Case Study for Handling Hybrid Influence Diagrams in Probabilistic Programming Language Figaro with Discrete Algorithms

Junkyu Lee, Alexander Iher, and Rina Dechter
Donald Bren School of Information and Computer Sciences
University of California, Irvine

October 12, 2018

Executive Summary

This report is a part of the project called PRECOG with Charles River Analytics.

Probabilistic programming languages provide a rich modeling framework for defining and solving sequential decision-making problems under uncertainty. Figaro can define a decision model including continuous and discrete random variables and allow composition of hybrid mixtures of continuous and discrete random variables, offering flexible and realistic decision models that reflect complex real-world scenarios. However, such rich and powerful modeling capability brings significant technical challenges in design and implementation of algorithms for solving sequential decision problems.

This report defines the semantics of sequential decision problems in terms of the graphical model framework called influence diagrams, clarifies some of the technical challenges imposed by allowing arbitrary composition of continuous and discrete random variables in the model, a new approach called model sampling with online planning, and gives and initial assessment of the proposed methods. In addition, the possible future work and directions to extend the current approach are given in the conclusion.

The final outcomes accompanied with this report are a set of python scripts that prototype the proposed method (model sampling and online planning for hybrid influence diagrams), and our approximate discrete model inference algorithm for solving large scale influence diagrams (join graph decomposition bounds for influence diagrams), which are also described at the end of this report.

Background

Sequential Decision Making with Influence Diagrams

Sequential decision making under uncertainty is the problem of finding a sequence of decisions (or policies) along with the optimal total expected utility, in the presence of stochastic dynamical state transitions influenced by past decisions, and utility functions defined over the states and decisions.

An influence diagram (ID) [Howard and Matheson, 2005] is a graphical model for sequential decision-making under uncertainty that compactly captures the local structure of the conditional independence of probabilistic state transition functions and the additivity of utility functions.

Figure 1 shows a simple Figaro [Pfeffer, Figaro] decision model and the corresponding influence diagram. An influence diagram models a sequence of decision problems, each of which is comprised of random

variables and utility functions. The random variables can be categorized as decision random variables (drawn as squares) and observed or hidden random variables (drawn as a circles). In Figure 1, p1 is a hidden random variable (meaning it is not observed by any decision), and is associated with a conditional distribution depending on its parent random variable s that is connected by a directed edge from the node s to the node p1 ("Chain(Select(0.2→1, 0.5→10, 0.3→15), (i:Int) => Normal(I, 1))"), and o1 is an observed random variable associated with a discrete probability distribution ("Flip(0.7)"). Decision d1 is a decision variable whose value is under the control of a decision maker, and contingent on the observed value of o1 (connected by a directed arc from o1 to d1 called an informational arc). A non-forgetting agent makes decisions in multi-staged manner based on the history available at each decision, so that decision d2 is contingent on a set of observed random variables and decisions that includes those from the first stage, i.e., (o1, d1, o2). Typically, an influence diagram only shows informational arcs from the immediate (or new) observations to a decision. The utility functions u1 and u2 are drawn as diamonds in the influence diagram, and associated with their definition in the Figaro model. For more details about Figaro syntax, we refer to the Figaro Manuals and Tutorials.

The main tasks for solving a sequential decision problem under uncertainty are computing the optimal value of the expected utility and the optimal values or policy functions for the decision random variables. In Figure 1, the policy functions are d1: dom(o1) x dom(d1) → [0,1] and d2: dom(o1) x dom(d1) x dom(o2) x dom(d2) → [0, 1], where dom(x) denotes the domain of a variable x.

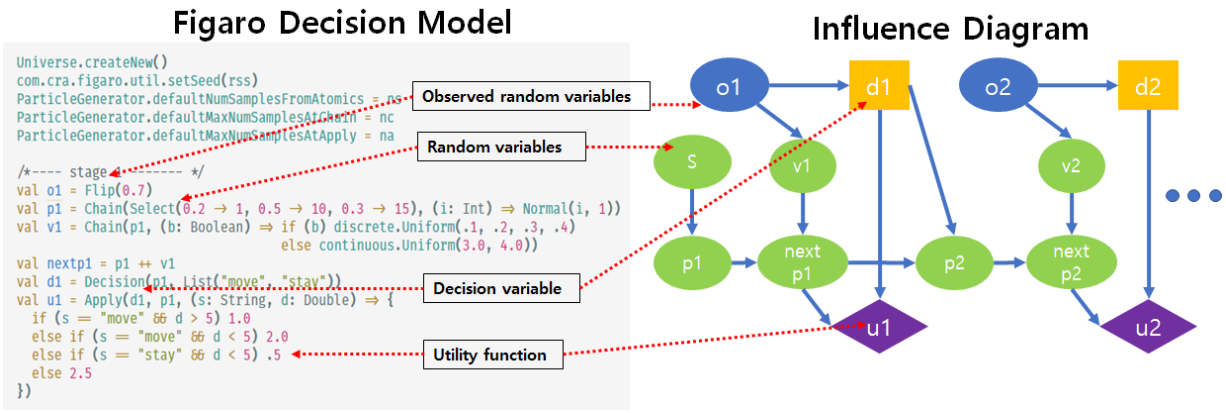


Figure 1. Figaro decision model and influence diagram

Hybrid Influence Diagrams and Technical Challenges

A hybrid influence diagram is an ID containing both discrete and continues random variables and decision variables. Figure 1 shows an example of a hybrid influence diagram. The hidden random variable v1 can be discrete or continuous, depending on the value of the observed random variable o1. If the value of o1 is true, then the conditional distribution Pr(v1| o1) is a discrete uniform distribution taking values from the discrete set {0.1, 0.2, 0.3, 0.4]. On the other hand, if the value of o1 is false, the conditional distribution Pr(v1| 01) is a continuous uniform distribution over the range (3.0, 4.0).

Continuous random variables help define more realistic probability models, but hybrids of continuous and discrete variables bring significant technical issues when designing and implementing algorithms. A hybrid influence diagram lacks a closed-form representation for the marginalization operations, i.e., the

summation and maximization for computing the expected utility and policy functions cannot be performed in a closed-form expression [Bielza, Gomez, and Shenoy 2011].

