Affine Algebraic Decision Diagrams (AADDs)

and their Application to Structured Probabilistic Inference

Scott Sanner (Toronto)

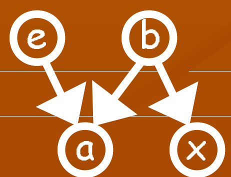
David McAllester (TTI-C)

Talk Outline

- Data structures for representing $B^n \rightarrow R$
  - Why is efficiency for these functions important?
  - Compact forms for logical, additive, & mult. structure
  - Efficient operations: $+$, $\cdot$, $max(F)$, $\oplus$, $\otimes$, $max(F_1,F_2)$
  - Practical considerations (numerical precision issues)
- Applications to probabilistic inference
  - Variable elimination in Bayes nets
  - Value iteration in MDPs
- Conclusions and future work

Motivations I

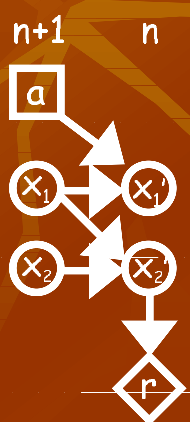
- Why do we need functions from $B^n \rightarrow R$?
- They are ubiquitous in uncertain reasoning, e.g.
  - Inference in Bayesian networks:



    - Conditional prob. tables: P(Alarm | Earthquake, Burglar)
    - Variable elimination:

$$\sum_{x1\ldots xi} \prod_{F1\ldots Fj} F_1(x_{1\ldots}x_i) \ldots F_j(x_{1\ldots}x_i)$$

  - Solving Markov decision processes (MDPs):



    - Value and reward functions: V(Box-1-delivered, … , Box-n-delivered)
    - Value iteration:

$$V^{n+1}(x_1\ldots x_i) = R(x_1\ldots x_i) + \gamma \cdot max_a \sum_{x1'\ldots xi'} \prod_{F1\ldots Fi} P_1(x_1'|_{\ldots}a) \ldots Pi(x_i'|_{\ldots}a)\, V^n(x_1'\ldots x_i')$$

Motivations II

- For $\mathsf{B}^n \to \mathsf{R}$, why do we need:
  - Compact representations?
  - Efficient operations: +, ·, max(F), ⊕, ⊗, max(F_1,F_2)?
- Reason 1: Space considerations
  - V(Box-1-delivered, … , Box-40-delivered) would require ~1 terabyte if all states were enumerated
- Reason 2: Time considerations
  - With 1 gigaflop/s. computing power, one binary operation on the above function would require ~1000 seconds
  - Even simple MDP/BN inference problems can require 100+ binary operator applications!

Function Representation (Tables)

- How do we represent a function from $B^n \rightarrow R$?
- How about a fully enumerated table…
- …OK, but can we be more compact?

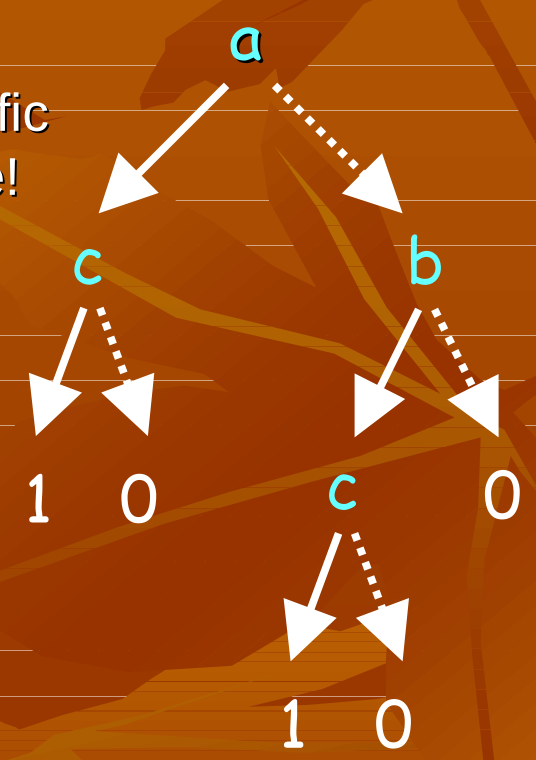
a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

Function Representation (Trees)

- How about a tree? Sure, now can simplify.

