Symbolic Methods for Probabilistic Inference, Optimization, and Decision-making

Scott Sanner

With much thanks to research collaborators:
Zahra Zamani, Ehsan Abbasnejad, Karina Valdivia Delgado, Leliane Nunes de Barros, Luis Gustavo Rocha Vianna, Cheng (Simon) Fang
Graphical Models are Pervasive in AI

- **Medical**
  - Pathfinder: Expert System
  - BUGS: Epidemiology

- **Text**
  - LDA and 1,000 Extensions

- **Vision**
  - Ising Model!

- **Robotics**
  - Dynamics and sensor models
Graphical Models + Symbolic Methods

• Specify a graphical model for problem at hand
  – Text, vision, robotics, etc.

• Goal: efficient inference and optimization in this model
  – Be it discrete or continuous

• Symbolic methods (e.g., decision diagrams) facilitate this!
  – Useful as building block in any inference algorithm
  – Exploit structure for compactness, efficient computation
    • Automagically!

Still partially a dream, but many recent advances

Exploit more structure than graphical model alone
Tutorial Outline

• Part I: Symbolic Methods for Discrete Inference
  – Graphical Models and Influence Diagrams
  – Symbolic Inference with Decision Diagrams

• Part II: Extensions to Continuous Inference
  – Case Calculus
  – Extended ADD (XADD)

• Part III: Applications
  – Graphical Model Inference
  – Sequential Decision-making
  – Constrained Optimization
Part I: Symbolic Methods for Discrete Inference and Optimization
Directed Graphical Models

Bayesian Network:

– compact (factored) specification of joint probability
– e.g., have binary variables $B, F, A, H, P$:

$$P(B,F,A,H,P) = P(H|B,F) \ P(P|F,A) \ P(B) \ P(F) \ P(A)$$
Undirected Graphical Models

- **Markov Random Fields (MRFs)**
  \[ P(V_1, V_2, V_3, V_4) = \frac{1}{Z} F(V_1, V_2) F(V_2, V_4) F(V_1, V_3) F(V_3, V_4) \]

- **Conditional MRFs (CRFs)**
  \[ P(V_1, V_2 | V_3, V_4) = \frac{1}{Z(V_3, V_4)} F(V_1, V_2) F(V_2, V_4) F(V_1, V_3) F(V_3, V_4) \]

Note: representation above is *factor graph* (FG), which works for directed models as well. For FGs of directed models, what conditions also hold?)
Dynamical Models & Influence Diagrams

• Dynamical models…
  – Represent state @ times t, t+1
    • Assume stationary distribution

• Influence diagrams…
  – Action nodes [squares]
    • Not random variables
    • Rather “controlled” variables
  – Utility nodes <diamonds>
    • A utility conditioned on state, e.g.
      \[ U(X_1', X_2') = \begin{cases} 10 & \text{if } X_1' = X_2' \\ 0 & \text{otherwise} \end{cases} \]
Discrete Inference & Optimization

- Probabilistic Queries in Bayesian networks:
  - Want $P(\text{Query}|\text{Evidence})$, e.g.
    
    $$P(E|X) = \frac{P(E,X)}{P(X)} \propto \sum_A \sum_B P(A,B,E,X) = \sum_A \sum_B P(A|B,E) P(X|B) P(E) P(X)$$

- Maximizing Exp. Utility in Influence Diagrams:
  - Want optimal action $a^* = \arg\max_a E[U|A=a,...]$, e.g.
    
    $$a^* = \arg\max_a E[U|A=a,X=x] = \arg\max_a \sum_{X'} U(X') P(X'|A=a,X=x)$$
Manipulating Discrete Distributions

• Marginalization

\[ \sum_b P(A, b) = P(A) \]

\[ \sum_b P(A, b) = P(A) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>.1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>.3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>.4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.4</td>
</tr>
<tr>
<td>1</td>
<td>.6</td>
</tr>
</tbody>
</table>
Manipulating Discrete Distributions

- Maximization

\[
\max_b P(A, b) = P(A)
\]

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Pr} \\
0 & 0 & .1 \\
0 & 1 & .3 \\
1 & 0 & .4 \\
1 & 1 & .2 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{A} & \text{Pr} \\
0 & .3 (B=1) \\
1 & .4 (B=0) \\
\end{array}
\]
Manipulating Discrete Distributions

