Dagstuhl 2018 Tutorial

Decision Diagrams for Planning and Optimization

Scott Sanner

UNIVERSITY OF TORONTO
Factored State Representations

- Natural state representations in planning

- State is inherently factored
  - Room location: \( R = \{1, 2, 3, 4, 5, 6\} \)
  - Door status: \( D_i = \{\text{closed, open}\}; \ i=1..7 \)

- Relational fluents, e.g., \( \text{At}(r_1, 6) \), (STRIPS) are ground variable templates: \( \text{at-r1-6} \)
Big Picture

• Planning states are usually represented in a propositionally factored form
  - (R=1, D₁=o, D₂=c, ..., D₇=o)

• How can decision diagrams speed up planning algorithms seen previously?

DDs: Friends don’t let friends enumerate states!
Using Factored State in Planning

• Classical planning
  – State given by variable assignments
    • \((R=1, D_1=o, D_2=c, \ldots, D_7=o)\)
  – Optimal planning needs to represent \(B^n \rightarrow B / Z / R\)
    – Useful to represent **properties of sets of states**
      » Reachable \((\rightarrow B)\)
      » Reachable with cost \((\rightarrow Z / R)\)
  – **Compute** forward and backward reachability, e.g.
    \[
    FR(x_1', x_2') = \exists x_1 \exists x_2 \ T(x_1'| x_1, x_2) \land T(x_2'| x_2) \land S(x_1, x_2)
    \]
  – See work by Torralba *et al* “Efficient Symbolic Search for Cost-Optimal Planning”, AIJ 2017
Using Factored State in Planning

- Non-deterministic planning
- Probabilistic planning

- Solutions often need to represent:
  - Policies: $B^n \rightarrow Z$ (action ids $\rightarrow Z$)
  - Value functions: $B^n \rightarrow R$

- And compute updates on these functions
  - E.g., Bellman backup

$$V(x_1, x_2) = R(x_1, x_2) + \sum_{x_1'} \sum_{x_2'} P(x_1'| x_1, x_2) P(x_2'| x_2) V'(x_1', x_2')$$

- SPUDD (Hoey, Boutilier, et al, 1999)
Why DDs for Planning?

• **Reason 1: Space considerations**
  – $V(\text{Door-1-open, \ldots, Door-40-open})$ requires
    $\sim 1$ terabyte if all states enumerated

• **Reason 2: Time considerations**
  – With 1 gigaflop/s. computing power, binary operation on above function requires $\sim 1000$ seconds
Function Representation (Tables)

• How to represent functions: \( B^n \rightarrow R \)?

• How about a fully enumerated table…

• Can we do better?

<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>

Think of BDDs as \(\{0,1\}\) subset of ADD range
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>

Think of BDDs as \{0,1\} subset of ADD range
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

• Planning needs $B^n \rightarrow B \ / \ Z \ / \ R$
  – 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)
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
    • Only one way to represent function!
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

Original var labels
$$x_1 \cdot x_2 + x_3 \cdot x_4 + x_5 \cdot x_6$$

Vars relabeled
$$x_1 \cdot x_3 + x_2 \cdot x_5 + x_3 \cdot x_6$$

Left = low. Right = high

Graph-Based Algorithms for Boolean Function Manipulation
Dynamic Variable Reordering

• Rudell’s sifting algorithm
  – Global reordering as pairwise swapping
  – Only need to redirect arcs
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 recursive cases

Need to handle base cases
Apply Tricks

• 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

  $$x_1 \oplus (x_2 \oplus (4 \otimes x_3)) \otimes (x_4)$$

• In general:

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

\[
x_1 + (x_2 + 4x_3) \times (x_4)
\]
\[
\rightarrow x_1 \oplus (x_2 \oplus (4 \otimes x_3)) \otimes (x_4)
\]
ZDDs
(zero-suppressed BDDs)

Represent sets of subsets
ZDDs for Sets of Subsets

• Example BDD and ZDD

Figure 2. The BDD and the ZDD for the set of subsets \({\{a,b}\}, \{a,c\}, \{c\}\).

An Introduction to Zero-Suppressed Binary Decision Diagrams
Alan Mishchenko
ZDDs vs. BDDs

- But ZDDs not universal replacement for BDDs...

Figure 1. BDD and ZDD for $F = ab + cd$.  

An Introduction to Zero-Suppressed Binary Decision Diagrams
Alan Mishchenko
How to Modify Apply for ZDDs?

