Scaling Integer Arithmetic in Probabilistic Programs (Supplementary Material)

William X. Cao[1] **Poorva Garg**[*1] **Ryan Tjoa**[*2] **Steven Holtzen**[3] **Todd Millstein**[1] **Guy Van den Broeck**[1]

[1]Department of Computer Science, University of California, Los Angeles, California, USA
[2] Department of Computer Science, University of Washington, Seattle, Washington, USA
[3]Department of Computer Science, Northeastern University, Boston, Massachussetts, USA

A EXPERIMENTAL DETAILS

All experiments were run on a server with a 2.20GHz CPU and 504GB RAM. WebPPL experiments were run on WebPPL v0.9.15; Psi experiments were run on Psi version `ec2cfc14a62a168afe7ce1d7269b92cf2882b830`. `Dice.jl` experiments were run using the code available at `https://github.com/Juice-jl/Dice.jl/tree/arithmetic`. The implementations of each model run are available at the same repository.

B BENCHMARK MODELS

We provide a small description of our benchmarks in this section with details of their sources. The code implementations of all models are available at the repository `https://github.com/Juice-jl/Dice.jl/tree/arithmetic`.

1. book: A model of flipping towards a target page in a book adapted from the Psi test directory [Gehr et al., 2022].
2. tugofwar: Adapted from a traditional tug-of-war example [Huang et al., 2021], with values made discrete.
3. caesar: The caesar-cipher example from Dice [Holtzen et al., 2020]. with different number of characters being observed.
4. ranking: A model for learning a ranking system, adapted from Kisa et al. [2014].
5. radar1: A model of radar reception, adapted from a continuous model from Psi's [Gehr et al., 2016] benchmark suite.
6. floydwarshall: An implementation of the Floyd-Warshall algorithm [Floyd, 1962] on a graph with edges of random weight.
7. linear-extensions: A model counting linear extensions [Dittmer and Pak, 2018] where we observe a partial order and get an output distribution over all matching total orders.
8. triangle: A model categorizing a triangle of random side lengths, adapted from the Psi test directory [Gehr et al., 2022].
9. gcd: A model checking if two random numbers are coprime implementing Euclid's algorithm [Lehmer, 1938].
10. disease: A discrete disease model taken from existing works [Laurel and Misailovic, 2020].
11. luhn: A probabilistic model of student IDs leveraging the Luhn algorithm [Luhn, 1960].

C ADDITIONAL EXPERIMENTAL RESULTS

We provide additional experimental results supplementing those in the main paper.

BDD size serves as a proxy for how well knowledge compilation can exploit structure: the more compact the BDD, the smaller the representation of our function, and the faster weighted model counting can be executed. Figure 1 compares the resultant BDD size for various models when compiled in Dice.jl using a binary or one-hot encoding. We can see that in almost all cases the binary encoding results in a smaller BDD, in some cases much smaller. Note that these models are those for which the one-hot encoding did not timeout; it is likely for those models that timed out that the true compiled BDD size will end up being much larger than the binary encoded BDD size.

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

Table 1: BDD sizes for probabilistic models using a binary vs one-hot encoding

Benchmarks	binary	one-hot
tugofwar	**2400**	2821
caesar-small	**1304**	3879
caesar-medium	**6344**	17879
caesar-large	**12644**	35379
ranking-small	**691**	1146
ranking-medium	**6218**	8297
ranking-large	**11491**	15680
radar1	**181**	332
floydwarshall-small	**10**	**10**
floydwarshall-medium	341	**237**
linear extensions-small	**29**	53
linear extensions-medium	**133**	257
linear extensions-large	**997**	1538
triangle-small	**10273**	150089
triangle-medium	**40419**	1156785
luhn-small	**518**	1010
luhn-medium	**2899**	7361

D BDD SIZES OF INTEGER DISTRIBUTION ENCODINGS (PROPOSITION 1)

D.1 NOTATION AND DEFINITIONS

A b-rooted BDD B with m decision variables computes some function $\{0,1\}^m \to \{0,1\}^b$ at its roots.

Let the values of the roots of B, given truth assignments to decision variables $\vec{x}$, be denoted $B_1(\vec{x}), \ldots, B_b(\vec{x})$. Let the variable order of B match the order of the components of $\vec{x}$. A BDD can thus be specified by a tuple of functions $(\{0,1\}^m \to \{0,1\})^b$. Let $B(\vec{x})$ denote this tuple, i.e. $(B_1(\vec{x}), \ldots, B_b(\vec{x}))$.

As b-length bitvectors are isomorphic to b-bit unsigned machine integers, we can also specify a BDD by a function from assignments to b-bit unsigned integers. Given $f : \{0,1\}^m \to \{0, \ldots, 2^b - 1\}$, let $[\![f]\!]$ denote the BDD defined such that $[\![f]\!]_i(\vec{x}) = \mathrm{LSB}_i(f(\vec{x}))$, where $\mathrm{LSB}_i(j)$ denotes the ith least significant bit of j.