• Binary Multiplication

\[ P(A|B) \cdot P(B|C) = P(A, B|C) \]

\[
\begin{array}{c|c|c} 
A & B & Pr \\
\hline
0 & 0 & .1 \\
0 & 1 & .9 \\
1 & 0 & .2 \\
1 & 1 & .8 \\
\end{array}
\begin{array}{c|c|c} 
B & C & Pr \\
\hline
0 & 0 & .1 \\
0 & 1 & .9 \\
1 & 0 & .2 \\
1 & 1 & .8 \\
\end{array}
\begin{array}{c|c|c|c|c} 
A & B & C & Pr \\
\hline
0 & 0 & 0 & .01 \\
0 & 0 & 1 & .09 \\
0 & 1 & 0 & .18 \\
0 & 1 & 1 & .72 \\
\end{array}
\]

• Same principle holds for all binary ops
  \(-, +, -, /, \text{max}, \text{etc}...\)
Discrete Inference & Optimization

- **Observation 1**: all discrete functions can be tables

  \[
  \begin{array}{ccc}
  A & B & Pr \\
  0 & 0 & .1 \\
  0 & 1 & .3 \\
  1 & 0 & .4 \\
  1 & 1 & .2 \\
  \end{array}
  \]

  \( P(A,B) = \)

- **Observation 2**: all operations computable in closed-form
  - \( f_1 \oplus f_2, f_1 \otimes f_2 \)
  - \( \max(f_1, f_2), \min(f_1, f_2) \)
  - \( \sum_x f(x) \)
  - \((\text{arg})\max_x f(x), (\text{arg})\min_x f(x)\)

Now can do **exact inference and optimization** in discrete graphical models and influence diagrams!
Discrete Inference & Optimization

• Probabilistic Queries in Bayesian networks:
  • Want $P(\text{Query}|\text{Evidence})$, e.g.

\[
P(E|X) = \frac{P(E,X)}{P(X)} \\
\propto \sum_A \sum_B P(A,B,E,X) \\
= \sum_A \sum_B P(A|B,E) P(X|B) P(E) P(X)
\]

• Maximizing Exp. Utility in Influence Diagrams:
  • Want optimal action $a^* = \text{argmax}_a E[U|A=a,\ldots]$, e.g.

\[
a^* = \text{argmax}_a E[U|A=a,X=x] \\
= \text{argmax}_a \sum_{X'} U(X') P(X'|A=a,X=x)
\]
Where are we?

We can specify discrete models

We know operations needed for inference

Can we optimize the order of operations?
Variable Elimination

- When marginalizing over $y$, try to factor out all probabilities independent of $y$:

$$P(X_1) = \sum_{x_2, \ldots, x_n, y} P(y|x_1, \ldots, x_n)P(X_1) \cdots P(x_n)$$

$$= P(X_1) \sum_{x_2} P(x_2) \cdots \sum_{x_n} P(x_n) \sum_{y} \underbrace{P(y|X_1, \ldots, x_n)}_{O(1)} = O(n)$$

- Curly braces show number of FLOPS

In tabular case.
VE: Variable Order Matters

• Sum commutes, can change order of elimination:

\[
P(X_1) = \sum_{y,x_n,...,x_2} P(y|x_1,...,x_n)P(X_1) \cdots P(x_n)
\]

\[
= \sum_{y,x_n,...,x_3} P(X_1)P(x_3) \cdots P(x_n) \sum_{x_2} P(x_2)P(y|X_1,...,x_n)
\]

\[= O(2^{n+1})\]

• With different variable order: \(O(n) \rightarrow O(2^{n+1})\)
  – Good variable order:
    • minimizes #vars in largest intermediate factor
    • a.k.a., \(\sim\)tree width (TW) = n+1
  – Graphical model inference is \(\sim O(2^{TW})\)

Actually TW+1 but the point is exponential
Recap

• Graphical Models and Influence Diagrams
  – why should you care?

• Versus non-factored models… GMs and IDs allow
  – exponential space savings in representation
  – exponential time savings in inference
    • exploit factored structure in variable elimination
  – exponential data reduction for learning
    • smaller models = fewer parameters = less data
Where are we?

- We can specify discrete models
- We know operations needed for inference
- We know how to optimize order of operations

Is this it? Is there more structure to exploit?
Symbolic Inference with Decision Diagrams

For Discrete Models
DD Definition

• Decision diagrams (DDs):
  – DAG variant of decision tree
  – Decision tests ordered
  – Used to represent:
    • \( f : B^n \rightarrow B \) (boolean – BDD, set of subsets \( \{\{a,b\},\{a\}\} \) – ZDD)
    • \( f : B^n \rightarrow Z \) (integer – MTBDD / ADD)
    • \( f : B^n \rightarrow R \) (real – ADD)

We’ll focus on ADDs in this tutorial.
What’s the Big Deal?