a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

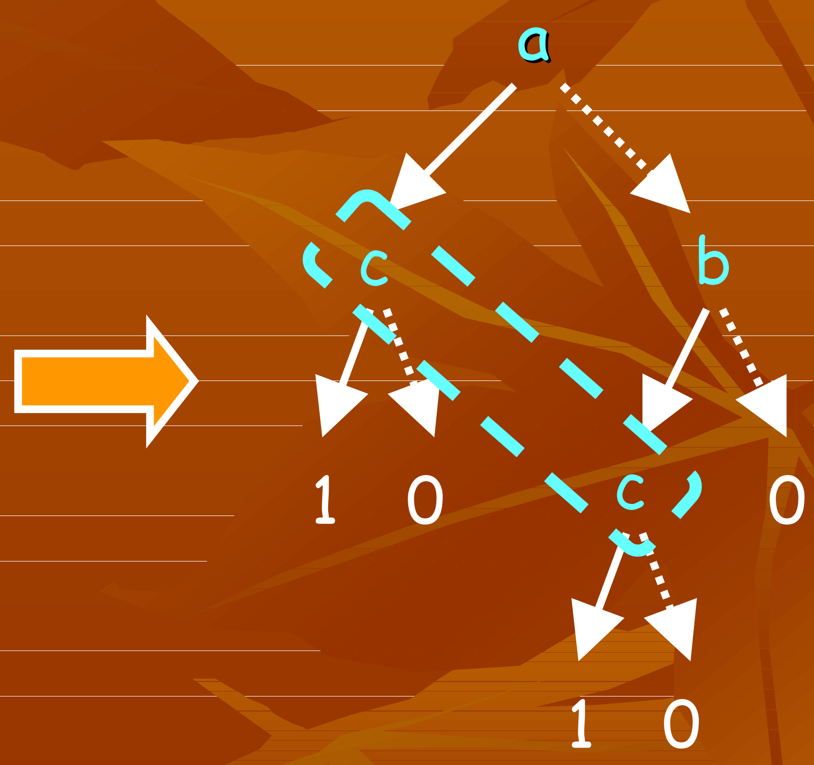
Context-specific
independence!



Function Representation (ADDs)

- Why not a directed acyclic graph (DAG)?

a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

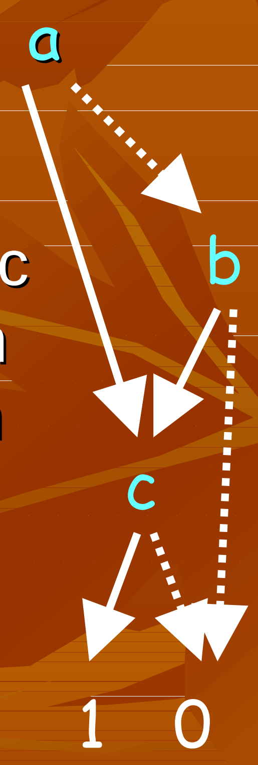


Function Representation (ADDs)

- Why not a directed acyclic graph (DAG)?

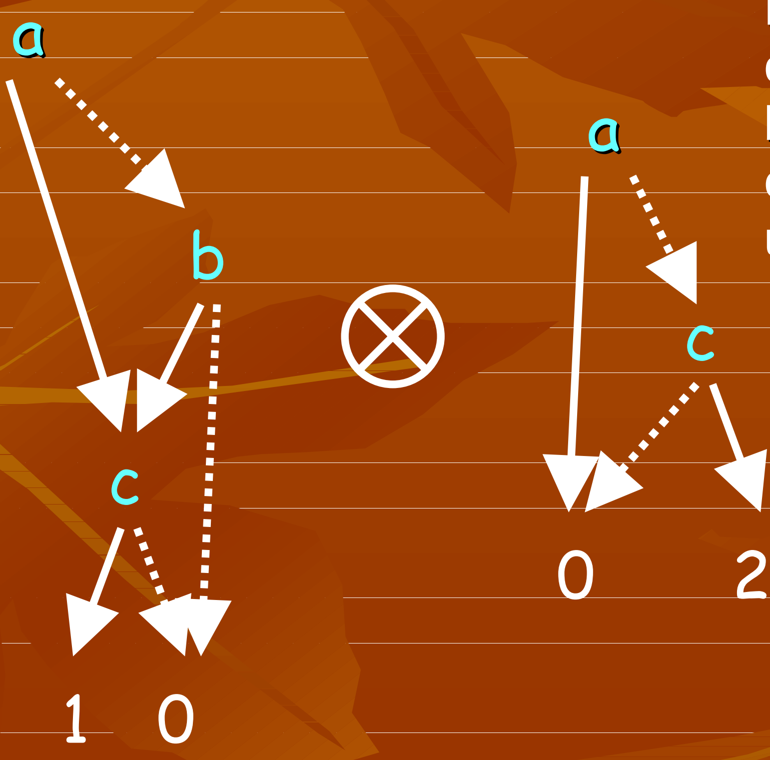
a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