Let $\vec{p} \in [0,1]^m$ assign a probability to each decision variable. For each p_i, let X_i denote an independent $\mathrm{Bernoulli}(p_i)$. We can construct a random variable for the result of B with $B(\vec{X})$ (passing in X_i for each x_i). Note the indexing of the decision variable x_i, the corresponding probability p_i, and the corresponding random variable X_i may vary to make notation clearer: for CATEG_INT, these will be zero-indexed; for BITWISE_INT, these will be indexed by bit strings.

Let $\mathcal{D}_b$ denote a distribution over b-length bit vectors. Let $v^\star$ be an arbitrary random variable sampled from $\mathcal{D}_b$, interpreted as an unsigned integer. Denote the bits of $v^\star$, from least to most significant, as $v^\star_1, \ldots, v^\star_b$.

Definition 1 (Integer distribution encoding)**.** A BDD and probability assignments $(B, \vec{p})$ *encode* a distribution $\mathcal{D}_b$ if $B(\vec{X}) \sim \mathcal{D}_b$.

D.2 CATEGORICAL ENCODING OF INTEGER DISTRIBUTIONS

Definition 2. Define $\text{encode}_{\text{CATEG}}(\mathcal{D}_b) = (\llbracket g \rrbracket, \vec{p})$ such that $p_i = \Pr(v^\star = i | v^\star \geq i)$, $0 \leq i \leq 2^b - 2$, and that g is defined as follows.

$$g(\vec{x}) = \begin{cases} 0 & x_0 \\ 1 & \neg x_0 \wedge x_1 \\ 2 & \neg x_0 \wedge \neg x_1 \wedge x_2 \\ \cdots & \\ j & x_j \wedge \bigwedge_{i=0}^{j-1} \neg x_i \text{ and } 0 \leq j \leq 2^b - 2 \\ \cdots & \\ 2^b - 1 & \bigwedge_{i=0}^{2^b-2} \neg x_i \end{cases}$$

We could also write this as the following, which is also how it would likely be implemented in a probabilistic programming language.

$$g(\vec{x}) = \begin{cases} 0 & \text{if } x_0 \\ 1 & \text{else if } x_1 \\ 2 & \text{else if } x_2 \\ \cdots & \\ 2^b - 2 & \text{else if } x_{2^b-2} \\ 2^b - 1 & \text{else} \end{cases}$$

Lemma 1. *If* $\text{Img}(v) \subseteq \mathbb{N}_0$*, then* $\prod_{i=0}^{x-1} \Pr(v \neq i | v \geq i) = \Pr(v \geq x)$*. Proof omitted; proceed by induction.*

Lemma 2. *For any distribution* $\mathcal{D}_b$ *over unsigned integers,* $\text{encode}_{\text{CATEG}}(\mathcal{D}_b)$ *encodes* $\mathcal{D}_b$*.*

Proof. Let $\text{encode}_{\text{CATEG}}(\mathcal{D}_b) = (B, \vec{p})$. We prove that for all $0 \leq j \leq 2^b - 1$, $\Pr(g(\vec{X}) = j) = \Pr(v^\star = j)$.

Case 1: $0 \leq j \leq 2^b - 2$.

$$\begin{aligned} \Pr(g(\vec{X}) = j) &= \Pr\left(X_j \wedge \bigwedge_{i=0}^{j-1} \neg X_i\right) \\ &= \Pr(X_j) \prod_{i=0}^{j-1} \Pr(\neg X_i) && \text{(Independence)} \\ &= \Pr(v^\star = j | v^\star \geq j) \prod_{i=0}^{j-1} \Pr(v^\star \neq i \mid v^\star \geq i) && \text{(As } \Pr(X_i) = p_i\text{)} \\ &= \Pr(v^\star = j | v^\star \geq j) \Pr(v^\star \geq j) && \text{(Lemma 1)} \\ &= \Pr(v^\star = j) \end{aligned}$$

Case 2: $j = 2^b - 1$.

$$\begin{aligned}
\Pr(g(\vec{X}) = 2^b - 1) &= \Pr\left(\bigwedge_{i=0}^{2^b-2} \neg X_i\right) \\
&= \prod_{i=0}^{2^b-2} \Pr(\neg X_i) && \text{(Independence)} \\
&= \prod_{i=0}^{2^b-2} \Pr(v^\star \neq i | v^\star \geq i) && (\text{As } \Pr(X_i) = p_i) \\
&= \Pr(v^\star \geq 2^b - 1) && \text{(Lemma 1)} \\
&= \Pr(v^\star = 2^b - 1)
\end{aligned}$$

□

D.3 BDD SIZE OF THE CATEG_INT ENCODING

We prove the first half of Proposition 1.

Proposition 1 (first half). A discrete distribution over the integers $\{0, 1 \ldots, 2^b - 1\}$ compiles to a BDD of size $\Theta(b2^b)$ when represented using CATEG_INT (Algorithm 1).