• More than compactness
  – Ordered decision tests in DDs support efficient operations
    • ADD: $\neg f, f \oplus g, f \otimes g, \max(f, g)$
    • BDD: $\neg f, f \land g, f \lor g$
    • ZDD: $f \setminus g, f \cap g, f \cup g$
  – Efficient operations key to inference
Function Representation (Tables)

- How to represent functions: $\mathbb{B}^n \rightarrow \mathbb{R}$?
- How about a fully enumerated table...
- …OK, but can we be more compact?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>F(a,b,c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Function Representation (Trees)

- How about a tree? Sure, can simplify.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>F(a, b, c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Context-specific independence!
Function Representation (ADDS)

• Why not a directed acyclic graph (DAG)?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>F(a,b,c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Exploits context-specific independence (CSI) and shared substructure.
Function Representation (ADDS)

- Why not a directed acyclic graph (DAG)?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>F(a,b,c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Exploits context-specific independence (CSI) and shared substructure.
Trees vs. ADDs

- Trees can compactly represent AND / OR
  - But not XOR (linear as ADD, exponential as tree)
  - Why? Trees must represent every path
Binary Operations (ADDS)

- Why do we order variable tests?
- Enables us to do efficient binary operations…

Result: ADD operations can avoid state enumeration
Summary

• We need $B^n \rightarrow B / Z / R$
  – We need compact representations
  – We need efficient operations

  → DDs are a promising candidate

• Great, tell me all about DDs…
  – OK 😊

Not claiming DDs solve all problems… but often better than tabular approach
Decision Diagrams: Reduce

(how to build canonical DDs)
How to Reduce Ordered Tree to ADD?

- Recursively build bottom up
  - Hash terminal nodes $R \rightarrow ID$
    - leaf cache
  - Hash non-terminal functions $(v, ID_0, ID_1) \rightarrow ID$
    - special case: $(v, ID, ID) \rightarrow ID$
    - others: keep in (reduce) cache
GetNode

- Removes redundant branches
- Maintains cache of internal nodes

**Algorithm 1: GetNode(v, F_h, F_l) \rightarrow F_r**

```plaintext
input : v, F_h, F_l : Var and node ids for high/low branches
output: F_r : Return values for offset, multiplier, and canonical node id

begin
  // If branches redundant, return child
  if (F_l = F_h) then
    return F_l;

  // Make new node if not in cache
  if ((v, F_h, F_l \rightarrow id is not in node cache) then
    id := currently unallocated id;
    insert \langle v, F_h, F_l \rangle \rightarrow id in cache;

  // Return the cached, canonical node
  return id;
end
```
Reduce Algorithm

**Algorithm 1: Reduce(F) → Fr**

**input**: F : Node id

**output**: Fr : Canonical node id for reduced ADD

begin

// Check for terminal node
if (F is terminal node) then

  return canonical terminal node for value of F;

// Check reduce cache
if (F → Fr is not in reduce cache) then

  // Not in cache, so recurse
  F_h := Reduce(F_h);
  F_l := Reduce(F_l);

  // Retrieve canonical form
  Fr := GetNode(F^var, F_h, F_l);

  // Put in cache
  insert F → Fr in reduce cache;

// Return canonical reduced node
return Fr;

end
Reduce Complexity

• Linear in size of input
  – Input can be tree or DAG

• Because of caching
  – Explores each node once
  – Does not need to explore all branches
Canonicity of ADDs via Reduce

• Claim: if two functions are identical, Reduce will assign both functions same ID

• By induction on var order
  – Base case:
    • Canonical for 0 vars: terminal nodes
  – Inductive:
    • Assume canonical for k-1 vars
    • GetNode result canonical for k\textsuperscript{th} var
      (only one way to represent)
Impact of Variable Orderings

• Good orders can matter

• Good orders typically have related vars together
  – e.g., low tree-width order in transition graphical model

Graph-Based Algorithms for Boolean Function Manipulation
In-diagram Reordering

• Rudell’s sifting algorithm
  – Global reordering as pairwise swapping
  – Only need to redirect arcs
    • Helps to use pointers
      → then don’t need to redirect parents, e.g.,

\[
\text{ID}_1 \quad \text{ID}_2 \quad \text{ID}_3 \quad \text{ID}_4
\]

\[
\text{Swap}(a, b)
\]

Can also do reorder using Apply… later
Beyond Binary Variables

• Multivalued (MV-)DDs
  – A DD with multivalued variables
  – straightforward $k$-branch extension
  – e.g., $k=6$

