.555 ± 3.4
grid	0.786	95 325	0.015 ± 0	$\mathbf{0.524 \pm 0.1}$	6.888 ± 2.6	64.520 ± 26.9	591.472 ± 225.1

Table 2: Compilation time, circuit size, and average time per action with respect to the size of a grid model, with fixed horizon ($H = 1$).

n	KC (s)	$\|\Delta\|$	Act (s)
2	0.27	49 969	0.55 ± 0
3	2.98	226 845	0.72 ± 0.1
4	97.64	9 177 175	1.36 ± 0.3
5	out of memory		—

Notice that, in this way, the planning process is exact. However, to stop the unbounded recursion in the labelling function, we need to limit the *lookahead horizon*. All the definitions given in Section 4 can be easily adapted to stop the recursion at the appropriate depth. For simplicity, we also do not use any discount factor — despite being common practice in these type of problems — but it can be easily added to the labelling definition where the recursive call happens.

We implemented the online planning algorithm and we designed two sets of preliminary experiments in order to validate it.

The first set aims at confirming the correctness and verifying how the solution scales with respect to the future lookahead horizon length. We tested our solution on three models: `monkey`, encoding Example 1; `row` encoding a row of five cells where the agent can either move (left or right) or stay, and it collects some utility along the way; `grid` encoding a $3{\times}3$ grid world where the agent can move and collect utility. Results are reported in Table 1. Bold numbers represent the horizon length on which the agent starts to follow the optimal policy. We can see that there are still some scalability issues due in part to the size of the circuit which slows down the evaluation. However we plan to explore sampling to reduce the search space.

The second set of experiments aims at verifying how the solution scales with respect to the problem size, given a fixed horizon ($H = 1$). We created an $n{\times}n$ `grid`-like model, parametric on the size of the grid. The results are reported in Table 2. We can see that the compilation has scalability issues as well. This is an already known problem in literature, however, there are some techniques we will investigate that might alleviate it [Shen et al., 2018, Choi et al., 2022].

The experiments have been run on a machine with an Intel(R) Xeon(R) CPU E3-1225 v3 @ 3.20GHz and 24GB of memory.

6 CONCLUSIONS

This work introduced and verified that dynamic decision circuits are a useful tool to make dynamic decisions. There are still some scalability issues, in particular, using knowledge compilation only partially alleviate the intrinsic cost of dynamic decision problems. Therefore, as next step we will focus on introducing an approximation — MCTS-like [Browne et al., 2012] — to prune the search space and thus alleviate the cost for the future lookahead. In addition, we will investigate in more detail the connections between dynamic decision circuits and other circuit-based MDP solvers. The fact we represent the problem with a circuit let us exploit all the existing machinery already available for circuits in literature [Vergari et al., 2021].

Moreover, an important novelty is that we inherit — almost for free — the ability to learn utilities from Derkinderen and De Raedt [2020]. In fact, utilities do not change over time, therefore we will be able to adapt the learning procedure to the dynamic setting with just minor changes.

Finally, in the future we will also focus on extending the work to partially observable decision processes.

Acknowledgements

This work was partially supported by the KU Leuven Research Fund (C14/18/062), the Research Foundation-Flanders (FWO, 1SA5520N), the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme, the EU H2020 ICT48 project “TAILOR” under contract #952215, and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957. ISBN 978-0-691-14668-3.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012. ISSN 1943-0698. doi: 10.1109/TCIAIG.2012.2186810.

Yoojung Choi, Tal Friedman, and Guy Van Den Broeck. Solving Marginal MAP Exactly by Probabilistic Circuit Transformations. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, pages 10196–10208. PMLR, May 2022.

Adnan Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, May 2003. ISSN 0004-5411. doi: 10.1145/765568.765570.

Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17:229–264, September 2002. ISSN 1076-9757. doi: 10.1613/jair.989.

Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(2):142–150, 1989. ISSN 1467-8640. doi: 10.1111/j.1467-8640.1989.tb00324.x.

Vincent Derkinderen and Luc De Raedt. Algebraic Circuits for Decision Theoretic Inference and Learning. In *Proceedings of the 24th European Conference on Artificial Intelligence*, volume 325, pages 2569–2576. IOS Press, January 2020. ISBN 978-1-64368-100-9. doi: 10.3233/FAIA200392.