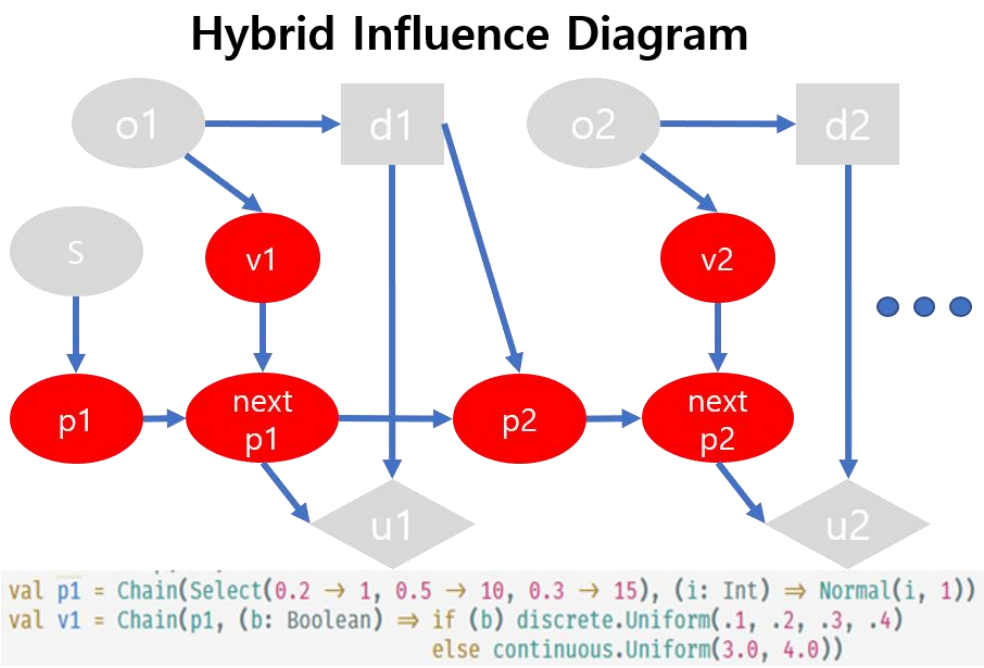


Figure 2. Hybrid influence diagram

Related work

Since general hybrid influence diagrams lack a closed-form expression for the computations, previous approaches restrict the original model to a tractable subclass of influence diagrams. Most of the methods are based on: 1) discretization of continuous distributions, 2) function approximation based on mixtures of Gaussians, truncated exponentials, or polynomials, 3) restricting the structure of the influence diagram to yield a closed-form expression, and 4) sampling the entire joint distribution [Li, and Shenoy 2012].

Model Sampling and Online Planning

In this report, a new approach is proposed for solving hybrid influence diagram based on model sampling and online planning. Model sampling refers to a process of generating a discrete influence diagram by sampling the domains of mixtures of continuous and discrete random variables. The advantage of generating discretized influence diagrams is that various existing inference algorithms can be applied to each discretized sampled model directly. However, it is important to note that the discretized model still lacks a closed-form representation for the policy functions if the observed random variables or decision random variables are continuous. Therefore, only the approximate value of the total expected utility and the best first action can be computed, giving an online planning framework.

Overall Process

Figure 3 shows the overall process of the proposed approach of model sampling and online-planning for solving hybrid influence diagrams. The overall process can be divided into three parts. The first part defines a Figaro model with a mix of continuous and discrete variables. The second part is an iterative process that alternates between generating discretized influence diagrams and solving each via discrete inference algorithms such as variable elimination until a desired number of iterations. The last part produces a final outcome, estimating the best action and the expected utility, by aggregating the results from the individual instances of discretized influence diagrams.

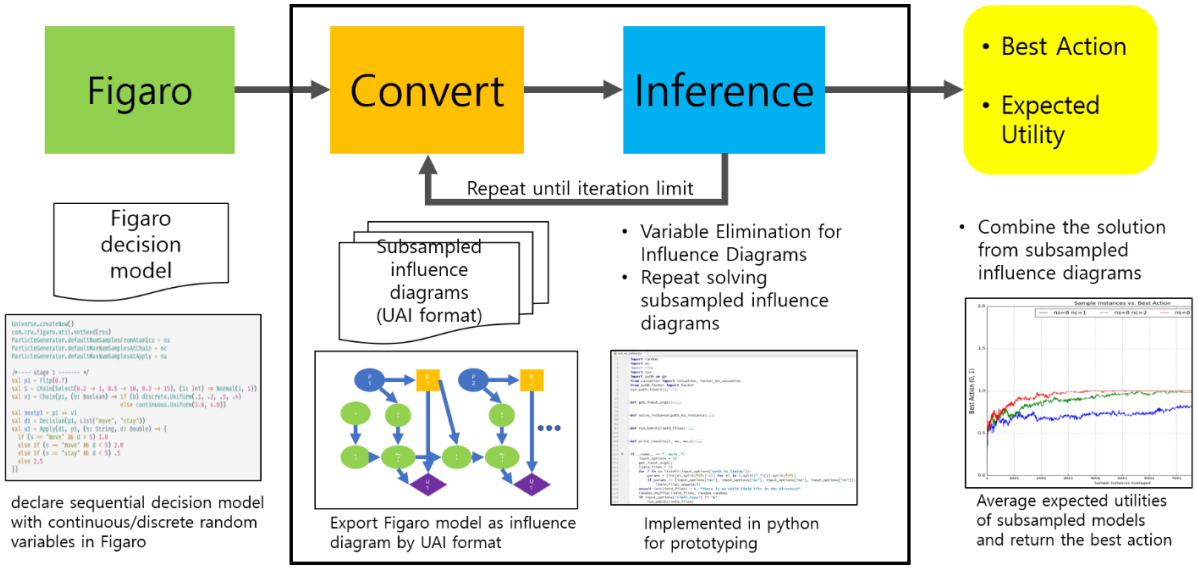


Figure 3. Overall process for solving hybrid influence diagram

Model Sampling and Online Planning

Figure 4 illustrates the model sampling and online planning process. Starting from the original hybrid influence diagram, we first generate a discrete distribution that samples Ns points from each variable by forward sampling in the influence diagram. Since the random variables in the influence diagram form a Bayesian network, the forward sampling can be performed by following the topological ordering of the nodes. The total size of the domains for a conditional probability distribution is O(Ns*(Ns^{pa}), where pa is the maximum number of parent nodes. We can see that the domain size grows exponentially over the number of time stages, O(Ns^{{pa}^T}), where T is the number of time stages.

