Approximate Dynamic Programming with Affine ADDs (AADDs)

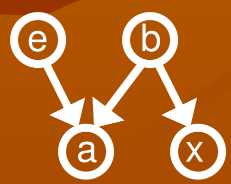
Scott Sanner (NICTA, ANU)
William Uther (NICTA, UNSW)
Karina Valdivia Delgado (USP)

Talk Outline

- Data structures for representing $B^n \rightarrow R$
  - Why important?
  - Compact forms
  - Efficient operations
- AADDs and use in MDPs
- **Contribution:**
  - **How to approximate AADDs efficiently?**
- Results on MDPs

Motivations I

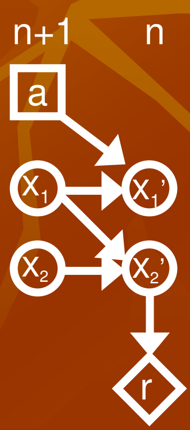
- Why do we need functions from $B^n \to R$?
- Inference in discrete graphical models:



  - Factors, e.g., CPTS: 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 Factored MDPs:



  - Dynamic Bayes Net (DBN)
  - Value and reward functions: $V^0(x_1\ldots x_i) = R(x_1\ldots x_i)$
  - 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 $B^n \rightarrow R$, why do we **need**:
  - Compact representations?
  - Efficient operations: $+$, $\cdot$, max(F), $\oplus$, $\otimes$, max(F_1,F_2)?

- Reason 1: Space considerations
  - V(Box-1-delivered, … , Box-40-delivered) requires ~1 terabyte if all states enumerated

- Reason 2: Time considerations
  - With 1 gigaflop/s. computing power, binary operation on above function requires ~1000 seconds

Function Representation (Tables)

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

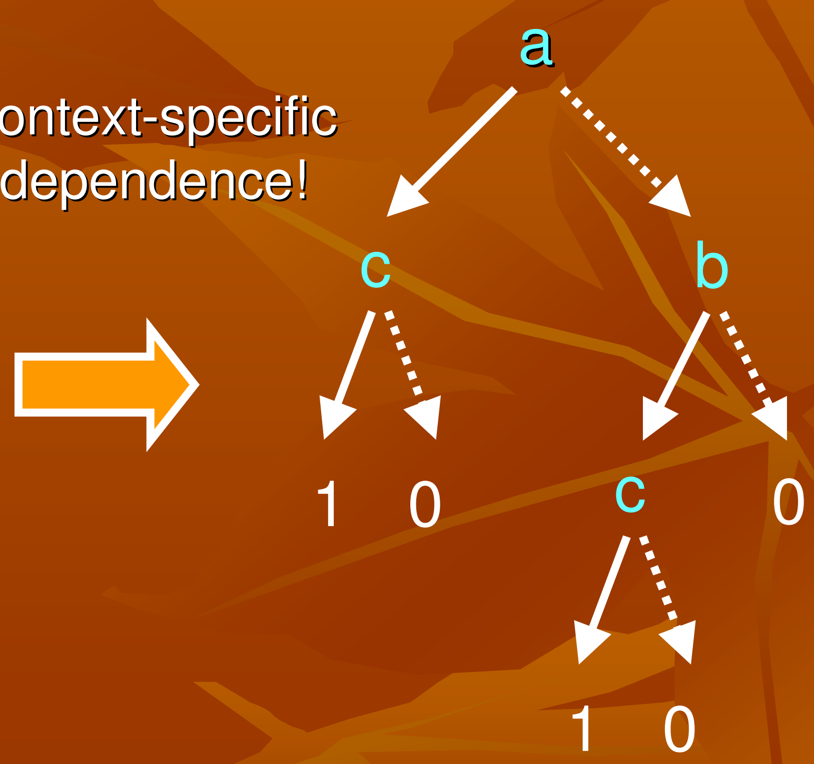
a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

Function Representation (Trees)

- How about a tree? Sure, 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

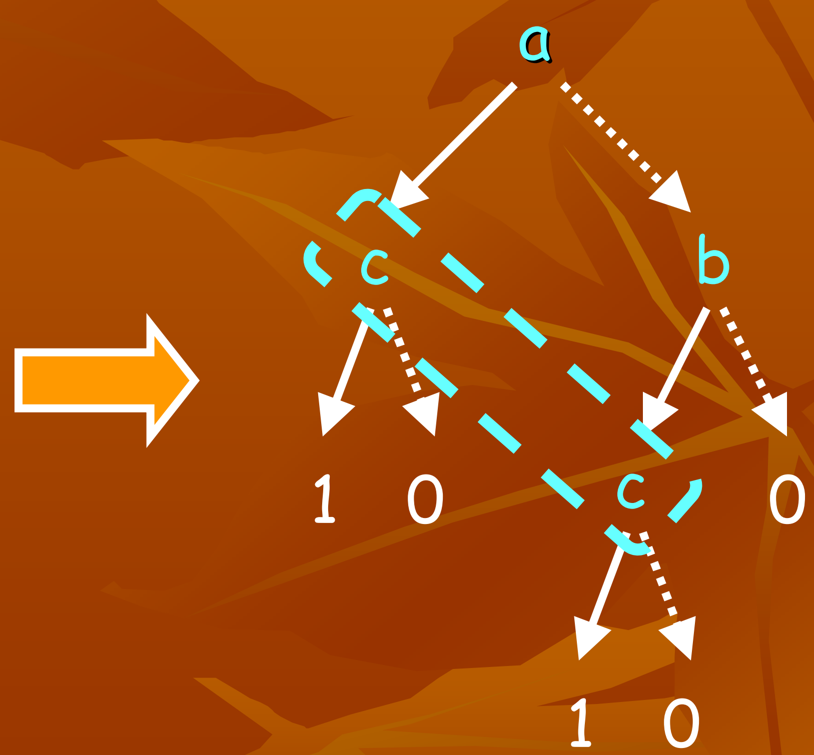
Context-specific independence!



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

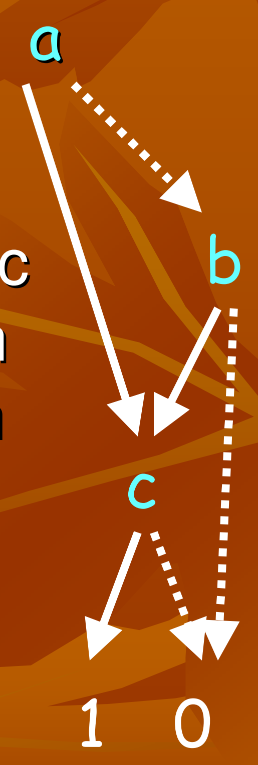


Function Representation (ADDs)

- Why not a directed acyclic graph (DAG)?