• Simple
  – $F_x$ is sub-ZDD for set with $x$
  – $F_{\setminus x}$ is sub-ZDD for set without $x$

• $F \cap G$:
  – if (x in set)
    • then $F_x \cap G_x$
    • else $F_{\setminus x} \cap G_{\setminus x}$

• This is just standard Apply
  – With properly defined GetNode, leaf ops: $\cap = \land$, $\cup = \lor$
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):**

  ![Diagram](image)

- **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)

*Need top-level affine transform to recover original range*
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
  
  • \( V(a,b) = V(a) \cdot V(b) \)
  
  \[ = \gamma^{(2a + b)}; \quad \gamma < 1 \]
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!
Other DDs
Multivalued (MV-)DD

- A DD with multivalued variables
  - straightforward k-branch extension
  - e.g., k=6

![Diagram of Multivalued DD]

Obvious generalizations to Apply and Reduce
Multi-terminal (MT-)BDD

- Imagine terminal is 3 bits… use 3 BDDs

- MT-BDD – combine into single diagram
  - Same as ADD using bit vector (integer) leaves
(F)EV-BDDs

- **EdgeValue-BDD** is like AADD where only additive constant subtracted
  - Not a full affine transform
  - **Better numerical precision properties than AADD**
    - Additive, but no multiplicative compression like AADD

- **Factor-EVBDD** is for integer leaves $\mathbb{Z}$
  - Instead of dividing by range…
    - factors out largest prime factor as multiplier
Further Afield

- **K*DDs, BMDs, K*BMDs**
  - Like ZDD, different ways to do decomposition
  - Mainly used in digital verification literature

- **FODDs, FOADDs**
  - Support first-order logical decision tests
    (Wang, Joshi, Khardon, JAIR-08)
    (Sanner, Boutilier, AIJ-09)

- **XADDs: continuous variables →**
  (Sanner, UAI-11)
Teaser: XADD Maximization

\[ \text{max}(y > 0, x > 0) = \]

May introduce new decision tests
Solving Continuous MDPs with XADDs!

\[ V^0(x) \]

\[ V^1(x) \]

\[ V^2(x) \]

\[ x \leq 2 \]

\[ x \geq -2 \]

\[ x \leq 12 \]

\[ 4 - x \times x \ (y = 0) \]

\[ x \leq 8 \]

\[ 0 \ (y = 0) \]

\[ x \geq -8 \]

\[ x \leq 10 \]

\[ -96 + 20 \times x \times x \times x \ (y = -10) \]

\[ 4 \ (y = -x) \]

\[ x \geq -10 \]

\[ -96 + 20 \times x \times x \times x \ (y = 10) \]

Reward

Value (Policy)
Approximation

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:

- Error is bounded!

(St-Aubin, Hoey, Boutilier, NIPS-00)
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

For operations, see “interval arithmetic”:
http://en.wikipedia.org/wiki/Interval_arithmetic
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 ≈ nodes from bottom up:
Decision Diagram Software

Work with decision diagrams in < 1 hour!
Software Packages

• CUDD
  – BDD / ADD / ZDD
  – http://vlsi.colorado.edu/~fabio/CUDD/
  – Hands down, the best package available

• JavaBDD (native interface to CUDD / others):
  – http://javabdd.sourceforge.net/

• NuSMV – Model Based Planner (MBP)
  – http://mbp.fbk.eu/

• SPUDD – ADD-based value iteration for MDPs
  – http://www.computing.dundee.ac.uk/staff/jessehoey/spudd/index.html

• Symbolic Perseus – Matlab / Java ADD version of value PBVI for POMDPs

• Java BDDs / ADDs / AADDs
  – https://code.google.com/p/dd-inference/
  – Scott’s code, not high performance, but functional
  – Includes Java version of SPUDD factored MDP solver & variable elimination
Example Applications 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>Time</td>
<td># Nodes</td>
</tr>
<tr>
<td>Alarm</td>
<td>1,192</td>
<td>2.97s</td>
<td>689</td>
</tr>
<tr>
<td>Barley</td>
<td>470,294</td>
<td>EML*</td>
<td>139,856</td>
</tr>
<tr>
<td>Carpo</td>
<td>636</td>
<td>0.58s</td>
<td>955</td>
</tr>
<tr>
<td>Hailfinder</td>
<td>9,045</td>
<td>26.4s</td>
<td>4,511</td>
</tr>
<tr>
<td>Insurance</td>
<td>2,104</td>
<td>278s</td>
<td>1,596</td>
</tr>
<tr>
<td>Noisy-Or-15</td>
<td>65,566</td>
<td>27.5s</td>
<td>125,356</td>
</tr>
<tr>
<td>Noisy-Max-15</td>
<td>131,102</td>
<td>33.4s</td>
<td>202,148</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...

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

- APRICODD (ADD)
- MADCAP (AADD)
Traffic Domain

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

- **Actions**
  - Light changes

- **Objective:**
  - Maximize empty cells in network
Results Traffic

- **Time (s)**: Graph showing time in seconds for different scenarios.
  - 20 Vars, Exact
  - 20 Vars, Approx (10%)
  - 24 Vars, Exact
  - 24 Vars, Approx (10%)

- **Space (# Nodes)**: Graph showing space (number of nodes) for different 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></th>
<th>ADD</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Management</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)
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
    - $\rightarrow \text{Size}(d\text{-DNNF}) \leq \text{Size}(\text{OBDD})$ so more efficient on d-DNNF
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)
DD Recap: What’s the Big Deal?

• More than compactness
  – Ordered decision tests in DDs support efficient operations
  • ADD: -f, f ⊕ g, f ⊗ g, max(f, g)
  • BDD: ¬f, f ∧ g, f ∨ g
  • ZDD: f \ g, f ∩ g, f ∪ g

– Useful operations for planning & optimization!
DDs: Friends don’t let friends enumerate states!