Due to the exponential growth in the size of the discretized domains, the discretized influence diagram becomes intractable after a small number of time stages. To mitigate the explosion of the maximum domain size, we further subsample the domains of the conditional distributions using a parameter Nc. The subsampling can be done along with the discretization in the following manner. When generating a discretized domain by using the sampling parameter of Ns, if the domain size of the conditional distribution is greater than Nc then we pick only Nc domain values from the domains of the conditional distribution, then up-weight the probability by (Ns/Nc) to estimate the probability of the removed configurations. This subsampling method generates a discrete distribution whose sum (total mass) is an unbiased estimator of that for the original joint distribution.

The last step is to solve each subsampled discrete influence diagrams via some discrete inference algorithm, such as variable elimination [Dechter 1999] [Maua, de Campos, and Zaffalon 2012], and estimate the maximum expected utility value from the average of the conditional expected utility value conditioned on the first decision, denoted by $\tilde{E}[\sum u_i|d_0]$. The final estimates are the first decision value that maximizes the average of the conditional expected utility values from the subsampled models, and the estimated maximum expected utility value obtained by this best decision.

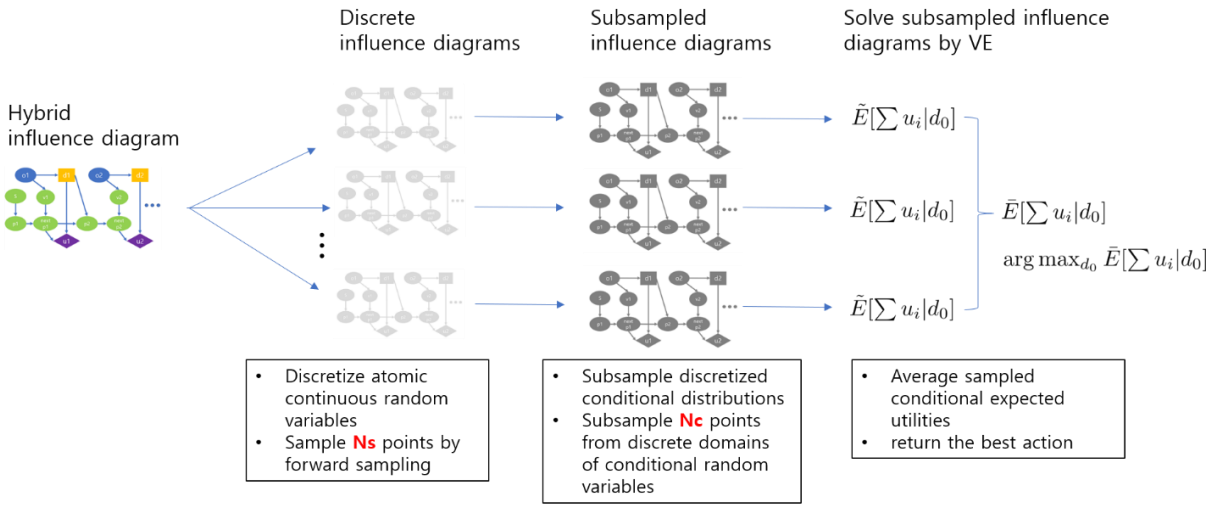


Figure 4. Model sampling and online planning

Complexity

The time and space complexity of solving an influence diagram by variable elimination is O(N*K^{w}), where N is the total number of random variables, K is the maximum domain size, and w is the induced width of the graphical model. As described in the previous section, the maximum domain size k grows super-exponentially in the number of time stages, hence the complexity of exact inference is also super-exponential in the number of time stages in the influence diagram, O(N*Ns^{w*{{pa}^T}}). In the presence of the two sampling parameters, one for the discretization of each continuous random variable, Ns, and the other for subsampling the discrete domains of any conditional distribution, Nc, the worst-case complexity is governed by Nc as O(N*Nc^{w}).

Figure 5 compares the complexity for solving a subsampled influence diagram using model sampling parameters Ns and Nc on a small 3 stage hybrid influence diagram model. The x-axis represents Nc and the y-axis represents the complexity for solving a sampled influence diagram as measured by the size of the pseudo tree. The size of the pseudo tree can be used to counts the number of operations required to perform variable elimination. We can see that the complexity of solving the subsampled influence diagram is exponential in w and Nc is the base of the exponent as O(N*Nc^{w}). compares the complexity for solving a subsampled influence diagram using model sampling parameters Ns and Nc on a small 3 stage hybrid influence diagram model. The x-axis represents Nc and the y-axis represents the complexity for solving a sampled influence diagram as measured by the size of the pseudo tree. The size of the pseudo tree can be used to counts the number of operations required to perform variable elimination. We can

see that the complexity of solving the subsampled influence diagram is exponential in w and Nc is the base of the exponent as O(N*Nc^{w}).