  ![Diagram](image)

  – Works for ADD extensions as well
Decision Diagrams: Apply

(how to do efficient operations on DDs)
Recap

- Recall the Apply recursion

Result: ADD operations can avoid state enumeration

Need to handle base cases

Need to handle recursive cases
Apply Recursion

- Need to compute $F_1 \ op \ F_2$
  - e.g., $op \in \{\oplus, \otimes, \land, \lor\}$

- Case 1: $F_1$ & $F_2$ match vars

  $F_h = \text{Apply}(F_{1,h}, F_{2,h}, op)$
  $F_l = \text{Apply}(F_{1,l}, F_{2,l}, op)$
  $F_r = \text{GetNode}(F_{1,\text{var}}, F_h, F_l)$

$F_r = \text{Apply}(F_{1,h}, F_{2,h}, op)$
$F_r = \text{Apply}(F_{1,l}, F_{2,l}, op)$
Apply Recursion

• Need to compute $F_1 \ op \ F_2$
  – e.g., $\ op \in \{\oplus, \odot, \land, \lor\}$

• Case 2: Non-matching var: $v_1 \not< v_2$

\[
\begin{align*}
F_h &= Apply(F_1, F_{2,h}, \ op) \\
F_l &= Apply(F_1, F_{2,l}, \ op) \\
F_r &= GetNode(F_{2,\text{var}}, F_h, F_l)
\end{align*}
\]
Apply Base Case: ComputeResult

- Constant (terminal) nodes and some other cases can be computed without recursion

<table>
<thead>
<tr>
<th>Operation and Conditions</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1 \ op F_2; F_1 = C_1; F_2 = C_2$</td>
<td>$C_1 \ op C_2$</td>
</tr>
<tr>
<td>$F_1 \oplus F_2; F_2 = 0$</td>
<td>$F_1$</td>
</tr>
<tr>
<td>$F_1 \oplus F_2; F_1 = 0$</td>
<td>$F_2$</td>
</tr>
<tr>
<td>$F_1 \ominus F_2; F_2 = 0$</td>
<td>$F_1$</td>
</tr>
<tr>
<td>$F_1 \odot F_2; F_2 = 1$</td>
<td>$F_1$</td>
</tr>
<tr>
<td>$F_1 \odot F_2; F_1 = 1$</td>
<td>$F_2$</td>
</tr>
<tr>
<td>$F_1 \ominus F_2; F_2 = 1$</td>
<td>$F_1$</td>
</tr>
<tr>
<td>$\min(F_1, F_2); \max(F_1) \cdot \min(F_2)$</td>
<td>$F_1$</td>
</tr>
<tr>
<td>$\min(F_1, F_2); \max(F_2) \cdot \min(F_1)$</td>
<td>$F_2$</td>
</tr>
<tr>
<td>similarly for max</td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>null</td>
</tr>
</tbody>
</table>

Table 1: Input and output summaries of ComputeResult.
Algorithm 1: \( \text{Apply}(F_1, F_2, \text{op}) \rightarrow F_r \)

\[
\begin{align*}
\text{input} : & F_1, F_2, \text{op} : \text{ADD nodes and op} \\
\text{output} : & F_r : \text{ADD result node to return} \\
\text{begin} \\
& // \text{Check if result can be immediately computed} \\
& \text{if} (\text{ComputeResult}(F_1, F_2, \text{op}) \rightarrow F_r \text{ is not null}) \text{ then} \\
& \quad \text{return } F_r; \\
& // \text{Check if result already in apply cache} \\
& \text{if} (\langle F_1, F_2, \text{op} \rangle \rightarrow F_r \text{ is not in apply cache}) \text{ then} \\
& \quad // \text{Not terminal, so recurse} \\
& \quad \text{var} := \text{GetEarliestVar}(F_1^{\text{var}}, F_2^{\text{var}}); \\
& \quad // \text{Set up nodes for recursion} \\
& \quad \text{if} (F_1 \text{ is non-terminal} \land \text{var} = F_1^{\text{var}}) \text{ then} \\
& \quad \quad F_l^{v_1} := F_{1,l}; \quad F_h^{v_1} := F_{1,h}; \\
& \quad \text{else} \\
& \quad \quad F_{l/h}^{v_1} := F_1; \\
& \quad \quad \text{if} (F_2 \text{ is non-terminal} \land \text{var} = F_2^{\text{var}}) \text{ then} \\
& \quad \quad \quad F_l^{v_2} := F_{2,l}; \quad F_h^{v_2} := F_{2,h}; \\
& \quad \quad \text{else} \\
& \quad \quad \quad F_{l/h}^{v_2} := F_2; \\
& \quad // \text{Recurse and get cached result} \\
& \quad F_l := \text{Apply}(F_l^{v_1}, F_l^{v_2}, \text{op}); \\
& \quad F_h := \text{Apply}(F_h^{v_1}, F_h^{v_2}, \text{op}); \\
& \quad F_r := \text{GetNode}(\text{var}, F_h, F_l); \\
& \quad // \text{Put result in apply cache and return} \\
& \quad \text{insert } \langle F_1, F_2, \text{op} \rangle \rightarrow F_r \text{ into apply cache}; \\
& \quad \text{return } F_r;
\end{align*}
\]