Algebraic Decision Diagram (ADD)



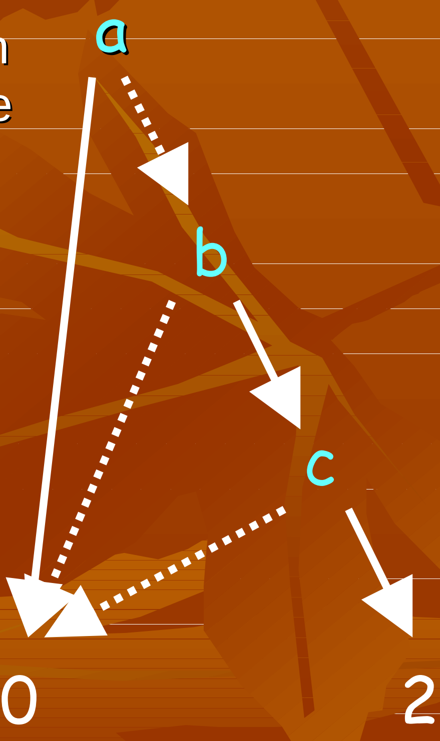
Binary Operations (ADDs)

- Why do we order the variables?
- This enables us to do efficient binary operations…



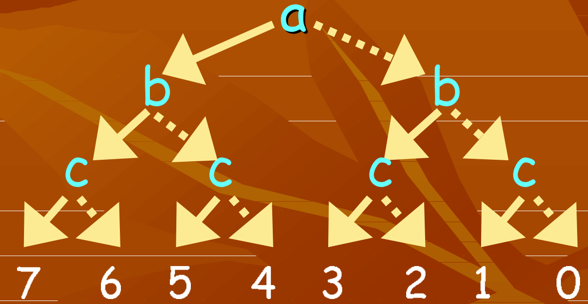
Result: ADD operations can be **much** more efficient than using tables

=

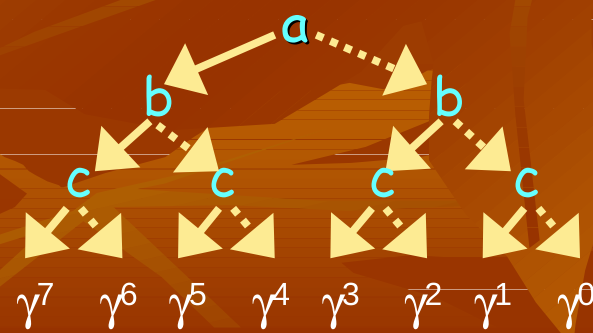


ADD Inefficiency

- Is context-specific independence enough?
- Or do we need more compactness?
- Example 1: Additive reward/utility functions
  - $R(a,b,c) = R(a) + R(b) + R(c)$
    $= 4a + 2b + c$

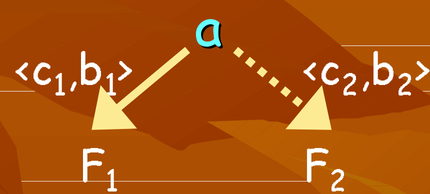


- Example 2: Multiplicative value functions
  - $V(a,b,c) = V(a) \cdot V(b) \cdot V(c)$
    $= \gamma^{(4a + 2b + c)}$

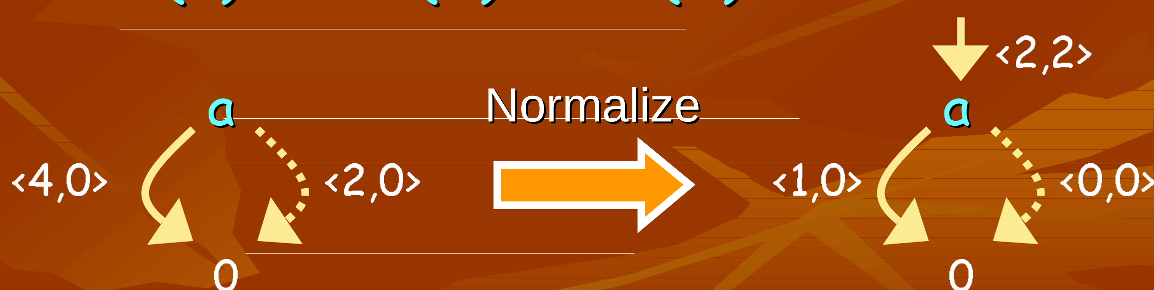


Affine ADD (AADD)