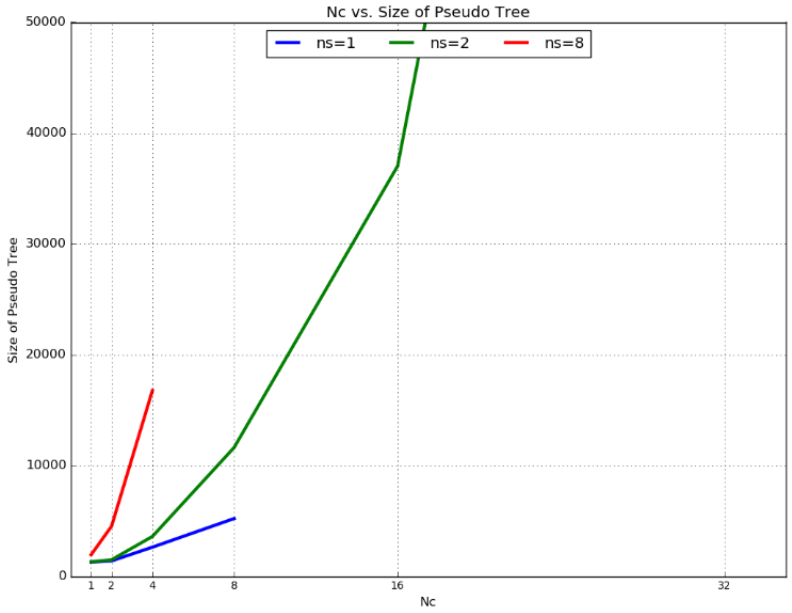


Figure 5. Complexity for solving subsampled influence diagrams by variable elimination algorithm

Bias in the estimate of MEU

We can view each discrete influence diagram instance as providing a sample of the conditional expected utility value conditioned on the first decision from the intractable hybrid influence diagram. Hence, we call the process model sampling, and we characterize the bias of model sampling in terms of the two sampling parameters Ns and Nc. Our model sampling and online planning approach produces a biased estimate of the maximum expected utility due to the stage-wise maximum operations applied individually to each discrete influence diagram, associated with each subsequent (non-first) decision.

Figure 6 shows the bias of the estimated maximum expected utility for a small 3 stage hybrid influence diagram model. The plot on the left-hand side shows the estimate of the maximum expected utility computed by various combinations of model sampling parameters, Ns=2 and Nc from 1 to 32. The x-axis represents the number of discrete influence diagram instances, and the y-axis represents the estimated maximum expected utility. We can see that the estimated maximum expected utility converges as more sample instances are drawn by the model sampling and online planning method. The figure on the right-hand side compares the changes in the converged estimate with varying Nc parameters. We can see that the estimate increases as Nc increased from 1 to the largest possible number (no subsampling).

In summary, the sampling parameter Ns over-estimates the maximum expected utility for lower values, and the subsampling parameter Nc under-estimates the maximum expected utility with lower values. These two bias effects combine, one causing over-estimation and the other under-estimation, to influence the value to which the estimated maximum expected utility converges.

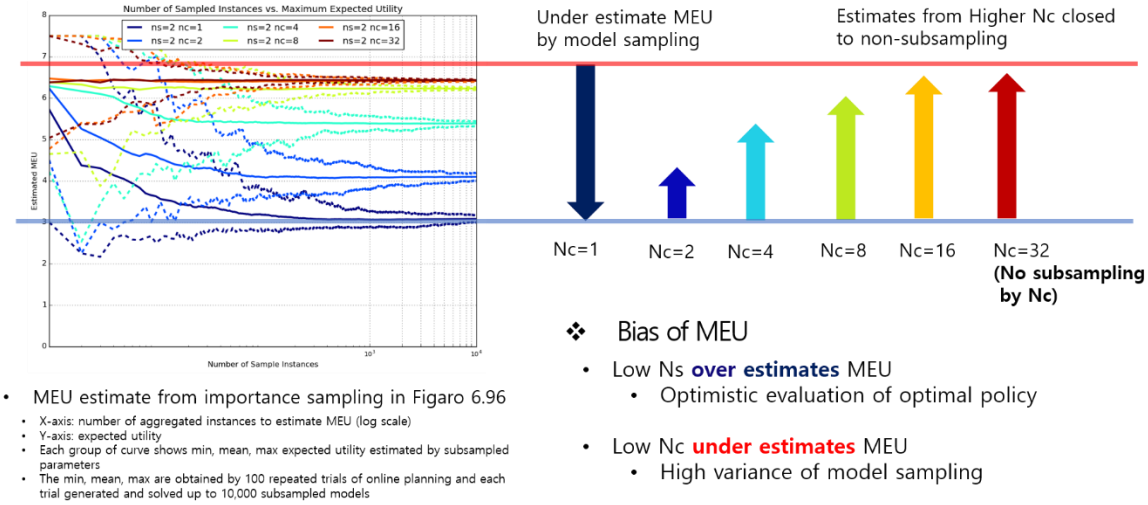


Figure 6. Bias of the maximum expected utility.

Experiments

In this section, we show the result of our experiments on the model sampling and online planning method using a 3-stage toy model. We focus on the bias behavior of the maximum expected utility of the model sampling, the sensitivity of the best first decision returned by online planning, and the computational running time complexity for solving the target model.

3 Stage Toy Model

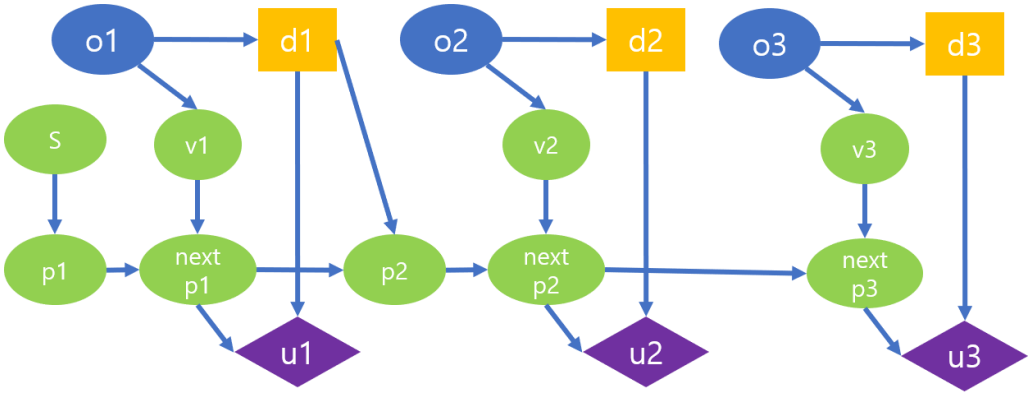


Figure 7. Influence diagram for the 3-stage toy model.