Zhengzhu Feng and Eric A. Hansen. Symbolic heuristic search for factored Markov decision processes. In *Eighteenth National Conference on Artificial Intelligence*, pages 455–460, USA, July 2002. American Association for Artificial Intelligence. ISBN 978-0-262-51129-2.

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 279–288, San Francisco, CA, USA, July 1999. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-614-2.

Ronald A. Howard and James E. Matheson. Influence diagrams. *Readings on Principles and Applications of Decision Analysis*, II(Menlo Park, CA: Strategic Decisions Group):721–762, 1984.

Keiji Kanazawa and Thomas Dean. A model for projection and action. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, pages 985–990, San Francisco, CA, USA, August 1989. Morgan Kaufmann Publishers Inc.

Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, July 2017. ISSN 1570-8683. doi: 10.1016/j.jal.2016.11.031.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 978-0-262-01319-2.

Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002.

Umut Oztok, Arthur Choi, and Adnan Darwiche. Solving PP PP -Complete Problems Using Knowledge Compilation. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, March 2016.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, September 1988. ISBN 978-1-55860-479-7.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, Hoboken, 4th edition edition, April 2020. ISBN 978-0-13-461099-3.

Yujia Shen, Arthur Choi, and Adnan Darwiche. Conditional PSDDs: Modeling and Learning With Modular Knowledge. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.12119.

Hari Tatavarti, Prashant Doshi, and Layton Hayes. Data-Driven Decision-Theoretic Planning using Recurrent Sum-Product-Max Networks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31:606–614, May 2021. ISSN 2334-0843.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In *Advances in Neural Information Processing Systems*, volume 34, pages 13189–13201. Curran Associates, Inc., 2021.

Jonas Vlasselaer, Wannes Meert, Guy Van den Broeck, and Luc De Raedt. Exploiting local and repeated structure in Dynamic Bayesian Networks. *Artificial Intelligence*, 232:43–53, March 2016. ISSN 0004-3702. doi: 10.1016/j.artint.2015.12.001.

A APPENDIX

In this appendix we provide some definitions which are not fundamental to understand the paper but might help to have a clearer picture.

A.1 WEIGHTED MODEL COUNTING

Weighted model counting (WMC) is an extension of the famous satisfiability (SAT) problem, where each literal has a weight assigned. In contrast to SAT, which checks if a theory admits a model, WMC requires computing the weighted sum of its models.

Definition 3 (Weighted model counting (WMC)) *Given a propositional logic theory T over a set of variables $\mathcal{V}$, and a weight function $w : \mathcal{L} \mapsto \mathbb{R}$, defined for all literals $\mathcal{L}$ in T. The task of weighted model counting is to compute:*

$$WMC(T, w) = \sum_{m \in \mathcal{M}(T)} \prod_{\ell \in m} w(\ell) \tag{5}$$

A.2 MAXIMUM EXPECTED UTILITY SEMIRING

Making decisions require to compute the maximum expected utility (MEU). In order to do so, with AMC, we will use the semiring S_{meu} [Derkinderen and De Raedt, 2020].

Definition 4 (S_{meu}) *The maximum expected utility semiring $S_{meu} = (\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ is defined as follows. Given the set of literals $\mathcal{L}$ of a propositional logic theory T, and $D \subset \mathcal{L}$ the set of literals representing decisions,*

$$\{\alpha(\ell) = (p_\ell, eu_\ell, \mathcal{D}_\ell) \mid \ell \in \mathcal{L}\} \subset \mathcal{A},$$

where p_ℓ is the probability of ℓ and eu_ℓ is its expected utility. Moreover, $\mathcal{D}_\ell = \{\ell\}$ if $\ell \in D$, or $\mathcal{D}_\ell = \emptyset$ otherwise. Given $a = (p, eu, \mathcal{D}) \in \mathcal{A}$ and $a' = (p', eu', \mathcal{D}') \in \mathcal{A}$,