- Let's define a new decision diagram – affine ADD
- Edges labeled by an offset (c) and multiplier (b):

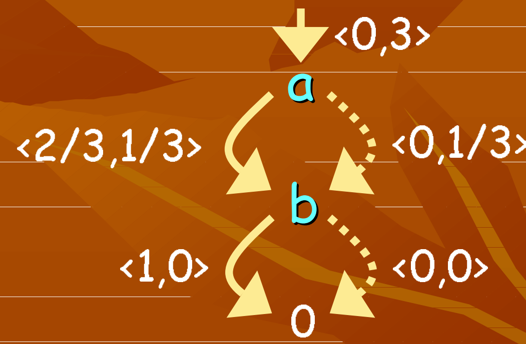


- General semantics: if (a) then ($c_1+b_1F_1$) else ($c_2+b_2F_2$)
- To maximize sharing, normalize all nodes [0,1]
- Example: if (a) then (4) else (2)

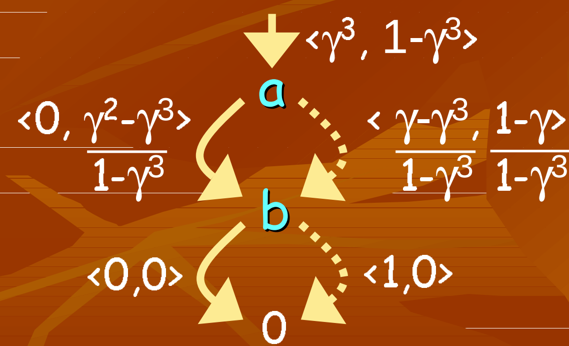


AADD Examples

- Back to our previous examples…
- Example 1: Additive reward/utility functions
  - R(a,b) = R(a) + R(b)
    = 2a + b



- Example 2: Multiplicative value functions
  - $V(a,b) = V(a) \cdot V(b)$
    $= \gamma^{(2a+b)};\ \gamma<1$



AADD Operations

- How do we perform operations on AADDs?
- Let's look at a simple addition example:

- Two recursive cases:
  - If a is true: $(0+1F_1) \oplus (0+2F_2) = (c_{1R} + b_{1R}F_{1R})$
  - If a is false: $(3+2F_1) \oplus (7+4F_2) = (c_{2R} + b_{2R}F_{2R})$
- But, can we avoid the second recursion?
  - By simple algebraic manipulation, we know:

    $(3+2F_1) \oplus (7+4F_2) = 10+2(F_1+2F_2) = 10+2(c_{1R}+b_{1R}F_{1R})$
- Suggests canonical caching scheme…

AADD Algorithms and Caching

- To maximize cache hits for binary operators, must use following canonical caching scheme:

Operator	Compute & Store	Return
$c_1+b_1F_1 \oplus c_2+b_2F_2$	$F_1 \oplus (b_2/b_1)F_2$	$c_1+c_2+b_1 \cdot \text{Result}$
$c_1+b_1F \oplus c_2+b_2F$	nothing	$c_1+c_2+(b_1+b_2)F$
$c_1+b_1F_1 \otimes c_2+b_2F_2$	$(c_1/b_1)+F_1 \otimes (c_2/b_2)+F_2$	$b_1b_2 \cdot \text{Result}$
$\max(c_1+b_1F_1, c_2+b_2F_2)$	$\max(F_1, ((c_2-c_1)/b_1) +(b_2/b_2)F_2)$	$c_1+b_1 \cdot \text{Result}$
$\max(c_1+b_1F, c_2+b_2F)$ (see return conditions)	nothing	$c_1 \geq c_2 \wedge b_1 \geq b_2$: c_1+b_1F $c_2 \geq c_1 \wedge b_2 \geq b_1$: c_2+b_2F

- Otherwise, operations similar to those for ADDs:
  - Just propagate affine transform on recursion
  - And normalize and cache results on return