In the following experiments, we used the 3-stage toy model defined in Figure 7 and Figure 8. The influence diagram capturing the relevance of the variables and functions is also shown in Figure 7. The Figaro model that defines the probability distributions and utility functions is shown in Figure 8.

Each stage comprised of Figaro values (val) that declare observed or hidden random variables, decision variables, and utility functions. For example, the parent1 is the observed value to the first decision and the parent2 is a random variable observed immediately before the second decision. The position1 is a hidden random variable defined as a conditional distribution using Figaro Chain, and the nextPosition1 is a functional relation that adds two hidden random variables position1 and velocity1 using Figaro Apply. The declaration of a decision variable requires the list of past history as shown in the decision3 (^^(decision1, parent2, decision2, parent3)). In the 3-stage toy model, utility functions are defined by the Lambda expression

```
Universe.createNew()
com.cra.figaro.util.setSeed(rss)
ParticleGenerator.defaultNumSamplesFromAtomics = ns
ParticleGenerator.defaultMaxNumSamplesAtChain = nc
ParticleGenerator.defaultMaxNumSamplesAtApply = na

/*---- Decision 1 ------- */
val parent1 = false
val position1 = Chain(Select(0.2 -> 1, 0.5 -> 10, 0.3 -> 15), (i: Int) => Normal(i, 1))
val velocity1 = continuous.Uniform(3.0, 4.0)
//    val velocity = Chain(parent1, (b: Boolean) => if (b) discrete.Uniform(.1, .2, .3, .4) else continuous.Uniform(3.0, 4.0))
val nextPosition1 = position1 ++ velocity1
val decision1 = Decision(List("move", "stay"))
// val decision = Decision(parent, List("move", "stay"))
val utility1 = Apply(decision1, nextPosition1, (s: String, d: Double) => {
  if (s == "move" && d > 5) 1.0
  else if (s == "move" && d < 5) 2.0
  else if (s == "stay" && d < 5) .5
  else 2.5
})

/*---- Decision 2 depends on Decision 1 ------- */
val parent2 = Flip(0.7)
val position2 = If(decision1 === Constant("move"), nextPosition1 ++ Normal(0.0, 1.0), nextPosition1 ++ Normal(0.0, 0.3))
val velocity2 = Chain(parent2, (b: Boolean) => if (b) discrete.Uniform(.1, .2, .3, .4) else continuous.Uniform(3.0, 4.0))
val nextPosition2 = position2 ++ velocity2
val decision2 = Decision(^^(decision1, parent2), List("move", "stay"))
val utility2 = Apply(decision2, nextPosition2, (s: String, d: Double) => {
  if (s == "move" && d > 5) 1.0
  else if (s == "move" && d < 5) 2.0
  else if (s == "stay" && d < 5) .5
  else 2.5
})

/*---- Decision 3 depends on Decision 2 ------- */
val parent3 = Flip(0.7)
val position3 = If(decision2 === Constant("move"), nextPosition2 ++ Normal(0.0, 1.0), nextPosition2 ++ Normal(0.0, 0.3))
val velocity3 = Chain(parent3, (b: Boolean) => if (b) discrete.Uniform(.1, .2, .3, .4) else continuous.Uniform(3.0, 4.0))
val nextPosition3 = position3 ++ velocity3
val decision3 = Decision(^^(decision1, parent2, decision2, parent3), List("move", "stay"))
val utility3 = Apply(decision3, nextPosition3, (s: String, d: Double) => {
  if (s == "move" && d > 5) 1.0
  else if (s == "move" && d < 5) 2.0
  else if (s == "stay" && d < 5) .5
  else 2.5
})
```

Figure 8. Figaro 3-stage toy model.

Estimated MEU

Figure 9 shows the convergence behavior of the estimated MEU from various combinations of the model sampling parameters Ns and Nc. The data was generated as follows. Given a fixed combination of the parameters Ns and Nc, we randomly generated 100,000 discretized influence diagram instances off-line.

For each run of online-planning phase, we randomly selected discretized influence diagrams from the pool of sampled models from 1 to 10,000 and computed the estimate of MEU by using the randomly selected models. The same online-planning phase was repeated 100 times and we visualize the minimum, mean, and maximum of the estimates. The plot on the left-hand side shows the estimates of MEU from Ns=1 with Nc=1, 2, and 4, and the plot in the center shows the estimates of MEU from Ns=2 with Nc=1, 2, 4, 8, 16, and, 32, and the plot on the right-hand side shows the estimates of MEU from Ns=8 with Nc=1, 2, and 4. Note that the colors of each set curves for the minimum, mean, and maximum are associated with the Nc.

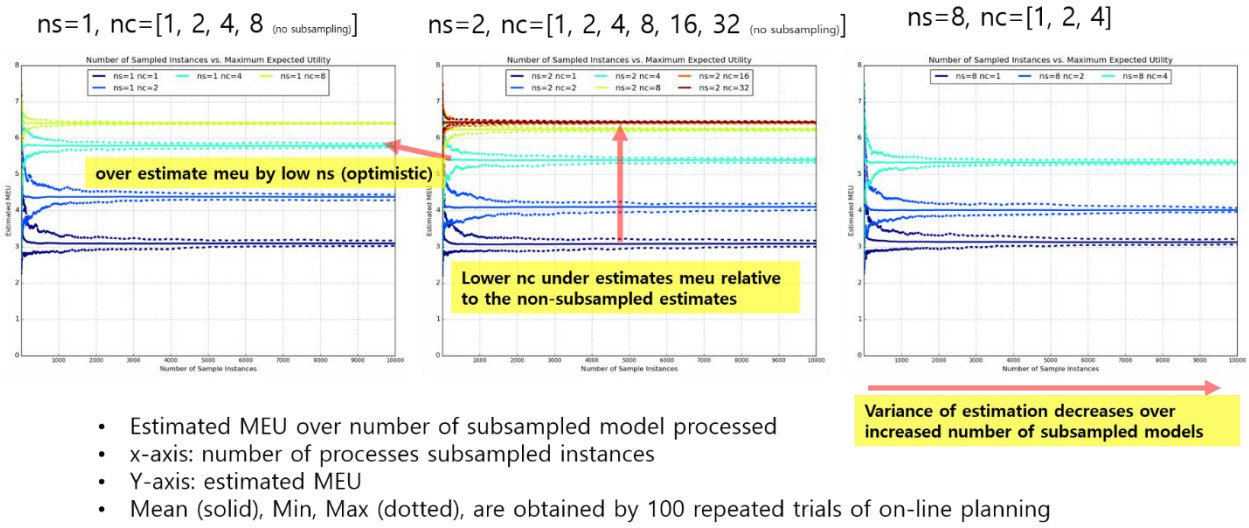


Figure 9. Estimated MEU.