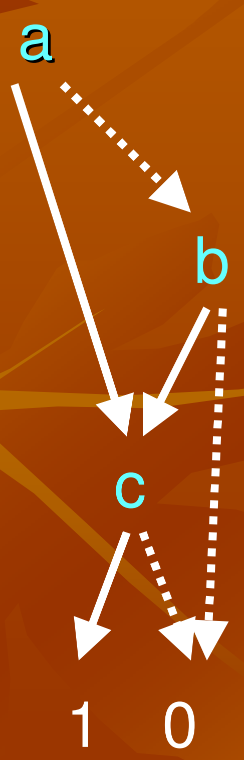
a	b	c	F(a,b,c)
0	0	0	0.00
0	0	1	0.00
0	1	0	0.00
0	1	1	1.00
1	0	0	0.00
1	0	1	1.00
1	1	0	0.00
1	1	1	1.00

Algebraic Decision Diagram (ADD)

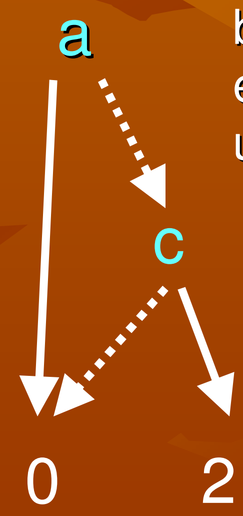


Binary Operations (ADDs)

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

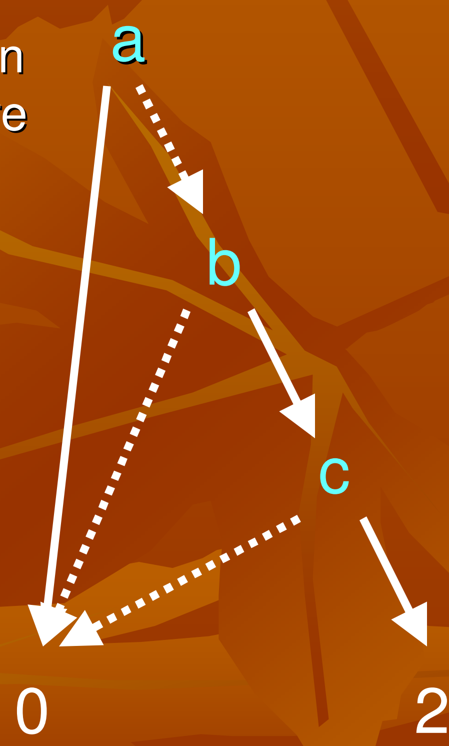


$\otimes$



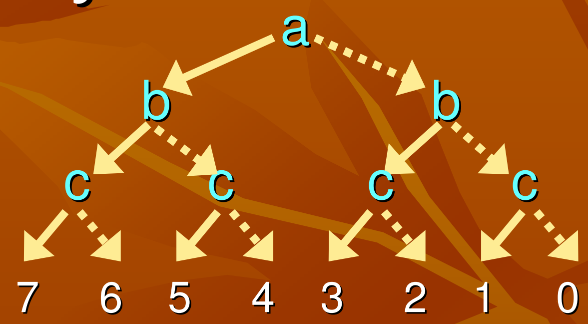
Result: ADD operations can be **much** more efficient than using tables

$=$

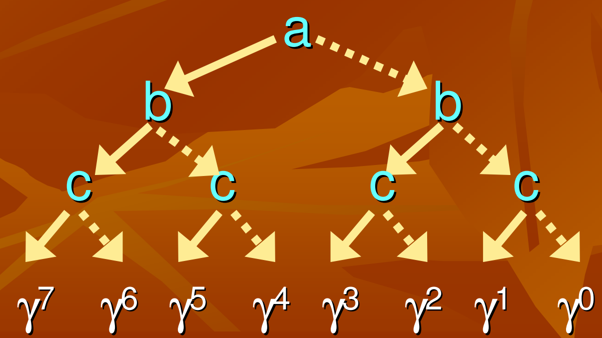


ADD Inefficiency

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

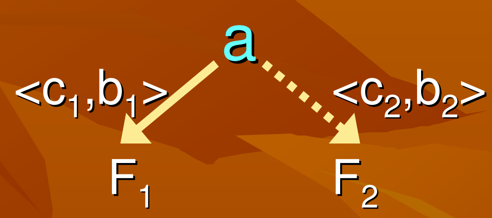


- Ex. 2: Multiplicative value functions
  - $V(a,b,c) = V(a) \cdot V(b) \cdot V(c)$
    $= \gamma^{(4a + 2b + c)}$



Affine ADD (AADD) (SanMcA05)

- Define a new decision diagram – **Affine ADD**
- Edges labeled by **offset (c)** and **multiplier (b)**:

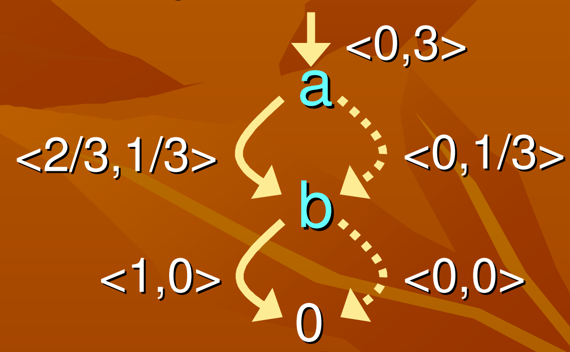


- **Semantics:** if (a) then $(c_1+b_1F_1)$ else $(c_2+b_2F_2)$
- Maximize sharing by **normalizing** nodes [0,1]
- Example: if (a) then (4) else (2)

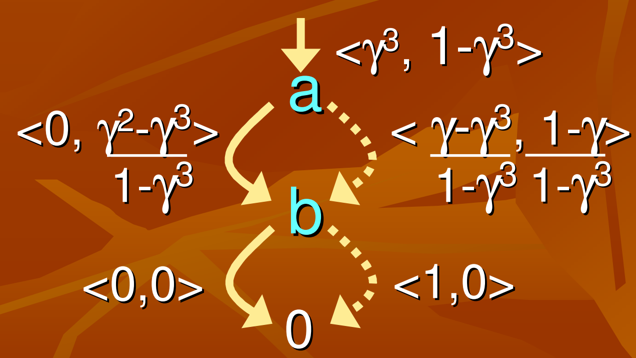
AADD Examples

Automatically Constructed!