Note: Apply works for any binary operation! Why?
Apply Properties

• **Apply uses** *Apply cache*
  - \((F_1, F_2, \text{op}) \rightarrow F_R\)

• **Complexity**
  - Quadratic: \(O(|F_1| \cdot |F_2|)\)
    - \(|F|\) measured in node count
  - Why?
    - Cache implies touch every pair of nodes at most once!

• **Canonical?**
  - Same inductive argument as Reduce
Reduce-Restrict

• Important operation

• Have
  – $F(x,y,z)$

• Want
  – $G(x,y) = F|_{z=0}$

• Restrict $F|_{v=value}$ operation performs a *Reduce*
  – Just returns branch for $v=value$ whenever $v$ reached
  – Need *Restrict-Reduce cache* for $O(|F|)$ complexity
Marginalization, etc.

- Use Apply + Reduce-Restrict
  \[ \sum_x F(x, \ldots) = F|_{x=0} \oplus F|_{x=1}, \text{ e.g.} \]

- Likewise for similar operations...
  - **ADD**: \[ \min_x F(x, \ldots) = \min(F|_{x=0}, F|_{x=1}) \]
  - **BDD**: \[ \exists x \ F(x, \ldots) = F|_{x=0} \lor F|_{x=1} \]
  - **BDD**: \[ \forall x \ F(x, \ldots) = F|_{x=0} \land F|_{x=1} \]
Apply Tricks I

• Build \( F(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \)
  – Don’t build a tree and then call Reduce!
  – Just use indicator DDs and Apply to compute

\[
\begin{array}{c}
\oplus \\
1 & 0 & 1 & 0 & 1 & 0
\end{array}
\]

– In general:

• Build *any* arithmetic expression bottom-up using Apply!

\[
x_1 + (x_2 + 4x_3) * (x_4)
\rightarrow x_1 \oplus (x_2 \oplus (4 \otimes x_3)) \otimes (x_4)
\]
Apply Tricks II

- Build *ordered* DD from *unordered* DD

\[ z \text{ is out of order} \quad \rightarrow \quad \text{result will have } z \text{ in order!} \]
Affine ADDs
ADD Inefficiency

- Are ADDs enough?
- Or do we need more compactness?
- **Ex. 1: Additive reward/utility functions**
  - \( R(a,b,c) = R(a) + R(b) + R(c) \)
  - \( = 4a + 2b + c \)

- **Ex. 2: Multiplicative value functions**
  - \( V(a,b,c) = V(a) \cdot V(b) \cdot V(c) \)
  - \( = \gamma(4a + 2b + c) \)
Affine ADD (AADD)

- Define a new decision diagram – **Affine ADD**

- Edges labeled by **offset** \((c)\) and **multiplier** \((b)\):

- **Semantics**: if \((a)\) then \((c_1+b_1F_1)\) else \((c_2+b_2F_2)\)
Affine ADD (AADD)

- Maximize sharing by **normalizing** nodes $[0,1]$

- Example: if $(a)$ then $(4)$ else $(2)$

![Diagram showing normalization process](image-url)
AADD Reduce

Key point: automatically finds additive structure
AADD Examples

• Back to our previous examples…

• Ex. 1: Additive reward/utility functions

  • \( R(a,b) = R(a) + R(b) \)
    \( = 2a + b \)

• Ex. 2: Multiplicative value functions

  • \( V(a,b) = V(a) \cdot V(b) \)
    \( = \gamma^{2a + b}; \gamma < 1 \)
AADD Apply & Normalized Caching