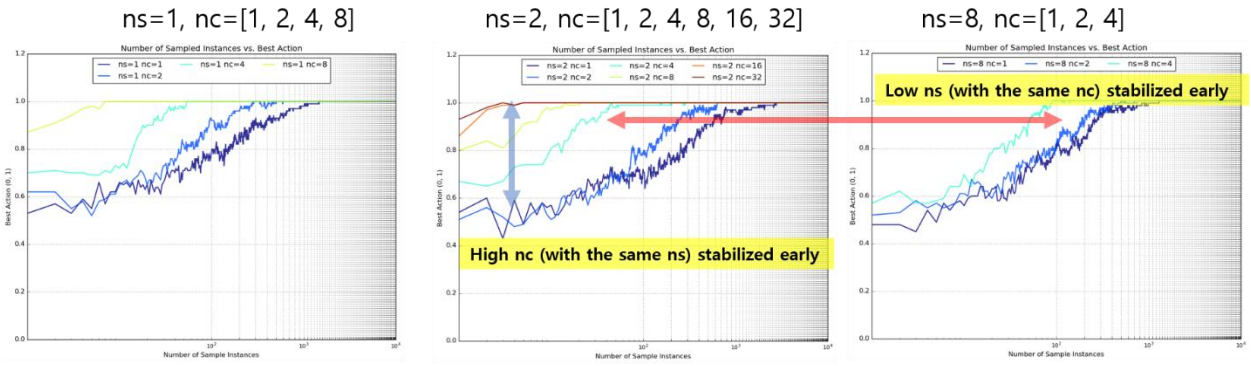
From all 3 plots, we can see that the variance of the estimates decreases over the increased number of sampled models. By comparing the curves in the same color across 3 different plots, we can observe that lower values of Ns produce higher estimates of the MEU. When the Nc was set sufficiently large number (no subsampling by Nc), **the Ns overestimates the MEU** because the online-planning phase evaluates the intermediate decisions in the opportunistic way that adapts to sampled trajectories.

By comparing the curves in the same plot, we can see that the lower values of Ns produce lower estimates of the MEU, i.e., **the Nc underestimates the MEU**. Since the Nc effectively rejects discretized trajectories produced by Ns by applying the weight of the subsampling (Ns/Nc) to the zero probability conditional trajectories, the subsampling method suffers from a very large bias that effectively underestimates the MEU in a finite number of model instances.

Best Action

Figure 10 visualizes the convergence behavior of the first decision returned by the online-planning phase. The model sampling parameters are the same as shown in Figure 9. The x-axis is the number of the sampled models that are aggregated to produce the estimate of the conditional expected utility conditioned on the first decision, and the y-axis is the value of the binary decision variable averaged over 100 repeated trials. For example, the value 0.5 for the best action implies that the online-planning phase returned two possible actions with the same number of occurrences within the repeated trials.

Comparing the curves in the same color across 3 different plots, the lower Ns shows faster convergence. On the other hand, the higher Nc shows faster convergence when we compare the curves within the same plot.

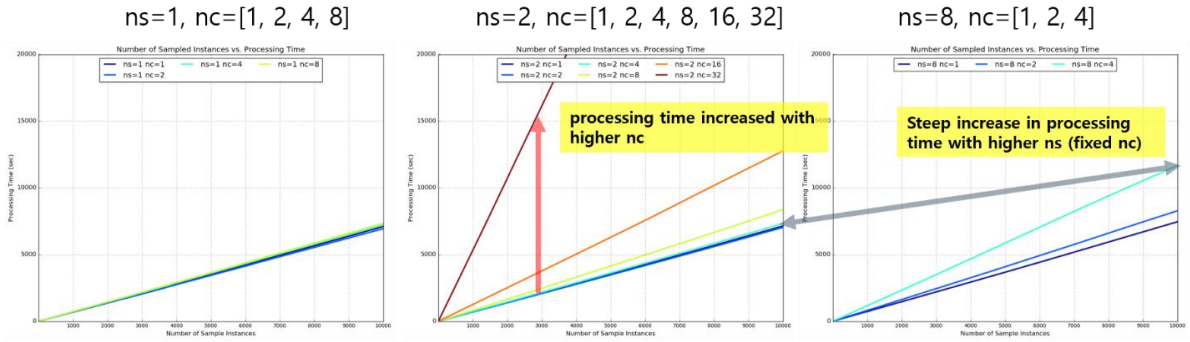


- Estimated MEU over number of subsampled model processed
- x-axis: number of processes subsampled instances (log-scale)
- Y-axis: average of binary actions (0, 1)
- Average actions are obtained by 100 repeated trials of on-line planning

Figure 10. The best action returned by online-planning.

Processing Time

Figure 11 shows the processing time for performing the online-planning with sampled models. The parameters are the same in the previous Figures, and the processing time on the y-axis was obtained by taking the average of 100 repeated trials.



- Estimated MEU over number of subsampled model processed
- x-axis: number of processes subsampled instances (log-scale)
- Y-axis: average time (seconds) to process instances
- Average times are obtained by 100 repeated trials of on-line planning

Figure 11. Processing time.

The experiment was conducted on the openlab UCI-ICS clusters with the following specifications: 1) Andromeda cluster comprised of 75 HP Proliant DL360 G5 servers equipped with Intel Xeon E5450 3 GHz and 32 GB RAM, 2) Archer cluster comprised of 21 generic servers equipped with AMD Opteron Processor 252 and 16 GB RAM, 3) Odin cluster with 1 Dell PowerEdge R815 server equipped with AMD Opteron

Processor 6378 and 512 GB RAM, and 4) Tristram cluster with 1 Dell PowerEdge R815 server equipped with AMD Opteron Processor 6378 and 512 GB RAM. The openlab clusters are operated by Sun Grid Engine with each job queue assigned 8 GB memory.