$$\begin{aligned} a \oplus a' &= \begin{cases} max(a, a') & \text{if } \mathcal{D} \neq \mathcal{D}' \\ (p + p', eu + eu', \mathcal{D}) & \text{otherwise} \end{cases} \\ a \otimes a' &= (p \cdot p', p \cdot eu + p' \cdot eu', \mathcal{D} \cup \mathcal{D}') \\ e^{\oplus} &= (0, 0, D) \\ e^{\otimes} &= (1, 0, \emptyset) \end{aligned}$$

where,

$$max(a, a') = \begin{cases} a & \text{if } a' = e^{\oplus} \\ a' & \text{if } a = e^{\oplus} \\ a & \text{if } \frac{eu}{p} \geq \frac{eu'}{p'} \\ a' & \text{otherwise} \end{cases}$$

Notice that while the MEU task requires three operations (sum, product and max), a semiring is a structure with only two. Therefore, the $\oplus$ operation is dynamically defined: if the two operands have the same decision set $\mathcal{D}$, it means we just have to perform the summation; otherwise, it means there is a decision to be made, thus, we perform a maximisation.

A.3 LABELLING FOR THE MONKEY ENCODING

The following example provides an example of labelling, as described in Section 4.2, for the Example 1.

Example 3 (Labelling for the Monkey encoding) *To clarify the idea, we continue the Monkey example, providing the labelling for time step $t = 0$. At the starting point the model simplifies as described in Section 3.1, thus we do not need to label anything related to the previous time step.*

$$\begin{aligned} &\alpha_0(\lambda_m) = \alpha_0(\lambda_h) = \alpha_0(\lambda_{\neg m}) = \alpha_0(\lambda_{\neg h}) = (1, 0, \emptyset) \\ &\alpha_0(\theta_h) = \alpha_0(\theta_{\neg h}) = (0.5, 0, \emptyset) \\ &\alpha_0(\theta_m) = (1, 0, \{m\}) \qquad \alpha_0(\theta_{\neg m}) = (1, 0, \{\neg m\}) \\ &\alpha_0(\theta_{u|h,m}) = (1, -4 + AMC(S_{meu}, \Delta_{|h,m}, \alpha_1), \emptyset) \\ &\alpha_0(\theta_{u|h,\neg m}) = (1, -3 + AMC(S_{meu}, \Delta_{|h,\neg m}, \alpha_1), \emptyset) \\ &\alpha_0(\theta_{u|\neg h,m}) = (1, -1 + AMC(S_{meu}, \Delta_{|\neg h,m}, \alpha_1), \emptyset) \\ &\alpha_0(\theta_{u|\neg h,\neg m}) = (1, 4 + AMC(S_{meu}, \Delta_{|\neg h,\neg m}, \alpha_1), \emptyset) \end{aligned}$$

A.4 ONLINE PLANNING ALGORITHM

Online algorithms for planning alternate a *planning phase*, where the agent find the next best action[2], reasoning on the model of the environment, and an *execution phase*, where the best action found is executed in the environment and a new state is reached. They are generally more efficient than offline algorithms because instead of providing a(n optimal) policy — *i.e.*, a mapping from a state to the (best) action in that state — for every possible state, they explore only what is reachable from the starting point.

This is the pseudocode for the online planning algorithm using dynamic decision circuits to find the best next decision.

Algorithm 1 Online planning algorithm

```
 1: procedure OnlinePlanning($s, \Delta$)
 2:     $steps \leftarrow 0$
 3:     while $steps < MAX$ do
 4:         $a \leftarrow$ NextAction($s, \Delta$)
 5:         $s' \leftarrow$ Execute($s, a, \Delta$)
 6:         $s \leftarrow s'$
 7:         $steps \leftarrow steps + 1$
 8:     end while
 9: end procedure
10: procedure NextAction($s, \Delta$)
11:     $\_, \_, d \leftarrow$ AMC($S_{meu}, \Delta_{|s}, \alpha_0, H$)
12:     return $d$
13: end procedure
```

It takes as input the initial state s, and the circuit Δ. The number of steps the agent can perform in the environment

[2]In planning the term "action" is commonly used instead of "decision". We use them interchangeably.

is limited by a constant. We omit the pseudocode for the execution procedure because it is not relevant to our purpose. The hyper-parameter H limits the future lookahead horizon.