AADD Theorems

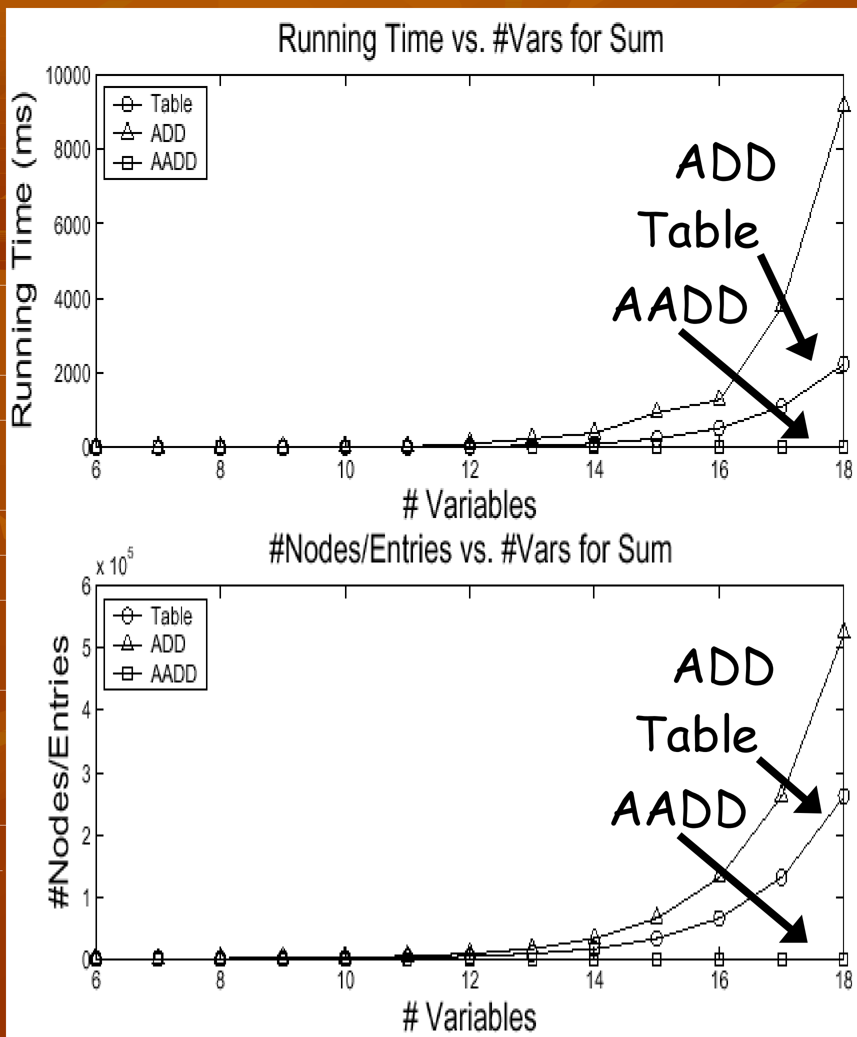
- How important is canonical caching?
  - Without it, AADD operations provably same as ADD!
  - With it, AADD can yield an exponential time/space improvement over ADD – and never performs worse!
- Stated more formally:
  - Theorem 1:
    - The time and space performance of the Reduce and Apply operations for AADDs is within a multiplicative constant of that of ADDs in the worst case.
  - Theorem 2:
    - There exist functions F_1 and F_2 and an operator 'op' such that the running time and space performance of $\langle F_1 \text{ op } F_2 \rangle$ can be linear in the number of variables (in F_1 and F_2) for the AADD when the corresponding ADD operation must be exponential in the number of variables.
    - E.g. $\langle \sum_{i=1}^{n} 2^i \cdot x_i \oplus \sum_{i=1}^{n} 2^i \cdot x_i \rangle$, $\langle \prod_{i=1}^{n} \gamma\hat{}(2^i \cdot x_i) \otimes \prod_{i=1}^{n} \gamma\hat{}(2^i \cdot x_i) \rangle$