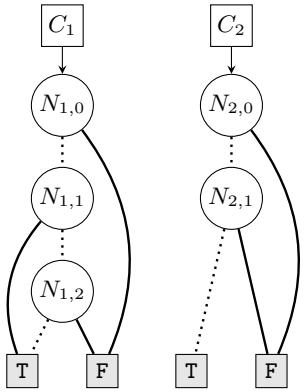
Lemma 3. *Let* $\text{encode}_{\text{CATEG}}(\mathcal{D}_b) = (\llbracket g \rrbracket, P)$. $\llbracket g \rrbracket$ *exactly matches a BDD* C *with the following reduced structure.*

For $1 \leq i \leq b$, C *has nodes* $N_{i,0}, N_{i,1}, \ldots, N_{i,2^b-2^{i-1}-1}$, *at the levels of decision variables* $x_0, x_1, \ldots, x_{2^b-2^{i-1}-1}$, *respectively.*

$$\begin{aligned}
\text{low}(N_{i,j}) &= \textit{if} \quad j < 2^b - 2^{i-1} - 1 \quad \textit{then} \quad N_{i,j+1} \quad \textit{else} \quad 1 \\
\text{high}(N_{i,j}) &= \text{LSB}_i(j)
\end{aligned}$$

Let the roots of C *be placed such that* $C_i(\vec{x})$ *matches* $N_{i,0}$ *given assignments to decision variables* $\vec{x}$.

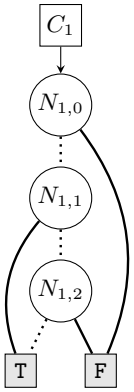
As an example, the BDD for $b = 2$ follows (terminal nodes visually duplicated for clarity).



Proof. We will show for all $\vec{x}$, $C_i(\vec{x}) = \text{LSB}_i(g(\vec{x}))$.

Let $S_{i,j}(\vec{x})$ denote the state of C_i at the level of x_j, given an assignment to $\vec{x}$. This is either a node at level x_j, or possibly one at a lower level if there was a low/high edge that skipped levels. See the following BDD and corresponding states as an example.

$$
\begin{aligned}
S_{1,0}(x_0, x_1, x_2) &= N_{1,1} \\
S_{1,1}(0, x_1, x_2) &= N_{1,2} \\
S_{1,1}(1, x_1, x_2) &= 0 \\
S_{1,2}(0, 0, x_2) &= N_{1,2} \\
S_{1,2}(0, 1, x_2) &= 1 \\
S_{1,2}(1, x_1, x_2) &= 0
\end{aligned}
$$



Let $P(t)$ be the statement, $S_{i,t} = \text{if} \quad g(\vec{x}) < t \vee t > 2^b - 2^{i-1} - 1 \quad \text{then} \quad \text{LSB}_i(g(\vec{x})) \quad \text{else} \quad N_{i,t}$.

$P(1)$ holds as the initial node of all $C_i(\vec{x})$ is always $N_{i,0}$.

Assume $P(t)$ for an arbitrary $t \leq 2^b - 1$.

Consider going to the next node. If we are at a terminal node, then the state stays the same, otherwise we go to the low or high edge based on x_t. Thus, to advance state, we replace $N_{i,t}$ with $\text{if} \quad x_t \quad \text{then} \quad \text{high}(N_{i,t}) \quad \text{else} \quad \text{low}(N_{i,t})$.

$$
\begin{aligned}
S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t \vee t > 2^b - 2^{i-1} - 1 \\
\text{then} \quad & \text{LSB}_i(g(\vec{x})) \\
\text{else} \quad & (\text{if} \quad x_t \quad \text{then} \quad \text{high}(N_{i,t}) \quad \text{else} \quad \text{low}(N_{i,t}))
\end{aligned}
$$

We substitute for $\text{high}(N_{i,t})$ and $\text{low}(N_{i,t})$.

$$
\begin{aligned}
S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t \vee t > 2^b - 2^{i-1} - 1 \\
\text{then} \quad & \text{LSB}_i(g(\vec{x})) \\
\text{else} \quad & \big(\text{if} \quad x_t \quad \text{then} \quad \text{LSB}_i(t) \quad \text{else} \quad \big(\text{if} \quad t < 2^b - 2^{i-1} - 1 \quad \text{then} \quad N_{i,t+1} \quad \text{else} \quad 1\big)\big)
\end{aligned}
$$

$x_t \wedge \neg(g(\vec{x}) < t)$ implies $g(\vec{x}) = t$.

$$
\begin{aligned}
S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t \vee t > 2^b - 2^{i-1} - 1 \\
\text{then} \quad & \text{LSB}_i(g(\vec{x})) \\
\text{else} \quad & \big(\text{if} \quad x_t \quad \text{then} \quad \text{LSB}_i(g(\vec{x})) \quad \text{else} \quad \big(\text{if} \quad t < 2^b - 2^{i-1} - 1 \quad \text{then} \quad N_{i,t+1} \quad \text{else} \quad 1\big)\big)
\end{aligned}
$$

Two branches are now equivalent ($\text{LSB}_i(g(\vec{x}))$) and can be combined.

$$
\begin{aligned}
S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t \vee t > 2^b - 2^{i-1} - 1 \vee x_t \\
\text{then} \quad & \text{LSB}_i(g(\vec{x})) \\
\text{else} \quad & \big(\text{if} \quad t < 2^b - 2^{i-1} - 1 \quad \text{then} \quad N_{i,t+1} \quad \text{else} \quad 1\big)
\end{aligned}
$$