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

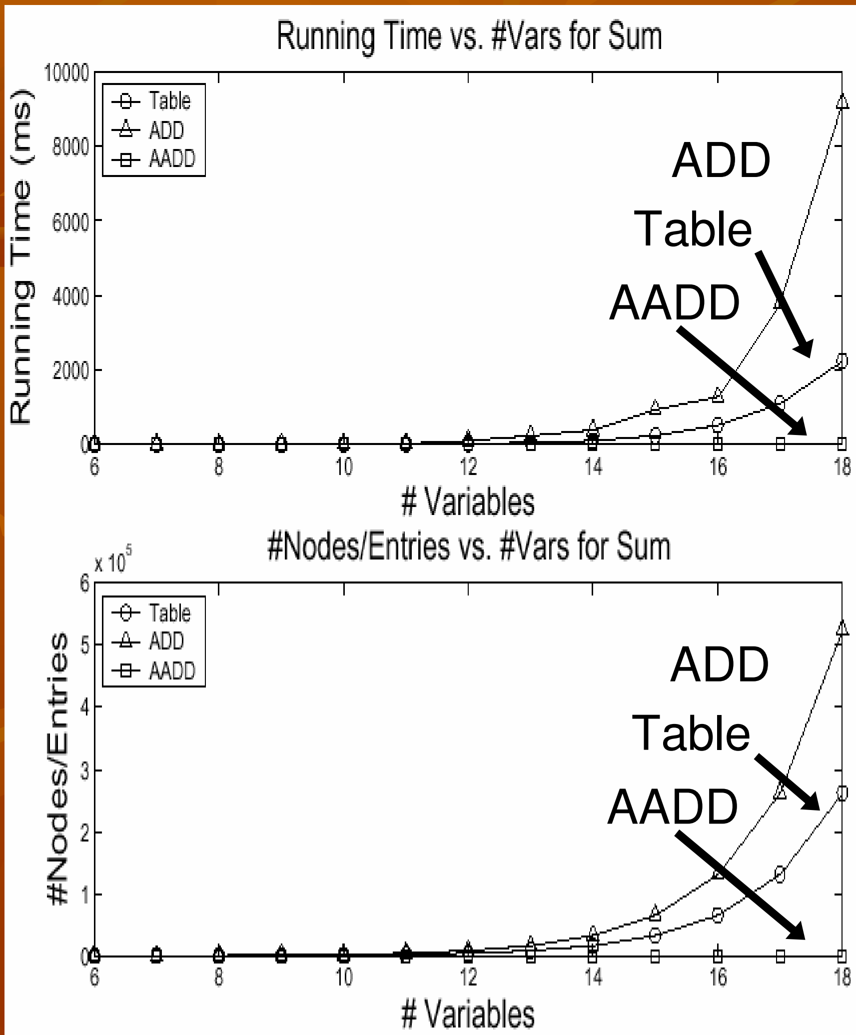


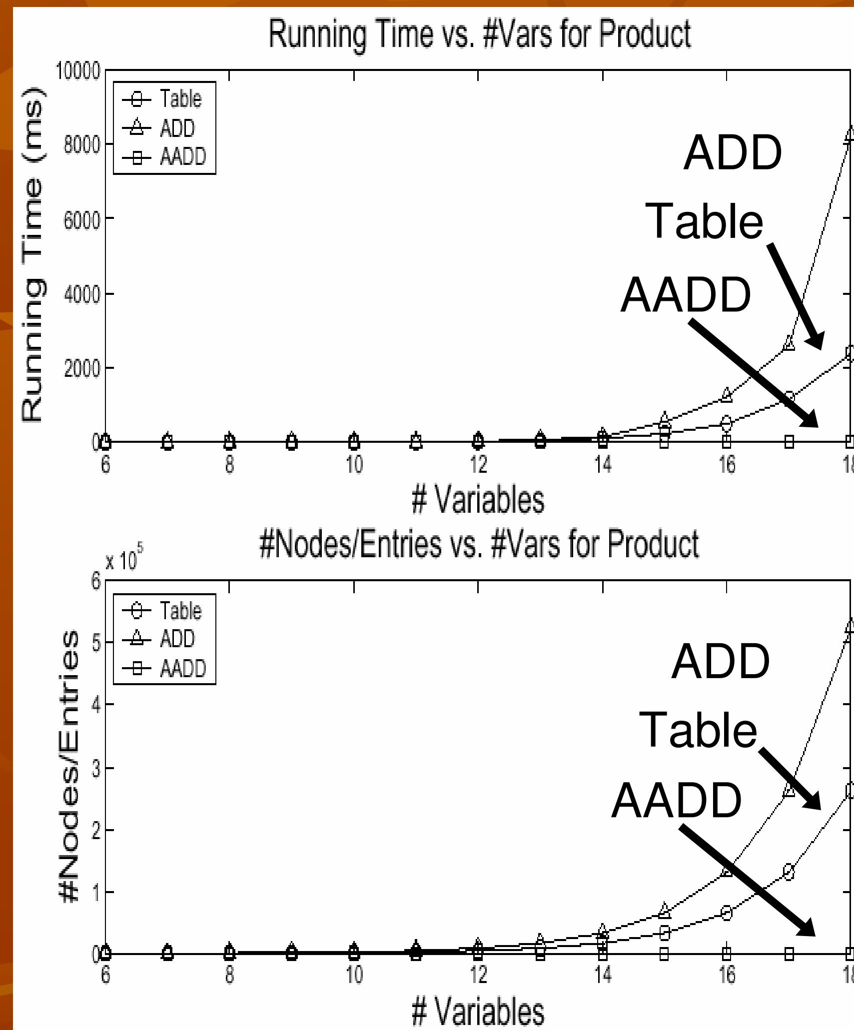
Main AADD Theorem

- AADD can yield **exponential time/space improvement** over ADD
  - and **never performs worse**!