We can see that the processing times Ns=1 remains similar with different Nc parameters. However, as Nc grows to 2 and 8, the processing time increases rapidly due to the complexity for solving each sampled model by the variable elimination algorithm.

Join Graph Decomposition Bounds for Influence Diagrams

In this report, the variable elimination algorithm was used for solving discretized influence diagram instances generated by the model sampling process. Since space and time complexity of the variable elimination algorithm is exponential in the graph parameter called the induced width, the variable elimination algorithm is intractable for solving influence diagram with a large induced width.

We developed an approximate inference scheme, join graph decomposition bounds for influence diagrams (JGDID), that provides upper bounds of the maximum expected utility of a discrete influence diagram. The JGDID extends dual decomposition for marginal MAP inference based on the valuation algebra for solving influence diagrams and join graph decomposition of influence diagrams. JGDID is an iterative message passing algorithm that decomposes the input influence diagram as a lower complexity join graph and tightens the upper bound by reparameterization of probability and utility functions relative to the join graph structure.

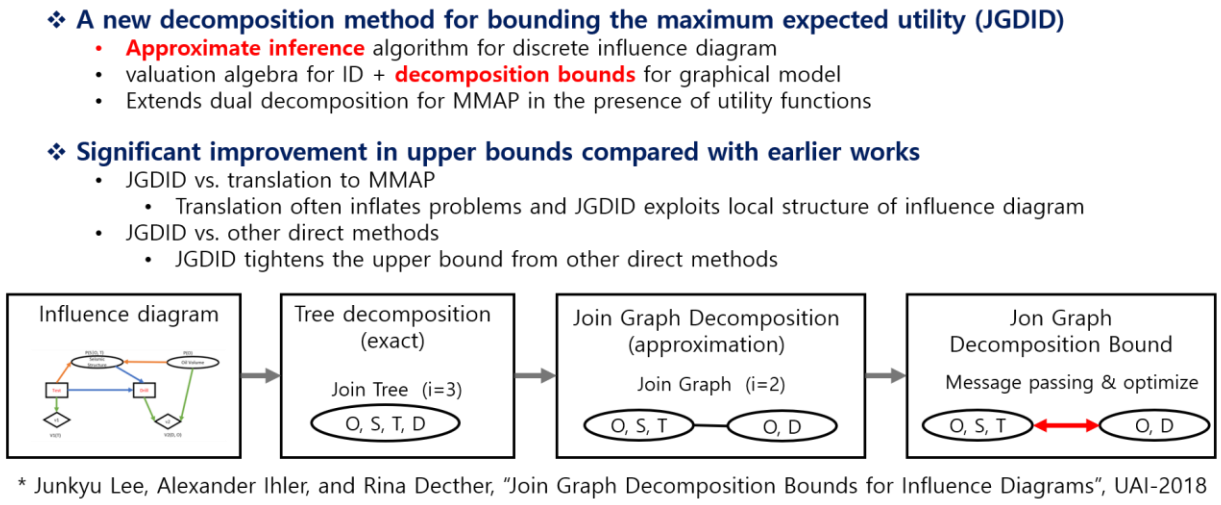


Figure 12. Join graph decomposition bounds for influence diagrams.

Figure 13 demonstrates the performance of JGDID compared with the existing approaches for solving discrete influence diagrams. The previous approaches can be divided into two groups. The first one is the translation-based approach that translates influence diagram as a Bayesian network and solves the Bayesian network with marginal MAP inference algorithms. The second approach is based on the direct relaxation of the input influence diagrams by mini-bucket elimination or information relaxation. The plots

on the righthand side compares the upper bounds from various approaches and we can see that JGDID produced the best upper bounds for all cases. Note that, the proposed JGDID uses mini-bucket elimination as its initialization step and it can be also combined with the information relaxation methods to yield superior upper bounds. For the technical details, we refer to [Lee, Ihler, and Dechter 2018].

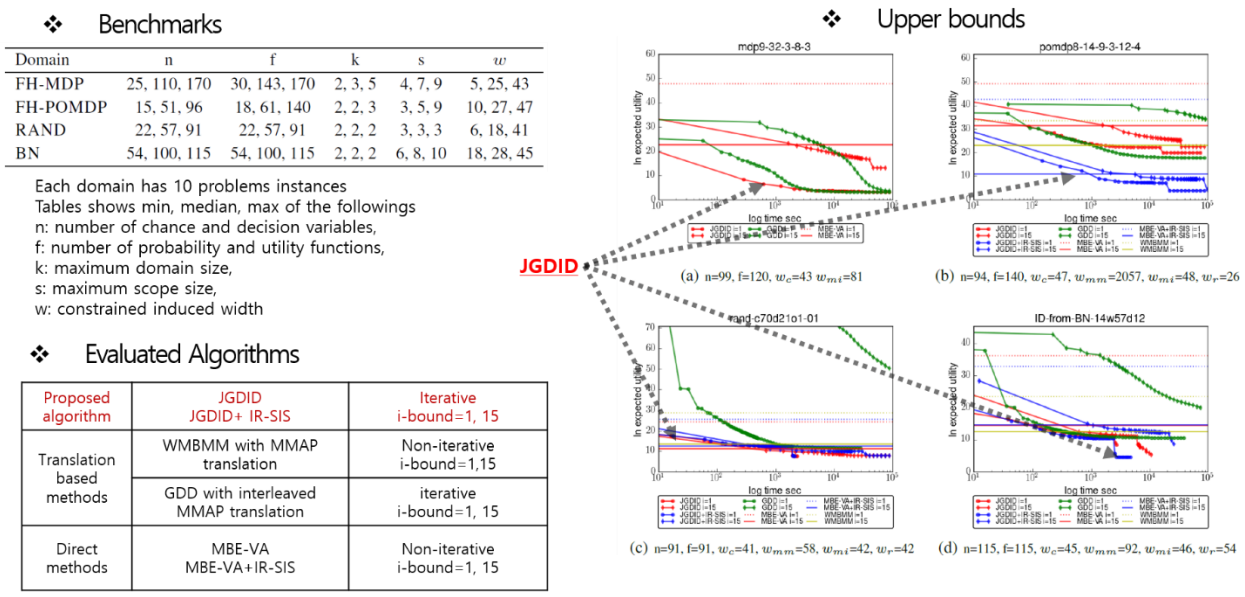


Figure 13. Demonstration of join graph decomposition bounds for influence diagrams.