$g(\vec{x}) < t \vee x_t$ is equivalent to $g(\vec{x}) < t + 1$.

$$
\begin{aligned}
S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t + 1 \vee t > 2^b - 2^{i-1} - 1 \\
\text{then} \quad & \text{LSB}_i(g(\vec{x})) \\
\text{else} \quad & \big(\text{if} \quad t < 2^b - 2^{i-1} - 1 \quad \text{then} \quad N_{i,t+1} \quad \text{else} \quad 1\big)
\end{aligned}
$$

In the outer else, $t \leq 2^b - 2^{i-1} - 1$ (by the condition). In the inner else, t is also $\geq 2^b - 2^{i-1} - 1$.

$$
\begin{aligned}
S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t + 1 \vee t > 2^b - 2^{i-1} - 1 \\
\text{then} \quad & \text{LSB}_i(g(\vec{x})) \\
\text{else} \quad & \big(\text{if} \quad t = 2^b - 2^{i-1} - 1 \quad \text{then} \quad N_{i,t+1} \quad \text{else} \quad 1\big)
\end{aligned}
$$

In the inner else, $g(\vec{x}) > t$ by the outer condition and $t = 2^b - 2^{i-1} - 1$ by the inner condition; together, $g(\vec{x}) > 2^b - 2^{i-1} - 1$. For all such $g(\vec{x})$, $\mathrm{LSB}_i(g(\vec{x})) = 1$ (as $2^b - 1 - 2^{i-1}$) is the largest number at most $2^b - 1$ with the i^{th} bit set to 0). Thus, we can merge two more branches.

$$\begin{aligned} S_{i,t+1} = \text{if} \quad & g(\vec{x}) < t+1 \vee t+1 > 2^b - 2^{i-1} - 1 \\ \text{then} \quad & \mathrm{LSB}_i(g(\vec{x})) \\ \text{else} \quad & N_{i,t+1} \end{aligned}$$

Thus $P(t) \to P(t+1)$ and thus $P(t)$ holds for all $1 \leq t \leq 2^b - 1$.

Consider $P(2^b - 1)$.

$$\begin{aligned} &S_{i,2^b-1} = \text{if} \quad g(\vec{x}) < 2^b - 1 \vee 2^b - 1 > 2^b - 2^{i-1} - 1 \quad \text{then} \quad \mathrm{LSB}_i(g(\vec{x})) \quad \text{else} \quad N_{i,2^b} \\ &S_{i,2^b-1} = \mathrm{LSB}_i(g(\vec{x})) \end{aligned}$$

Therefore $[\![g]\!]$ and C are equivalent. □

Proposition 1 (first half).

Proof. Let $\mathrm{encode}_{\mathrm{CATEG}}(\mathcal{D}_b) = (B, \vec{p})$. B has $b2^b - 2^b + 1$ decision nodes.

It follows directly from Lemma 3 that $\sum_{i=1}^{n} 2^b - 2^{i-1} = b2^b - 2^b + 1$ nodes are needed for B as well. □

D.4 BITWISE ENCODING OF INTEGER DISTRIBUTIONS

Let s_i denote the i^{th} bit of a 1-indexed bit string; for example, $0010_3 = 1$. Let $|s|$ denote the length of a bit string. Let $\frown$ denote concatenation for bits and bit strings.

Definition 3. Define $\mathrm{encode}_{\mathrm{BITWISE\_INT}}(\mathcal{D}_b) = (B, \vec{p})$ such that:

For each bit string s, $0 \leq |s| < b$, there is a decision variable x_s with truth probability $p_s = \Pr(v^\star_{|s|+1} \mid \bigwedge_{i=1}^{|s|} v^\star_i = s_i)$.

Intuitively, the bits are chosen left-to-right, and each decision variable chooses the next bit given a bit string of past choices. Consider the following examples, where the empty bit string is denoted as ε.

$$\begin{aligned} \Pr(x_\varepsilon) &= \Pr(v^\star_1) \\ \Pr(x_0) &= \Pr(v^\star_2 \mid \neg v^\star_1) \\ \Pr(x_1) &= \Pr(v^\star_2 \mid v^\star_1) \\ \Pr(x_{0100}) &= \Pr(v^\star_5 \mid \neg v^\star_1 \wedge v^\star_2 \wedge \neg v^\star_3 \wedge \neg v^\star_4) \end{aligned}$$

We specify $B_i(\vec{x})$ for $1 \leq i \leq b$:

$$B_i(\vec{x}) = x_{B_1(\vec{x}) \frown B_2(\vec{x}) \frown \ldots \frown B_{i-1}(\vec{x})}$$

For example, $B_1(\vec{x}) = x_\varepsilon$, $B_2(\vec{x}) = x_{B_1(\vec{x})}$, $B_3(\vec{x}) = x_{B_1(\vec{x}) \frown B_2(\vec{x})}$, and so on. Note that the specification above is equivalent to the following, which is a more natural representation in probabilistic programming, and an "unrolled" version of Algorithm 2.