- Need to normalize Apply cache keys, e.g., given

$$\langle 3 + 4F_1 \rangle \oplus \langle 5 + 6F_2 \rangle$$

- before lookup in Apply cache, normalize

$$8 + 4 \cdot \langle 0 + 1F_1 \rangle \oplus \langle 0 + 1.5F_2 \rangle$$

<table>
<thead>
<tr>
<th>Operation and Conditions</th>
<th>Normalized Cache Key and Computation</th>
<th>Result Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$\langle c_1 + b_1 F_1 \rangle \oplus \langle c_2 + b_2 F_2 \rangle; ; F_1 \neq 0$$</td>
<td>$$\langle c_r + b_r F_r \rangle = \langle 0 + 1F_1 \rangle \oplus \langle 0 + (b_2/b_1)F_2 \rangle$$</td>
<td>$$\langle (c_1 + c_2 + b_1 c_r) + b_1 b_r F_r \rangle$$</td>
</tr>
<tr>
<td>$$\langle c_1 + b_1 F_1 \rangle \oplus \langle c_2 + b_2 F_2 \rangle; ; F_1 \neq 0$$</td>
<td>$$\langle c_r + b_r F_r \rangle = \langle 0 + 1F_1 \rangle \oplus \langle 0 + (b_2/b_1)F_2 \rangle$$</td>
<td>$$\langle (c_1 - c_2 + b_1 c_r) + b_1 b_r F_r \rangle$$</td>
</tr>
<tr>
<td>$$\langle c_1 + b_1 F_1 \rangle \langle c_2 + b_2 F_2 \rangle; ; F_1 \neq 0$$</td>
<td>$$\langle c_r + b_r F_r \rangle = \langle (c_1/b_1) + F_1 \rangle \langle (c_2/b_2) + F_2 \rangle$$</td>
<td>$$\langle b_1 b_2 c_r + b_1 b_2 b_r F_r \rangle$$</td>
</tr>
<tr>
<td>$$\langle c_1 + b_1 F_1 \rangle \langle c_2 + b_2 F_2 \rangle; ; F_1 \neq 0$$</td>
<td>$$\langle c_r + b_r F_r \rangle = \langle (c_1/b_1) + F_1 \rangle \oplus \langle (c_2/b_2) + F_2 \rangle$$</td>
<td>$$\langle (b_1/b_2) c_r + (b_1/b_2) b_r F_r \rangle$$</td>
</tr>
<tr>
<td>$$\max(\langle c_1 + b_1 F_1 \rangle, \langle c_2 + b_2 F_2 \rangle); ; F_1 \neq 0$$</td>
<td>$$\langle c_r + b_r F_r \rangle = \max(\langle 0 + 1F_1 \rangle, \langle (c_2 - c_1)/b_1 + (b_2/b_1)F_2 \rangle)$$</td>
<td>$$\langle (c_1 + b_1 c_r) + b_1 b_r F_r \rangle$$</td>
</tr>
<tr>
<td>any $$\langle op \rangle$$ not matching above: $$\langle c_1 + b_1 F_1 \rangle \langle op \rangle \langle c_2 + b_2 F_2 \rangle$$</td>
<td>$$\langle c_r + b_r F_r \rangle = \langle c_1 + b_1 F_1 \rangle \langle op \rangle \langle c_2 + b_2 F_2 \rangle$$</td>
<td>$$\langle c_r + b_r F_r \rangle$$</td>
</tr>
</tbody>
</table>
ADDs vs. AADDs

• Additive functions: $\sum_{i=1..n} x_i$

Note: no context-specific independence, but subdiagrams shared: result size $O(n^2)$
ADDs vs. AADDs

- Additive functions: $\sum_i 2^i x_i$
  - Best case result for ADD (exp.) vs. AADD (linear)
ADDs vs. AADDs

- Additive functions: $\sum_{i=0}^{n-1} F(x_i, x_{(i+1) \mod n})$

Pairwise factoring evident in AADD structure
Main AADD Theorem

- AADD can yield exponential time/space improvement over ADD
  - and never performs worse!
- But...
  - Apply operations on AADDs can be exponential
  - Why?
  - Reconvergent diagrams possible in AADDs (edge labels), but not ADDs
  - Sometimes Apply explores all paths if no hits in normalized Apply cache
Recap: Symbolic Inference with DDs

- **Probabilistic Queries in Bayesian networks:**

  \[
  P(E|X) = \frac{P(E,X)}{P(X)} = \sum_A \sum_B P(A,B,E,X) = \sum_A \sum_B P(A|B,E) P(X|B) P(E) P(X)
  \]

- **Maximizing Exp. Utility in Influence Diagrams:**

  \[
  a^* = \arg\max_a \mathbb{E}[U|A=a,X=x] = \arg\max_a \sum_{X'} U(X') P(X'|A=a,X=x)
  \]