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: $\prod_{i=1}^{n} \gamma\text{^}(2^i \cdot x_i) \otimes \prod_{i=1}^{n} \gamma\text{^}(2^i \cdot x_i)$





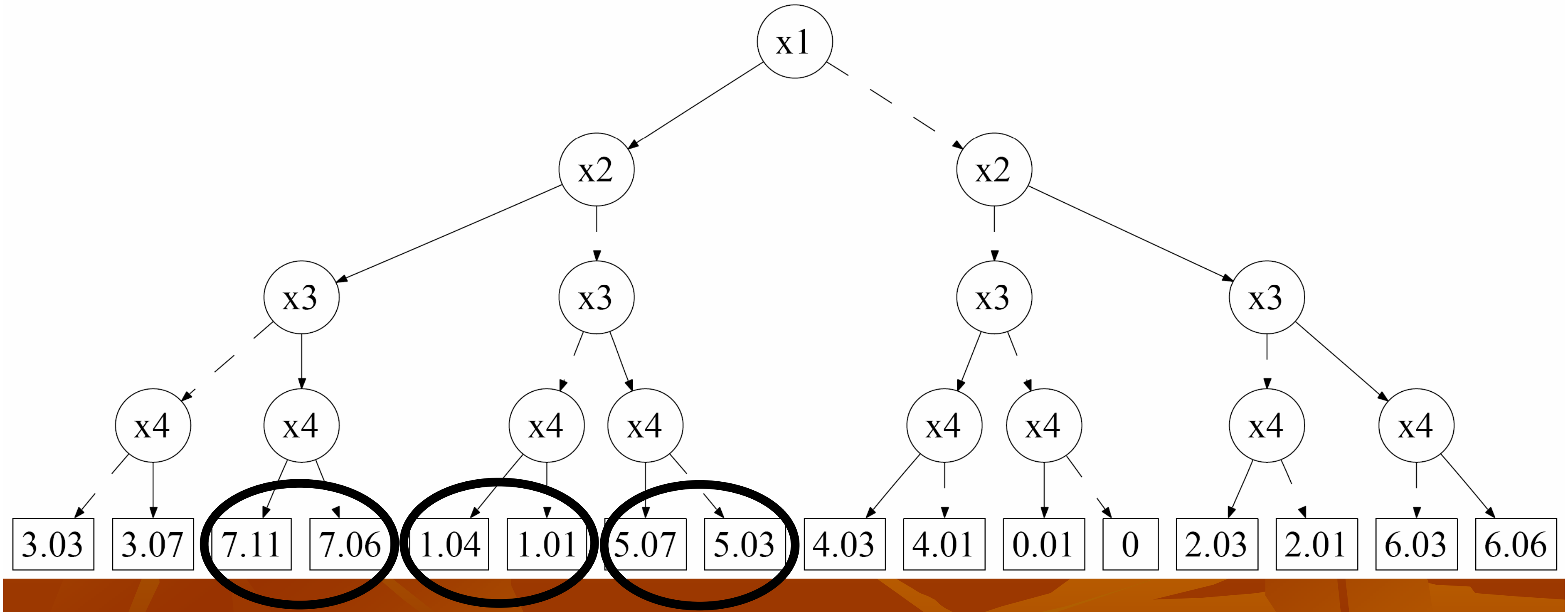
Application: MDP Solving

- Extend SPUDD (HSHB99)
  - Replace ADD with AADD in value iteration algorithm:

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

Problem: Value ADD Too Large

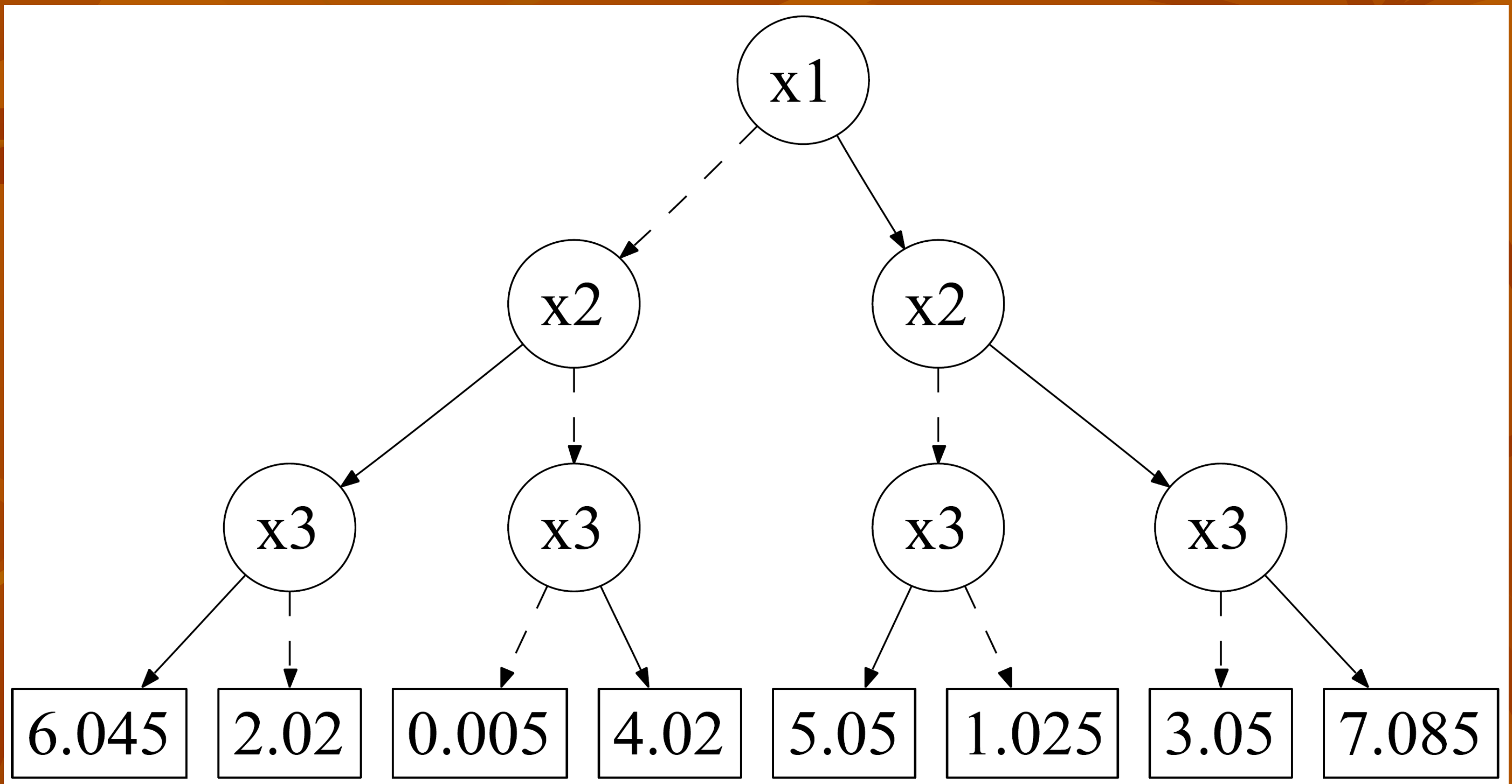
- Sum: $\sum_{i=1} x_i + \varepsilon\text{-}Noise$



- How to approximate?