$$\begin{aligned} B_1(\vec{x}) &= x_\varepsilon \\ B_2(\vec{x}) &= \text{if } x_\varepsilon \text{ then } x_1 \text{ else } x_0 \\ B_3(\vec{x}) &= \text{if } x_\varepsilon \text{ then (if } x_1 \text{ then } x_{11} \text{ else } x_{10}) \text{ else (if } x_0 \text{ then } x_{01} \text{ else } x_{00}) \\ &\vdots \\ B_n(\vec{x}) &= \ldots \end{aligned}$$

Lemma 4. *For any distribution $\mathcal{D}_b$ over unsigned integers,* $\mathrm{encode}_{\mathrm{BITWISE\_INT}}(\mathcal{D}_b)$ *encodes $\mathcal{D}_b$.*

Proof. We prove that for any possible assignment to bits $\vec{a} \in \{0,1\}^b$, $\Pr(B(\vec{X}) = \vec{a}) = \Pr(\vec{v^\star} = \vec{a})$.

$$
\begin{aligned}
\Pr(B(\vec{X}) = \vec{a}) &= \prod_{i=1}^{b} \Pr(B_i(\vec{X})) = a_i \mid \bigwedge_{j=1}^{i-1} B_j(\vec{X}) = a_j) && \text{(Chain rule of probability)} \\
&= \prod_{i=1}^{b} \Pr(X_{B_1(\vec{X})^\frown \ldots ^\frown B_{i-1}(\vec{X})} = a_i \mid \bigwedge_{j=1}^{i-1} B_j(\vec{X}) = a_j) && \text{(Bit specification)} \\
&= \prod_{i=1}^{b} \Pr(X_{a_1^\frown \ldots ^\frown a_{i-1}} = a_i \mid \bigwedge_{j=1}^{i-1} B_j(\vec{X}) = a_j) && \text{(Condition)} \\
&= \prod_{i=1}^{b} \Pr(X_{a_1^\frown \ldots ^\frown a_{i-1}} = a_i \mid \bigwedge_{j=1}^{i-1} X_{B(\vec{X})_1^\frown \ldots ^\frown B_{j-1}(\vec{X})} = a_j) && \text{(Bit specification)} \\
&= \prod_{i=1}^{b} \Pr(X_{a_1^\frown \ldots ^\frown a_{i-1}} = a_i) && \text{(Independence)} \\
&= \prod_{i=1}^{b} \Pr(b_i^\star = a_i \mid \bigwedge_{j=1}^{i-1} b_j^\star = a_j) && \text{(As } \Pr(X_s) = p_s\text{)} \\
&= \Pr(\vec{b^\star} = \vec{a}) && \text{(Chain rule of probability)}
\end{aligned}
$$

□

D.5 BDD SIZE OF THE BITWISE_INT ENCODING

We prove the second half of Proposition 1.

Proposition 1 (second half). A discrete distribution over the integers $\{0, 1 \ldots, 2^b - 1\}$ compiles to a BDD of size $\Theta(2^b)$ when represented using BITWISE_INT(Algorithm 2).

Lemma 5. *Let* $\text{encode}_{\text{BITWISE\_INT}}(\mathcal{D}_b) = (B, P)$.

B exactly matches a BDD C with the following reduced structure.

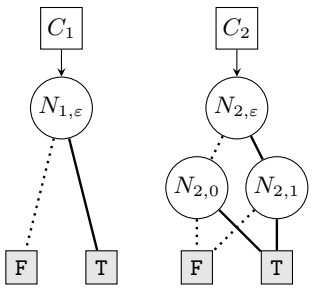
For all $1 \leq i \leq b$, for all bit strings s of length less than i, let C have a node $N_{i,s}$ corresponding to decision variable x_s.

$$
\begin{aligned}
\text{low}(N_{i,s}) &= \textit{if} \quad |s| = i-1 \quad \textit{then} \quad 0 \quad \textit{else} \quad N_{i,s^\frown 0} \\
\text{high}(N_{i,s}) &= \textit{if} \quad |s| = i-1 \quad \textit{then} \quad 1 \quad \textit{else} \quad N_{i,s^\frown 1}
\end{aligned}
$$

Let the roots of C be placed such that $C_i(\vec{x})$ matches $N_{i,\varepsilon}$ given assignments to decision variables $\vec{x}$.

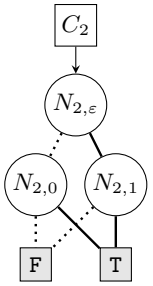
Any ordering of decision variables is valid as long is they are ordered by increasing bit string lengths. We arbitrarily choose to order bit strings of the same length by their lexicographical order.

As an example, the BDD for $b = 2$ follows (terminal nodes visually duplicated for clarity).