DDs can be used in **any** algorithm: use in VE, Loopy BP, Junction tree, etc. DDs automatically exploit structure in factors / msgs.
Approximate Inference

Sometimes no DD is compact, but bounded approximation is…
Problem: Value ADD Too Large

- Sum: \( \left( \sum_{i=1..3} 2^i \cdot x_i \right) + x_4 \cdot \varepsilon\)-Noise

- How to approximate?
Solution: APRICODD Trick

• Merge \( \approx \) leaves and reduce:

\[
\begin{array}{cccccc}
\text{x1} & \text{x2} & \text{x2} \\
6.045 & 2.02 & 0.005 & 4.02 & 5.05 & 1.025 & 3.05 & 7.085 \\
\end{array}
\]

• Error is bounded!
Can use ADD to Maintain Bounds!

- Change leaf to represent range \([L,U]\)
  - Normal leaf is like \([V,V]\)
  - When merging leaves…
    - keep track of min and max values contributing

More Compactness? AADDs?

- Sum: \((\sum_{i=1..3} 2^i \cdot x_i) + x_4 \cdot \varepsilon\)-Noise

- How to approximate? Error-bounded merge
Solution: MADCAP Trick

• Merge $\approx$ nodes from bottom up:

```
ROOT
  \text{<0 + 7.11 * >}
  \text{x1}
  \text{<0 + 0.852 * >} \text{<0.142 + 0.858 * >}
  \text{x2}
  \text{<0.332 + 0.668 * >} \text{<0 + 0.665 * >}
  \text{x3}
  \text{<0 + 0 * >} \text{<1 + 0 * >}
  0
```
Key Approximation Message

- automatic, efficient methods for finding logical, CSI, additive, and multiplicative structure in bounded approximations!
Example Inference Results using Decision Diagrams

Do they really work well?
Empirical Comparison: Table/ADD/AADD

- Sum: $\sum_{i=1}^{n} 2^i \cdot x_i \oplus \sum_{i=1}^{n} 2^i \cdot x_i$
- Prod: $\prod_{i=1}^{n} \gamma^{(2^i \cdot x_i)} \otimes \prod_{i=1}^{n} \gamma^{(2^i \cdot x_i)}$
Application: Bayes Net Inference

• Use variable elimination
  – Replace CPTs with ADDs or AADDs
  – Could do same for clique/junction-tree algorithms

• Exploits
  – Context-specific independence
    • Probability has logical structure:
    
    \[ P(a|b,c) = \text{if } b \text{ ? } 1 : \text{if } c \text{ ? } .7 : .3 \]
  
    – Additive CPTs
    • Probability is discretized linear function:
    
    \[ P(a|b_1\ldots b_n) = c + b \cdot \sum_i 2^i b_i \]
  
    – Multiplicative CPTs
    • Noisy-or (multiplicative AADD):
    
    \[ P(e|c_1\ldots c_n) = 1 - \prod_i (1 - p_i) \]

• If factor has above compact form, AADD exploits it
## Bayes Net Results: Various Networks

<table>
<thead>
<tr>
<th>Bayes Net</th>
<th>Table</th>
<th>ADD</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Entries</td>
<td># Nodes</td>
<td>Time</td>
</tr>
<tr>
<td>Alarm</td>
<td>1,192</td>
<td>689</td>
<td>2.97s</td>
</tr>
<tr>
<td>Barley</td>
<td>470,294</td>
<td>139,856</td>
<td>EML</td>
</tr>
<tr>
<td>Carpo</td>
<td>636</td>
<td>955</td>
<td>0.58s</td>
</tr>
<tr>
<td>Hailfinder</td>
<td>9,045</td>
<td>4,511</td>
<td>26.4s</td>
</tr>
<tr>
<td>Insurance</td>
<td>2,104</td>
<td>1,596</td>
<td>278s</td>
</tr>
<tr>
<td>Noisy-Or-15</td>
<td>65,566</td>
<td>125,356</td>
<td>50.2s</td>
</tr>
<tr>
<td>Noisy-Max-15</td>
<td>131,102</td>
<td>202,148</td>
<td>42.5s</td>
</tr>
</tbody>
</table>

*EML: Exceeded Memory Limit (1GB)
Application: MDP Solving

• SPUDD Factored MDP Solver (Hoey et al, 99)
  – Originally uses ADDs, can use AADDs
  – Implements factored value iteration…

\[ V^{n+1}(x_1 \ldots x_i) = R(x_1 \ldots x_i) + \gamma \cdot \max_a \sum_{x_1' \ldots x_i'} \prod_{F_1 \ldots F_i} P(x_1'|\ldots x_i) \ldots P(x_i'|\ldots x_i) \cdot V^n(x_1' \ldots x_i') \]
Application: SysAdmin