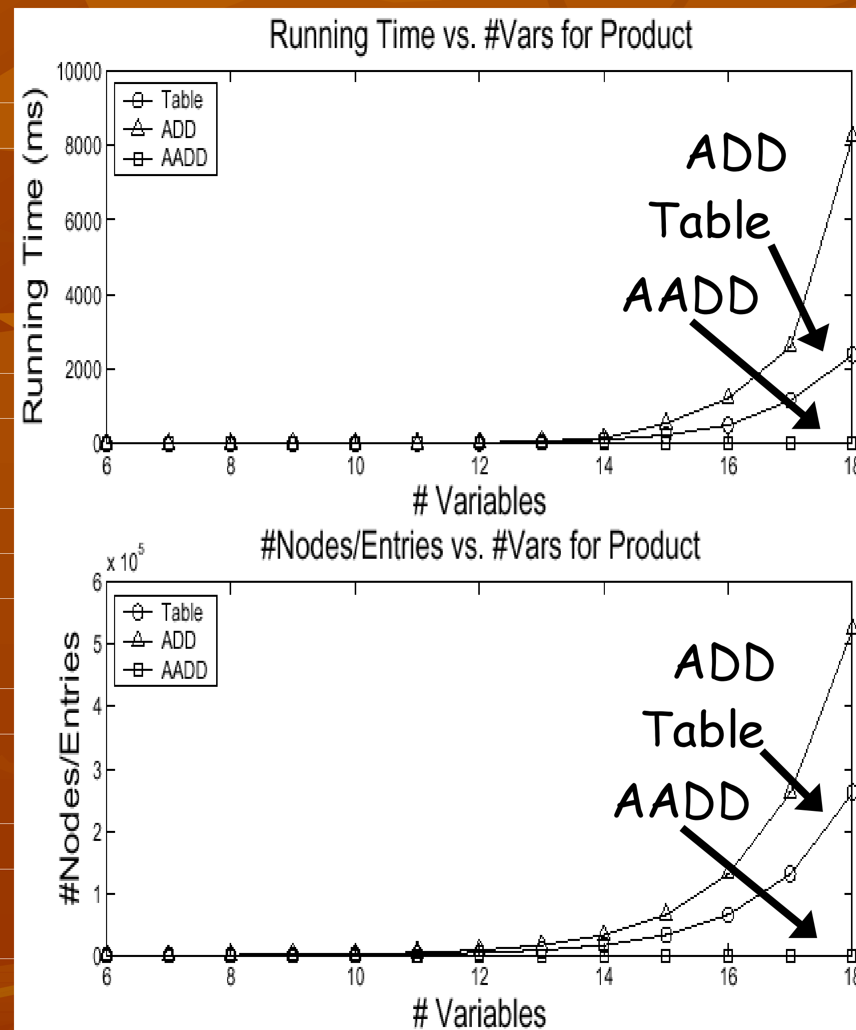
Numerical Precision Issues

- AADDs introduce numerical precision issues:
  - AADDs perform many subtractions and divisions
  - Operations cache has to store keys $\langle b_1,c_1,F_1,b_2,c_2,F_2,op \rangle$ where b_1, c_1, b_2, and c_2 are floating point values
  - If cache keys stored exactly $\Rightarrow$ explosion of distinct nodes equivalent within numerical precision!
- Solution: Implement "nearest neighbor" cache lookup
  - Hash on node distance from origin
  - Truncate distance at desired precision
  - $\Rightarrow$ Nodes equivalent within precision will hash to same bucket (with high probability), test equality within precision
- Works efficiently and effectively in practice

Empirical Comparison: Table/ADD/AADD

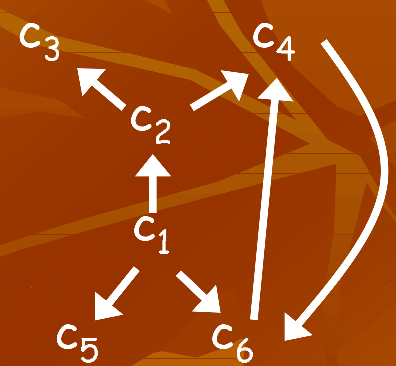
- Sum: $\sum_{i=1}^{n} 2^i \cdot x_i \oplus \sum_{i=1}^{n} 2^i \cdot x_i$
- Prod: $\prod_{i=1}^{n} \gamma\hat{}(2^i \cdot x_i) \otimes \prod_{i=1}^{n} \gamma\hat{}(2^i \cdot x_i)$



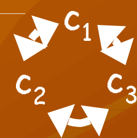
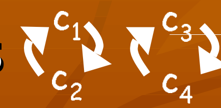