Proof. Let $S_{i,j}(\vec{x})$ denote the state of C_i at the level of the first decision variable whose bit string is of length j. See the following BDD and corresponding states as an example.

$$
\begin{aligned}
S_{2,0}(x_\varepsilon, x_0, x_1) &= N_{2,\varepsilon} \\
S_{2,1}(0, x_0, x_1) &= N_{2,0} \\
S_{2,1}(1, x_0, x_1) &= N_{2,1} \\
S_{2,2}(0, x_0, x_1) &= x_0 \\
S_{2,2}(1, x_0, x_1) &= x_1
\end{aligned}
$$



Let $P(t)$ be the statement, $S_{i,t}(\vec{x}) = \text{if} \quad t \geq i \quad \text{then} \quad B_i(\vec{x}) \quad \text{else} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})}$.

$P(0)$ holds as there is no bit index greater than or equal to 0 and $S_{i,0}(\vec{x}) = N_{i,\varepsilon}$ by the placement of the roots.

Assume $P(t)$, which specifies $S_{i,t}$. Consider advancing to the next node in the BDD based on the assignment to decision variables (which does nothing if we are already at a terminal node):

$$
\begin{aligned}
\mathrm{next(S_{i,t})} = \text{if} \quad & t \geq i \\
\text{then} \quad & B_i(\vec{x}) \\
\text{else} \quad & \big(\text{if} \quad x_{B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})} \quad \text{then} \quad \mathrm{high}\big(N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})}\big) \quad \text{else} \quad \mathrm{low}\big(N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})}\big)\big)
\end{aligned}
$$

We replace $x_{B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})}$ with $B_{t+1}(\vec{x})$.

$$
\begin{aligned}
\mathrm{next(S_{i,t})} = \text{if} \quad & t \geq i \\
\text{then} \quad & B_i(\vec{x}) \\
\text{else} \quad & \big(\text{if} \quad B_{t+1}(\vec{x}) \quad \text{then} \quad \mathrm{high}\big(N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})}\big) \quad \text{else} \quad \mathrm{low}\big(N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})}\big)\big)
\end{aligned}
$$

We replace the high and low edges by the definition:

$$
\begin{aligned}
\mathrm{next(S_{i,t})} = \text{if} \quad & t \geq i \\
\text{then} \quad & B_i(\vec{x}) \\
\text{else} \quad & (\quad \text{if} \quad B_{t+1}(\vec{x}) \\
& \quad \text{then} \quad \big(\text{if} \quad t = i-1 \quad \text{then} \quad 1 \quad \text{else} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})^\frown 1}\big) \\
& \quad \text{else} \quad \big(\text{if} \quad t = i-1 \quad \text{then} \quad 0 \quad \text{else} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})^\frown 0}\big)\big)
\end{aligned}
$$

We rearrange the if conditions:

$$
\begin{aligned}
\mathrm{next(S_{i,t})} = \text{if} \quad & t \geq i \\
\text{then} \quad & B_i(\vec{x}) \\
\text{else} \quad & (\quad \text{if} \quad t = i-1 \\
& \quad \text{then} \quad (\text{if} \quad B_{t+1}(\vec{x}) \quad \text{then} \quad 1 \quad \text{else} \quad 0) \\
& \quad \text{else} \quad \big(\text{if} \quad B_{t+1}(\vec{x}) \quad \text{then} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})^\frown 1} \quad \text{else} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_t(\vec{x})^\frown 0}\big)\big) \\
\mathrm{next(S_{i,t})} = \text{if} \quad & t \geq i \\
\text{then} \quad & B_i(\vec{x}) \\
\text{else} \quad & (\quad \text{if} \quad t = i-1 \\
& \quad \text{then} \quad B_{t+1}(\vec{x}) \\
& \quad \text{else} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_{t+1}(\vec{x})}\big)
\end{aligned}
$$

The first two then branches collapse:

$$
\begin{aligned}
\mathrm{next(S_{i,t})} = \text{if} \quad & t+1 \geq i \\
\text{then} \quad & B_i(\vec{x}) \\
\text{else} \quad & N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_{t+1}(\vec{x})}
\end{aligned}
$$

As the only node we now reach has bit string length $t+1$, $\text{next}(S_{i,t}) = S_{i,t+1}$. Therefore $P(t) \to P(t+1)$.

Consider $P(b-1)$: $S_{i,b-1} = \text{if} \quad b-1 \geq i \quad \text{then} \quad B_i(\vec{x}) \quad \text{else} \quad N_{i,B_1(\vec{x})^\frown \ldots ^\frown B_{b-1}(\vec{x})}$.

For all $i \in \{1, \ldots, b-1\}$, we see that the root labeled $C_i(\vec{x})$ reaches the value $B_i(\vec{x})$. For $i = b$, the state reaches $N_{n,B_1(\vec{x})^\frown \ldots ^\frown B_{b-1}(\vec{x})}$, whose high and low edges are 1 and 0, respectively. The last step goes to $\big(\text{if} \quad x_{b_1^\frown \ldots ^\frown b_{b-1}} \quad \text{then} \quad 1 \quad \text{else} \quad 0\big) = B_b(\vec{x})$. Therefore B and C are equivalent. □

Proposition 1 (second half).

Proof. Let $\text{encode}_{\text{BITWISE\_INT}}(\mathcal{D}_b) = (B, \vec{p})$. B has $2^{b+1} - b - 2$ decision nodes.