Solution: APRICODD Trick

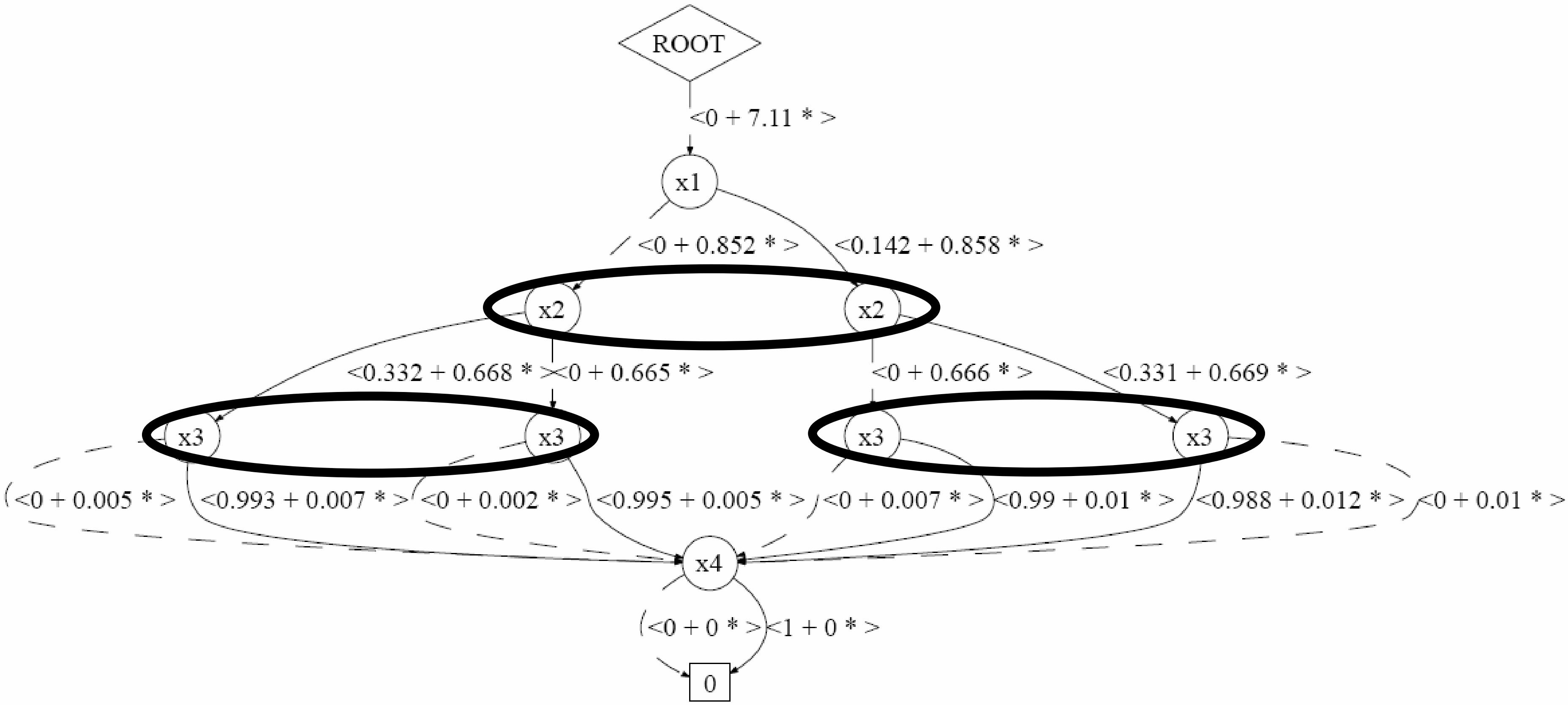
- Merge ≈ leaves and reduce:



- Error is bounded!

More Compactness? AADDs?