- SysAdmin MDP (GKP, 2001)
  - Network of computers: $c_1, \ldots, c_k$
  - Various network topologies
  - Every computer is running or crashed
  - At each time step, status of $c_i$ affected by
    - Previous state status
    - Status of incoming connections in previous state
  - Reward: +1 for every computer running (additive)
Results I: SysAdmin (10% Approx)
Results II: SysAdmin

Graphs showing the relationship between True Approximation Error and Space (# Nodes) for different methods, including APRICODD (ADD) and MADCAP (AADD).
Traffic Domain

• Binary **cell transmission model (CTM)**

• Actions
  – Light changes

• Objective:
  – Maximize empty cells in network
Results Traffic

![Bar chart showing time and space for different scenarios.]

- **Time (s):**
  - APRICODD: 6.5 x 10^5
  - MADCAP: 1.0 x 10^5

- **Space (# Nodes):**
  - APRICODD: 7.0 x 10^4
  - MADCAP: 4.0 x 10^4

Scenarios:
- 20 Vars, Exact
- 20 Vars, Approx(10%)
- 24 Vars, Exact
- 24 Vars, Approx(10%)
Application: POMDPs

- Provided an AADD implementation for Guy Shani’s factored POMDP solver
- Final value function size results:

<table>
<thead>
<tr>
<th>Network Management</th>
<th>ADD</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7000</td>
<td>92</td>
</tr>
<tr>
<td>Rock Sample</td>
<td>189</td>
<td>34</td>
</tr>
</tbody>
</table>
Inference with Decision Diagrams vs. Compilations (d-DNNF, etc.)

Important Distinctions
BDDs in NNF

- Can express BDD as NNF formula
- Can represent NNF diagrammatically

Definitions / Diagrams from “A Knowledge Compilation Map”, Darwiche and Marquis. JAIR 02
d-DNNF

- **Decomposable NNF:** sets of leaf vars of conjuncts are disjoint

- **Deterministic NNF:** formula for disjuncts have disjoint models (conjunction is unsatisfiable)

---

Definitions / Diagrams from “A Knowledge Compilation Map”, Darwiche and Marquis. JAIR 02
d-DNNF

- D-DNNF used to compile single formula
  - d-DNNF does not support efficient binary operations ($\lor, \land, \neg$)
  - d-DNNF shares some polytime operations with OBDD / ADD
    - (weighted) model counting (CT) – used in many inference tasks
    - $\rightarrow$ Size(d-DNNF) $\leq$ Size(OBDD) so more efficient on d-DNNF

Children inherit polytime operations of parents

Size of children $\geq$ parents

<table>
<thead>
<tr>
<th>Notation</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO</td>
<td>polytime consistency check</td>
</tr>
<tr>
<td>VA</td>
<td>polytime validity check</td>
</tr>
<tr>
<td>CE</td>
<td>polytime clausal entailment check</td>
</tr>
<tr>
<td>IM</td>
<td>polytime implicant check</td>
</tr>
<tr>
<td>EQ</td>
<td>polytime equivalence check</td>
</tr>
<tr>
<td>SE</td>
<td>polytime sentential entailment check</td>
</tr>
<tr>
<td>CT</td>
<td>polytime model counting</td>
</tr>
<tr>
<td>ME</td>
<td>polytime model enumeration</td>
</tr>
</tbody>
</table>

Table 4: Notations for queries.
Compilations vs Decision Diagrams

• Summary
  – If you can compile problem into **single formula** then compilation is likely preferable to DDs
    • provided you only need ops that compilation supports

  – Not *all* compilations efficient for *all binary* operations
    • e.g., all ops needed for progression / regression approaches
    • fixed ordering of DDs help support these operations

• Note: other compilations (e.g., arithmetic circuits)
  – Great software: http://reasoning.cs.ucla.edu/
Part I Summary: Symbolic Methods for Discrete Inference

• Graphical Models and Influence Diagrams
  – **Representation**: products of discrete factors
  – **Inference**: operations on discrete factors
    • Order of operations matters

• Symbolic Inference with Decision Diagrams (DDs)
  – DDs *more compact than tables or trees*
    • Logical (AND, OR, XOR)
    • Context-specific independence (CSI) & shared substructure
    • Additive and multiplicative structure (Affine ADD)
  – DD **operations exploit structure**
  – DDs support **efficient bounded approximations**

Gogate and Domingos (UAI-13) have recent approx. prob. inference contributions and a good reference list.