It directly follows from Lemma 5 that B requires the following number of nodes.

$$\sum_{i=1}^{b} (\text{\# of bit strings of length less than } i) = 2^{n+1} - n - 2$$

□

D.6 ENCODING UNIFORM INTEGER DISTRIBUTIONS

Let U_n denote the discrete uniform distribution over the integers $\{0, 1, ..., n-1\}$

$$U_n(i) = \begin{cases} \frac{1}{n} & i \in 0, 1, ..., n-1 \\ 0 & \text{otherwise} \end{cases}$$

Lemma 6. *For all non-negative integers b and i such that* $0 < i < b$, $x_i \in \{0, 1\}$

$$U_{2^b}(< x_b, x_{b-1}, \ldots, x_1 >) = \frac{1}{2} \cdot U_{2^{b-1}}(< x_{b-1}, \ldots, x_1 >)$$

Proof. Let $a =< x_b, x_{b-1}, \ldots, x_1 >$, then

$$U_{2^b}(a) = \frac{1}{2^b} = \frac{1}{2} \cdot \frac{1}{2^{b-1}} = \frac{1}{2} \cdot U_{2^{b-1}}(a - x_b 2^{b-1}) = \frac{1}{2} \cdot U_{2^{b-1}}(< x_{b-1}, \ldots, x_1 >)$$

□

Lemma 7. *For integers b and i such that* $0 < i < b$, $x_i \in \{0, 1\}$ *and* $b > 1$

$$[UNIFORM(b)](x_b, x_{b-1}, ..., x_1) = \frac{1}{2}[UNIFORM(b-1)](x_{b-1}, ..., x_1)$$

Proof. Consider the case when $x_b = 0$. The proof for the other case when $x_b = 1$ would go in a similar fashion. Let π be the set of executions of UNIFORM(b) resulting in $[0, x_{b-1}, ..., x_1]$. Let π' be the set of executions of UNIFORM($b-1$) resulting in $[x_{b-1}, ..., x_1]$. Since $b > 1$ is false, X_b gets assigned flip$_1$ evaluating to false; we return 0 ::UNIFORM($b-1$).

$$\begin{aligned} &[UNIFORM(b)](0, x_{b-1}, ..., x_1) \\ &= \sum_{p \in \pi} w(p) = \frac{1}{2} \sum_{p \in \pi'} w(p) = \frac{1}{2}[UNIFORM(b-1)](x_{b-1}, ..., x_1) \end{aligned}$$

□

Proposition 2. $\forall b > 0$, [UNIFORM(b)] = U_{2^b}

Proof. The proof is by induction on b
Base Case: For $b = 1$,

$$U_2(a) = \begin{cases} \frac{1}{2} & a = 0 \\ \frac{1}{2} & a = 1 \\ 0 & \text{otherwise} \end{cases}$$

Executing UNIFORM(1) would result in $X_1 = \text{flip}\frac{1}{2}$. So, $\Pr(X_1 = 1) = \frac{1}{2}$ and $\Pr(X_1 = 0) = \frac{1}{2}$
Induction Hypothesis: $\forall b' < b\ [UNIFORM(b')] = U_{2^{b'}}$
Inductive Step: To prove $[UNIFORM(b)] = U_{2^b}$
Let the output of UNIFORM(b) be a where $a_b =< x_b, x_{b-1}, ..., x_1 >$

Case 1: $\forall 0 \leq a < 2^{b-1}$
Since $a < 2^{b-1}, x_b = 0$

$$\begin{aligned} &[\text{UNIFORM}(b)](0, x_{b-1}, ..., x_1) \\ &= \frac{1}{2}[\text{UNIFORM}(b-1)](x_{b-1}, ..., x_1) = \frac{1}{2} \cdot U_{2^{b-1}}(< x_{b-1}, ..., x_1 >) && \text{(by Lemma 7 and I.H.)} \\ &= U_{2^b}(< 0, x_{b-1}, ..., x_1 >) && \text{(by Lemma 6)} \end{aligned}$$

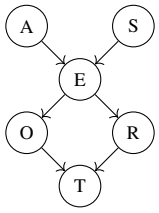
Case 2: $\forall 2^{b-1} \leq a < 2^b$
Since $a \geq 2^{b-1}, x_b = 1$, the proof is similar to that for Case 1. □

E BETA PRIOR APPLICATION: BAYESIAN NETWORK PARAMETER LEARNING

One natural application of a Beta prior is as a prior distribution for learning the parameters of a (binary) Bayesian network. The task of Bayesian network parameter learning can be described as follows: given a set of data consisting of instantiations of network variables, we want to find the network parameters maximizing the probability of this data. One interesting case is when our data is incomplete; that is, there are some variables not given a value. In this setting, a Bayesian approach to parameter learning must consider all (exponentially many) possible instantiations of these missing values [Darwiche, 2009].

Using our Beta prior implementation described in the main paper, this setting can naturally be modeled within `Dice.jl`.. A Bayesian network can be expressed in the probabilistic program with Beta priors on each network parameter; a dataset can then be observed. By returning the distribution over our Beta parameters α and β, we obtain our posterior, a mixture over Beta distributions. These can then be manually combined to obtain the exact posterior density function.