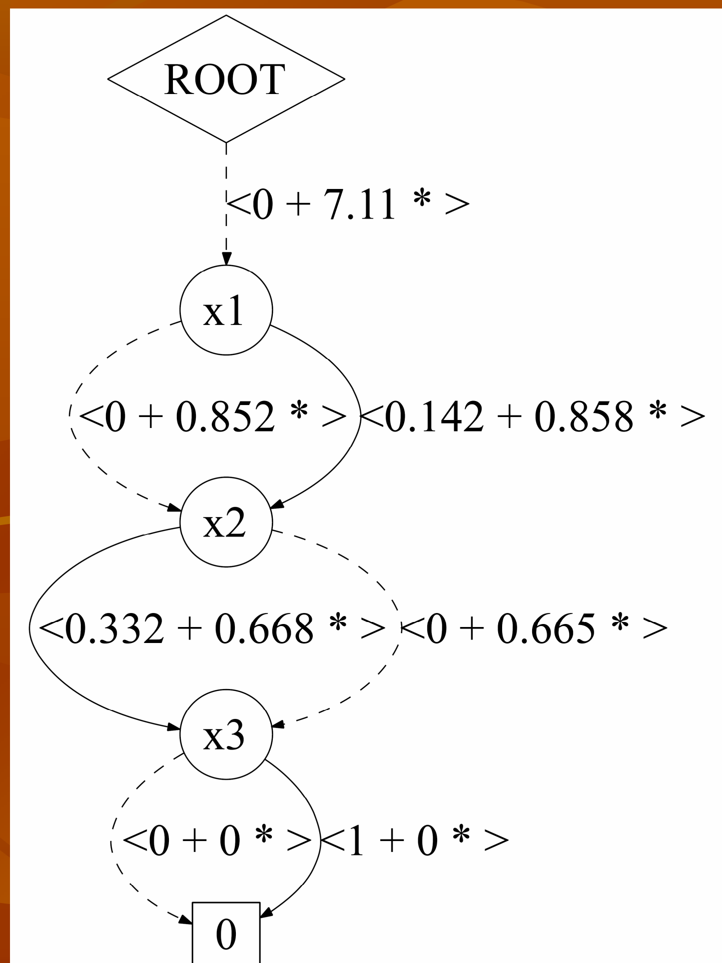
- Sum: $\sum_{i=1} x_i$ + *ε-Noise*



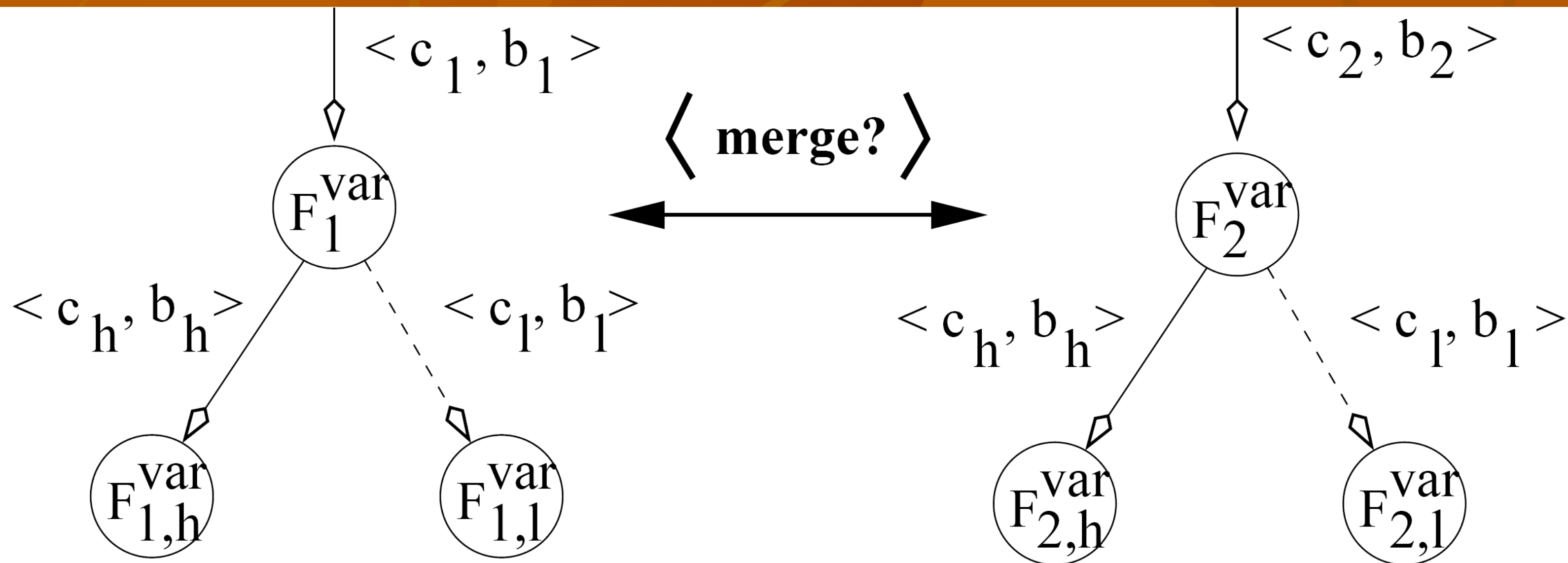
- How to approximate?

Solution: MADCAP Trick

- Merge ≈ nodes from bottom up:



Error Analysis



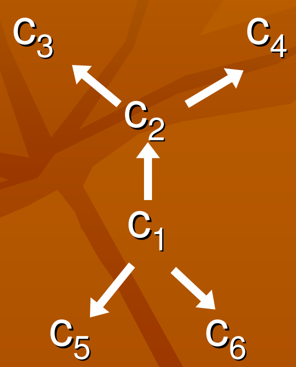
- Error of node merge:

$$error := \max(F_1^{MaxRange}, F_2^{MaxRange})$$
$$\cdot \max(|F_1.c_l - F_2.c_l| + |F_1.b_l - F_2.b_l|,$$
$$|F_1.c_h - F_2.c_h| + |F_1.b_h - F_2.b_h|)$$

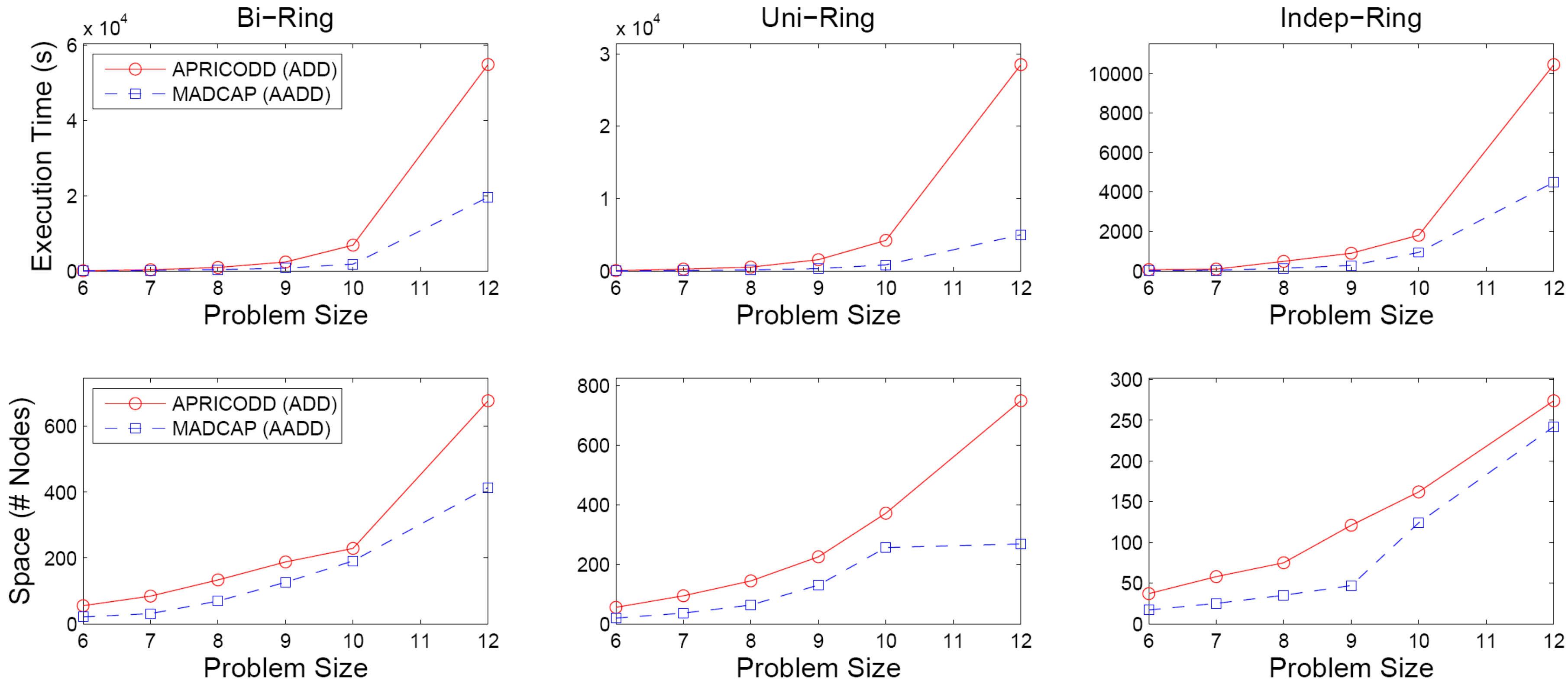
Error if replace one node with other?