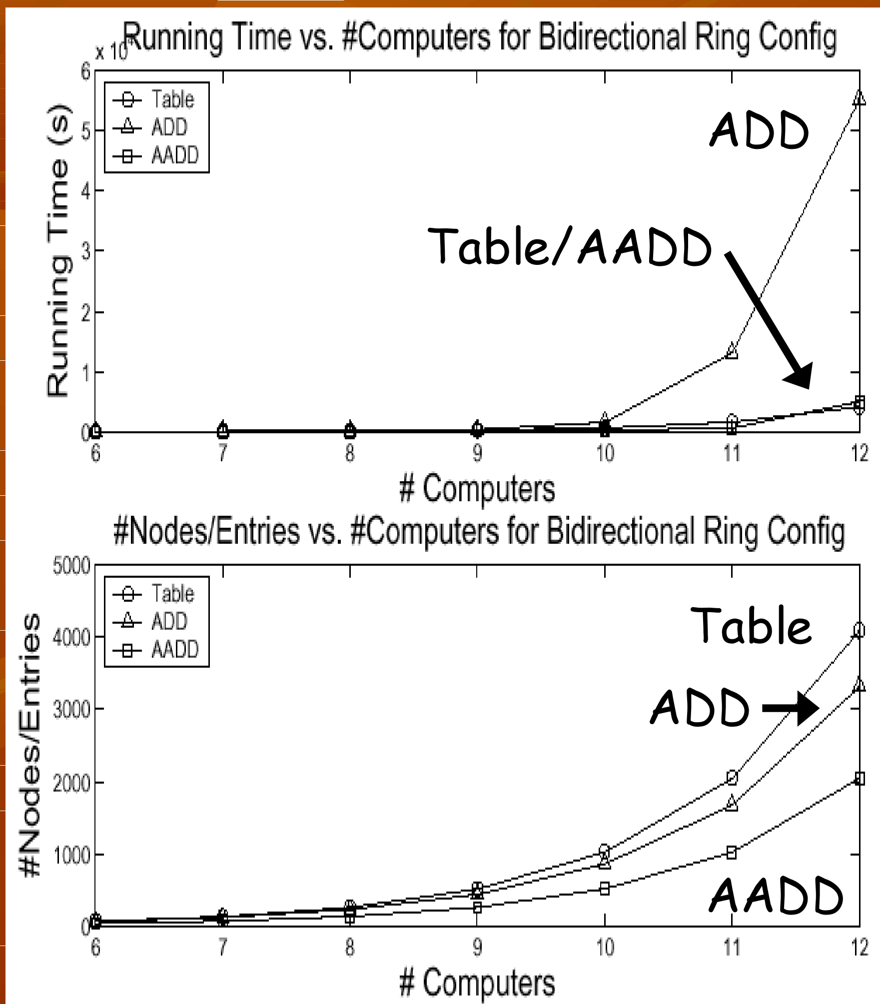
Application: MDP Solving

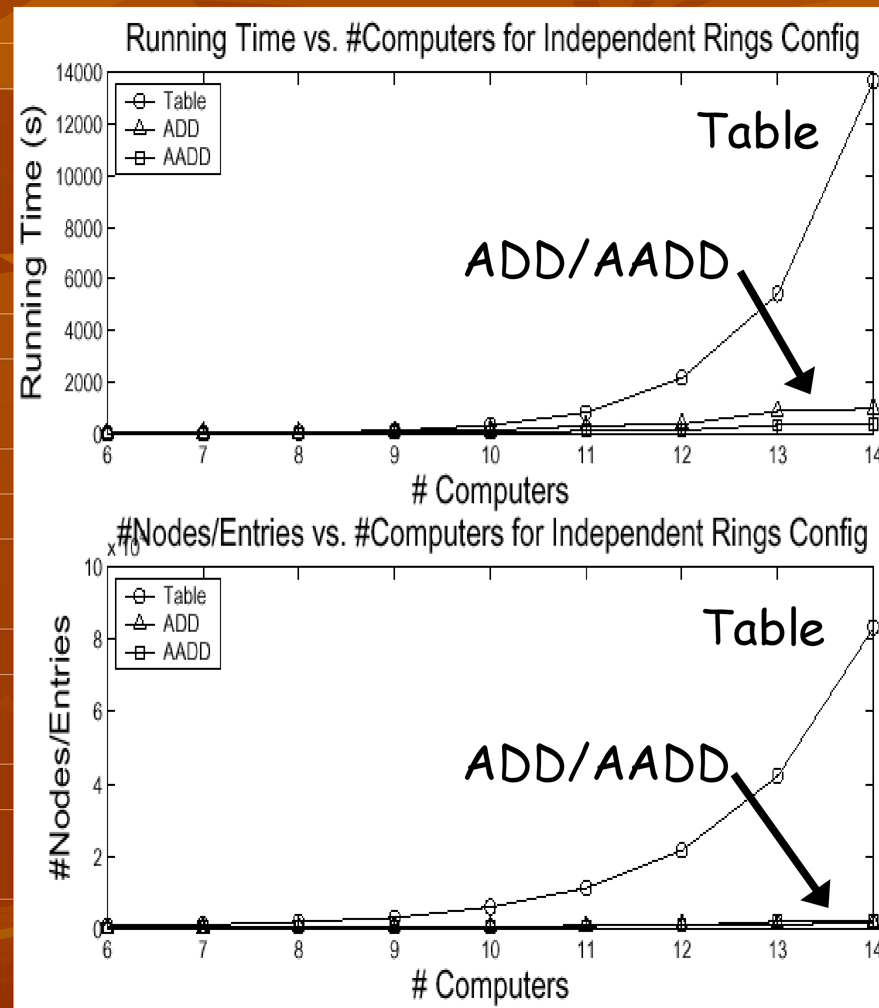
- Extend SPUDD (Hoey, St-Aubin, Hu, Boutilier; 1999)
  - Replace ADD with AADD, same value iteration algorithm
  - $V^{n+1}(x_1 \ldots x_i) = R(x_1 \ldots x_i) +$
    $\gamma \cdot \max_a \sum_{x1' \ldots xi'} \prod_{F1 \ldots Fi} P_1(x_1' |_{\ldots} x_i) \ldots Pi(x_i' |_{\ldots} x_i)\, V^n(x_1' \ldots x_i')$
- SysAdmin MDP (Guestrin, Koller, Parr; 2001)
  - Have a network of computers
  - Various network configurations
  - Every computer is running or crashed
  - At each time step, status is affected by
    - Previous status
    - Status of incoming connections
  - Reward: +1 for every computer running (additive)