Python Scripts for Demonstration

We provide python scripts that demonstrate the model sampling and online-planning methods. The script is composed of a python library pyGM that provides implementations of graphical model inference algorithms including the variable elimination, and a single python script run_ve_online.py, which reads sampled models generated by Figaro UAI exporter.

The run_ve_online.py requires the following arguments in sequence.

```
$Python3 run_ve_online.py <path to model directory> <Ns> <Nc> <Na> <Nt> <b or t> <max limit> <verbose>
```

The <path to model directory> is the path to the directory storing the model instances generated by UAI exporter, <Ns> is the model sampling parameter Ns, <Nc> is the model sampling parameter Nc, <Na> is another model sampling parameter similar to Nc but applies to Figaro APPLY with default value 1000000, <Nt> is the number of time stages, <b or t> defines the mode of online planning b for solving fixed number of batches and t for solving model instances within fixed time limit, <max limit> is the maximum number of model instances or time in seconds depending on the previous option, and the final optional argument <verbose> shows the intermediate results when it is provided.

Figure 14 demonstrates the model sampling and online planning with the python scripts. In the example, the provided arguments are: <models/Toy3> <2> <2> <1000000> <3> <b> <2> <v>, which solves 3 stage toy model with model sampling parameters Ns=2 and Nc=2 up to 2 model instances.

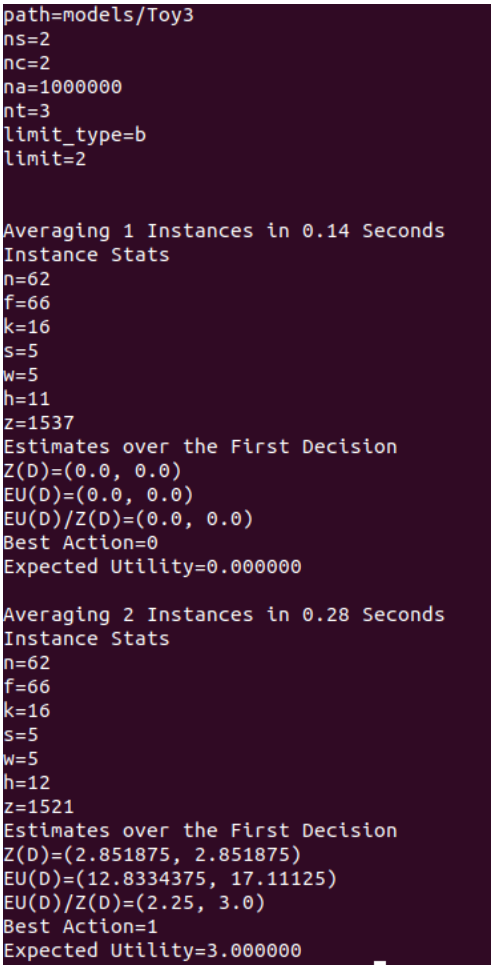


Figure 14. Demonstration of model sampling and online planning.

Conclusion

In this report, we described a new method, model sampling and online planning for solving hybrid influence diagrams, for solving hybrid influence diagrams that allow the arbitrary mixture of continuous and discrete random variables. The method is iterative in the sense that it generates discretized influence diagrams by sampling the hybrid mixture of continuous and discrete random variables and it is online in the sense that the best action is returned by solving sampled models within a time bound.

Allowing hybrid mixture of continuous and discrete random variables brings significant challenges in the knowledge representation and inference due to the absence of the symbolic form of the intermediate solution process such as marginalization of random variables or maximization of decision variables. The proposed method overcomes the representational issue by 1) solving samples of the discretized model by inference algorithms for discrete graphical models, and 2) aggregating the conditional expected utilities from sampled influence diagrams to approximate the true maximum expected utility. However, the forward sampling process with the parameter Ns in the model sampling phase creates discrete models

with very large domain size, and the subsampling process with the parameter Nc and Na introduces bias on the estimate of the maximum expected utility.

Finally, we point out possible future directions to resolve the technical issues. From the representation viewpoint, large and dense tables that encode functional relations are problematic in practical situations, especially in the presence of large scope size relations or large domain size variables. The changes in the representation require modifications to the existing inference algorithms. It is desirable to devise approximation schemes that approximate the intractable hybrid mixture of continuous and discrete distributions to a tracible class of distribution since the forward sampling in the model sampling phase creates intractable discrete models in practice. Most approximate inference algorithms for solving discrete graphical models are designed to mitigate large induced width of the model. New approximation algorithms are desired to improve the time and space complexity in the presence of the random variables with a very large domain size.

References

[Howard and Matheson, 2005] Influence diagrams. Decision Analysis, 2(3):127-143.

[Pfeffer, 2009] An object-oriented probabilistic programming language, Charles River Analytics Technical Report.

[Bielza, Gomez, and Shenoy 2011] A review of representation issues and modeling challenges with influence diagrams. Omega, 39(3), pp.227-241.

[Li, and Shenoy 2012] A framework for solving hybrid influence diagrams containing deterministic conditional distributions, Decision Analysis, 9(1):55-75.

[Decther 1999] Bucket elimination: A unifying framework for reasoning. Artificial Intelligence, 113(1):41–85.

[Mauá, de Campos, and Zaffalon 2012] Solving limited memory influence diagrams. Journal of Artificial Intelligence Research, 44:97–140.

[Lee, Ihler, and Dechter 2018] Join graph decomposition bounds for influence diagrams, Uncertainty in Artificial Intelligence 2018.