Application: SysAdmin

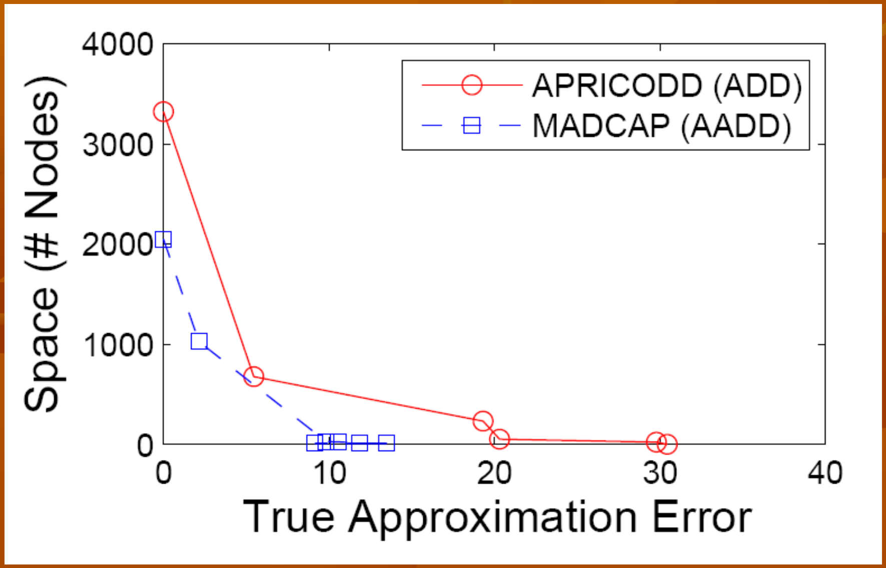
- SysAdmin MDP (GKP, 2001)
  - Network of computers: $c_1, \ldots, c_k$
  - Various network topologies
  - Every computer is running or crashed
  - At each time step, status of c_i affected by
    - Previous state status
    - Status of incoming connections in previous state
  - Reward: +1 for every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

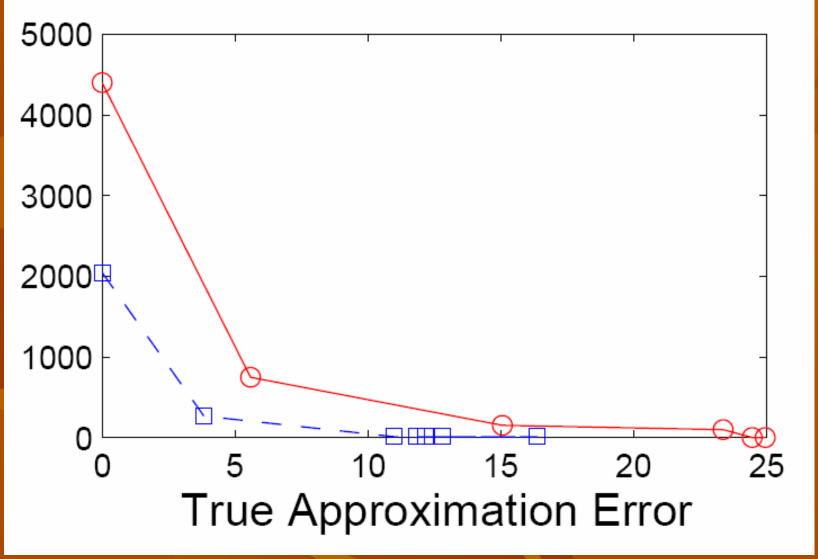


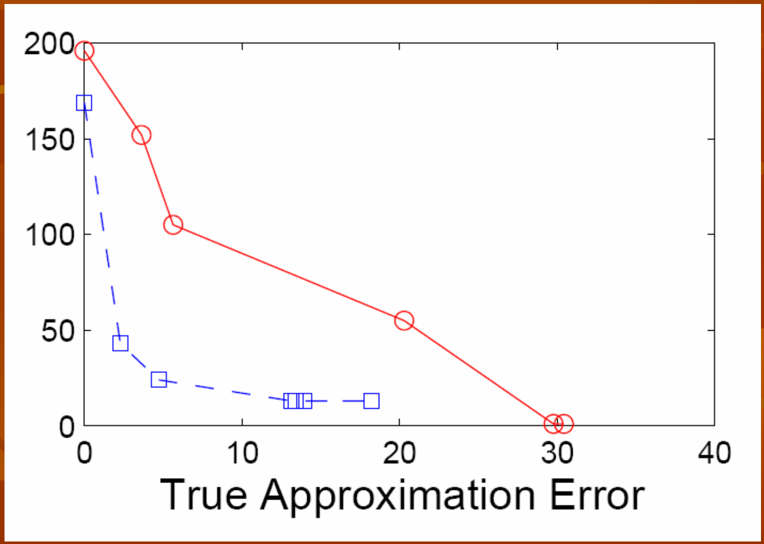
Results I: SysAdmin (10% Approx)