MDP Results: SysAdmin

- Bidirectional Ring 
- Independent Rings 





Application: Bayes Net Inference

- Use variable elimination (Zhang & Poole; 1994)
  - Just replace CPTs with AADDs, same algorithm
    - Could do same for clique/junction-tree algorithms
  - $P(query|evidence) = \sum_{x1 \ldots xi} \prod_{F1 \ldots Fj} F_1(x_{1 \ldots} x_i) \ldots F_j(x_{1 \ldots} x_i)$
- Handles CPT structure without specialized inference/transformation
  - Context-specific independence
    - Probability has logical structure: $P(a|b,c) = \text{if } b\ ?\ 1 : \text{if } c\ ?\ .7 : .3$
  - Additive CPTs
    - Probability is discretized linear function: $P(a|b_1 \ldots b_n) = c + b \cdot \sum_i (1/2^i)\, b_i$
  - Multiplicative CPTs
    - Noisy-or represented by multiplicative AADD: $P(e|c_1 \ldots c_n) = 1 - \prod_i (1 - p_i)$
- When CPT or intermediate factor has such a compact form, AADD will find it…

Bayes Net Results: Various Networks

Bayes Net	Table		ADD		AADD	
	# Entries	Time	# Nodes	Time	# Nodes	Time
Alarm	1,192	2.97s	689	2.42s	405	1.26s
Barley	470,294	EML*	139,856	EML*	60,809	207m
Carpo	636	0.58s	955	0.57s	360	0.49s
Hailfinder	9,045	26.4s	4,511	9.6s	2,538	2.7s
Insurance	2,104	278s	1,596	116s	775	37s
Noisy-Or-15	65,566	27.5s	125,356	50.2s	1,066	0.7s
Noisy-Max-15	131,102	33.4s	202,148	42.5s	40,994	5.8s

*EML: Exceeded Memory Limit (1GB)

Related Work

- A lot of related work in formal verification literature:
  - Work in MTBDDs that looks at additive & multiplicative structure (algorithms/caching similar to AADD):
    - *BMDs, K*BMDs, EVBDDs, FEVBDDs, HDDs
  - MTBDDs use finite integer terminals (not floating point)
    - Can still approximate reals via rationals or fixed point values
  - **Problem:** Uncontrollable error when rationals/fixed point representations used for probabilitistic inference
    - Probabilities often below representable min, errors accumulate

- *PHDD is an MTBDD that should work in *theory*:
  - Terminal value represents IEEE floating point representation
  - **Problem:** Would amount to doing floating point arithmetic in software (AADD uses doubles and native arithmetic)

Conclusions and Future Work

- Conclusions:
  - AADDs are a general replacement for Tables/ADDs
  - Compactly represent logical (CSI), additive, and multiplicative structure
  - In theory: never more than a constant times worse than ADDs, potentially exponentially less space/time
  - Empirically on MDP/BN examples: never worse than ADD/Table, sometimes exponentially better
- Future Work:
  - Approximate inference similar to APRICODD (St. Aubin, Hoey, Boutilier, 2000)
  - Other approximation techniques