We provide a brief example of this on the survey Bayesian network[1]. The structure of this network is shown below; we simplify the network by making all variables binary so that the Beta a suitable prior. We note that we can actually generalize to the non-binary case with a Dirichlet prior using an approach similar to that used for the Beta.



Suppose we want to learn parameters from the dataset given below; the entry ? indicates a missing value. We focus on the specific parameter $\theta_{o|e} = \Pr(O = 1|E = 1)$, which we give a uniform prior $\text{Beta}(1, 1)$.

[1] https://www.bnlearn.com/bnrepository/

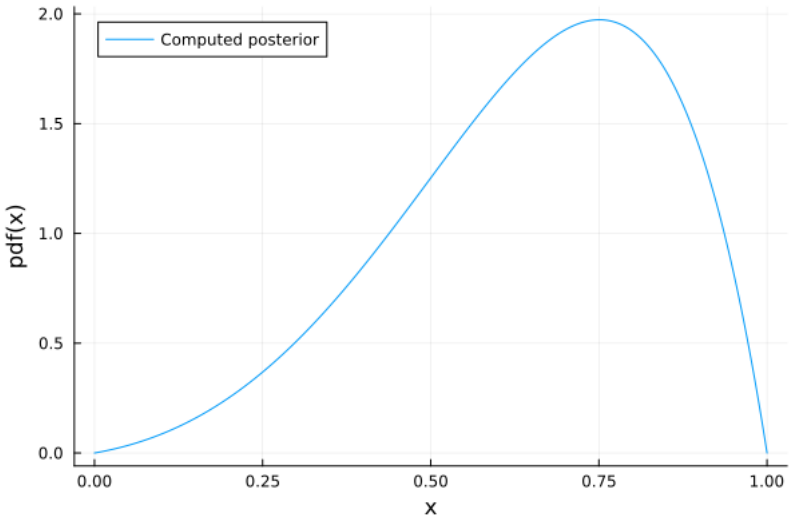
A	S	E	O	R	T
1	?	?	1	1	1
1	0	?	1	0	1
1	0	1	?	0	1
1	?	1	0	?	1
0	0	1	1	?	1
0	1	?	1	1	?
1	?	0	0	1	0
0	?	1	?	?	?
1	1	1	?	1	?
1	0	0	?	1	1

By running the program representing this task, we get the following large output distribution over Beta parameters - a mixture over Beta distributions. Note that it does not contain as many entries as possible instantiations, as some complete variable instantiations result in the same posterior Beta.

α	β	Pr(.)
9	3	0.226
8	4	0.207
10	2	0.177
7	5	0.160
6	6	0.109
5	7	0.066
4	8	0.035
3	9	0.016
2	10	0.005

If we plot the corresponding mixture of Betas, we can get the following posterior PDF — note that this is an exact posterior recovered from the Beta parameters, in constrast to the approximate result one would get from sampling-based inference methods.

The code used in this example is available in the same `Dice.jl` repository (`https://github.com/Juice-jl/Dice.jl/tree/arithmetic`).



References

Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. doi: 10.1017/CBO9780511811357.

Samuel Dittmer and Igor Pak. Counting linear extensions of restricted posets, 2018. URL https://arxiv.org/abs/1802.06312.

Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, jun 1962. ISSN 0001-0782. doi: 10.1145/367766.368168. URL https://doi.org/10.1145/367766.368168.

Timon Gehr, Sasa Misailovic, and Martin Vechev. Psi: Exact symbolic inference for probabilistic programs. In *International Conference on Computer Aided Verification*, pages 62–83. Springer, 2016.

Timon Gehr, Sasa Misailovic, and Martin Vechev. Psi: Exact symbolic solver github, 2022. URL https://github.com/eth-sri/psi/tree/master/test.

Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Scaling exact inference for discrete probabilistic programs. In *Proc. ACM Program. Lang.*, OOPSLA 2020, pages 140:1–140:31. Association for Computing Machinery, 2020. doi: 10.1145/3428208.

Zixin Huang, Saikat Dutta, and Sasa Misailovic. Aqua: Automated quantized inference for probabilistic programs. In *International Symposium on Automated Technology for Verification and Analysis*, pages 229–246. Springer, 2021.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams: Learning with massive logical constraints. In *ICML Workshop on Learning Tractable Probabilistic Models (LTPM)*, June 2014. URL http://starai.cs.ucla.edu/papers/KisaLTPM14.pdf.

Jacob Laurel and Sasa Misailovic. Continualization of probabilistic programs with correction. In Peter Müller, editor, *Programming Languages and Systems*, pages 366–393, Cham, 2020. Springer International Publishing. ISBN 978-3-030-44914-8.

D. H. Lehmer. Euclid's algorithm for large numbers. *The American Mathematical Monthly*, 45(4):227–233, 1938. doi: 10.1080/00029890.1938.11990797. URL https://doi.org/10.1080/00029890.1938.11990797.

H.P. Luhn. Computer for verifying numbers. U.S. Patent US2950048A, 1960.