Results II: SysAdmin

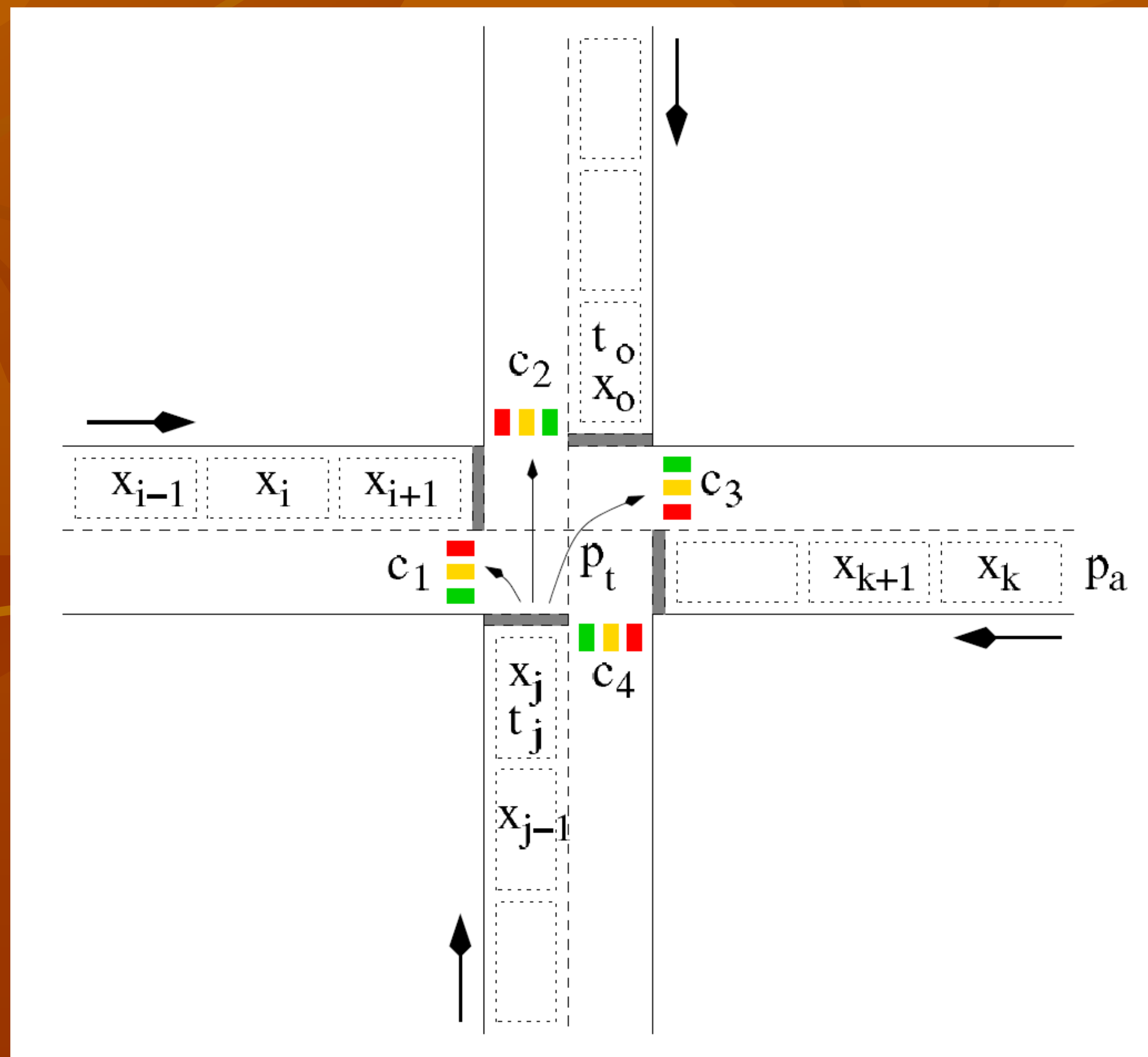




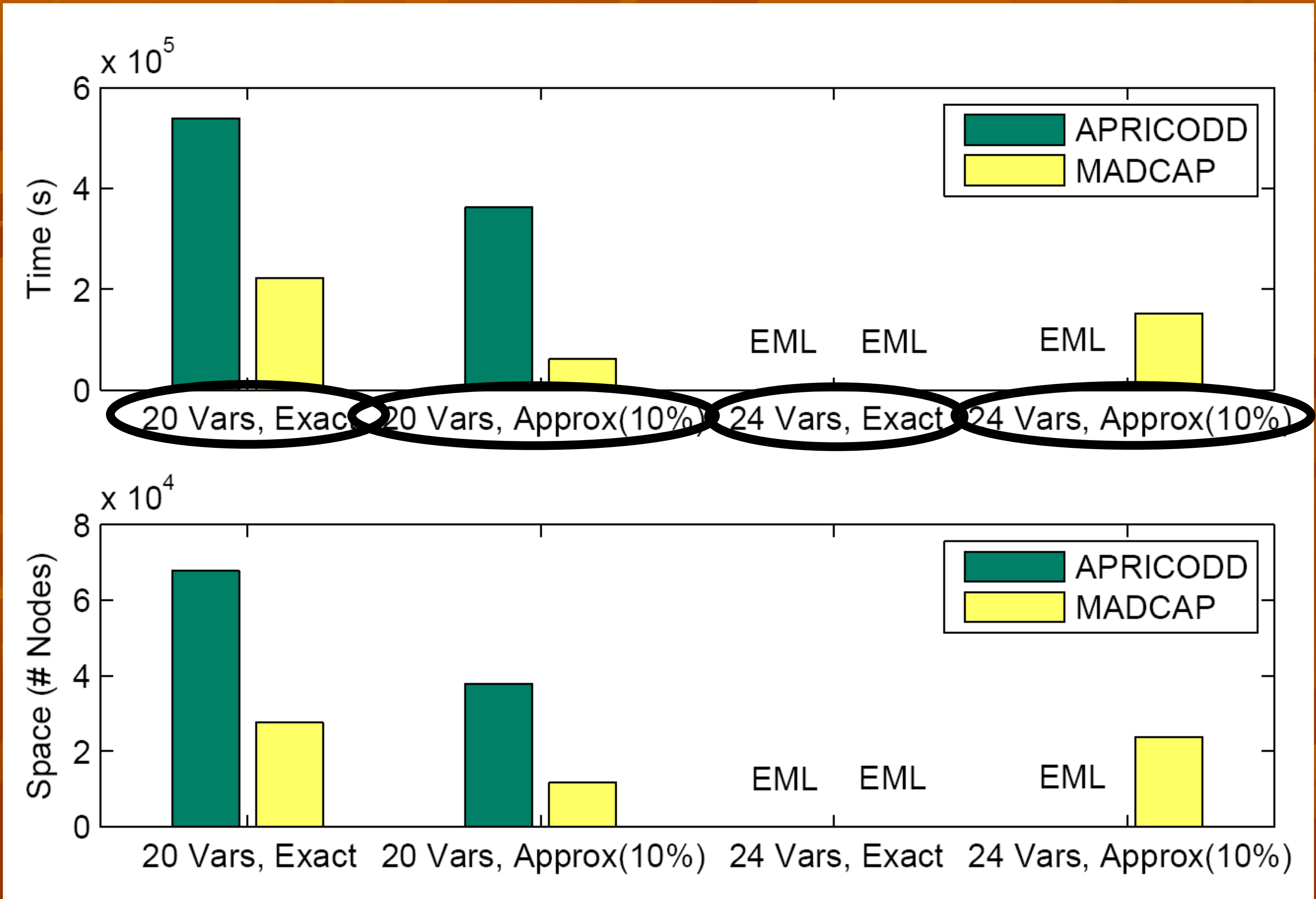


Traffic Domain

- **Binary cell transmission model (CTM)**
- Actions
  - Light changes
- Objective:
  - Maximize empty cells in network



Results Traffic



Conclusions

- AADDs replace Tables & ADDs
- AADD Properties:
  - Never worse than ADD or Table
  - Sometimes exp. reduction in space & time
- **Can now approximate AADD efficiently!**
  - MADCAP: Approx. DP for MDPs
  - Finds logical, additive, & multiplicative stuctured approximations automatically!