SPUDD: Stochastic Planning using Decision Diagrams

Kushagra Chandak

Paper by: Hoey et al, UAI 1999

30th Nov, 2020

Introduction: SPUDD

- Large state space: Curse of dimensionality.

Introduction: SPUDD

- Large state space: Curse of dimensionality.
- Abstraction/Aggregation techniques to obviate state enumeration.

Introduction: SPUDD

- Large state space: Curse of dimensionality.
- Abstraction/Aggregation techniques to obviate state enumeration.
- SPUDD: VI for MDPs and POMDPs using ADDs to <u>represent</u> value functions and policies.

Introduction: SPUDD

- Large state space: Curse of dimensionality.
- Abstraction/Aggregation techniques to obviate state enumeration.
- SPUDD: VI for MDPs and POMDPs using ADDs to represent value functions and policies.
- ADDs: Generalization of BDDs.

Introduction: SPUDD

- Large state space: Curse of dimensionality.
- Abstraction/Aggregation techniques to obviate state enumeration.
- SPUDD: VI for MDPs and POMDPs using ADDs to represent value functions and policies.
- ADDs: Generalization of BDDs.
- Derives from SPI which uses decision trees (unscalable) to represent π and V.

Introduction: SPUDD

- Large state space: Curse of dimensionality.
- Abstraction/Aggregation techniques to obviate state enumeration.
- SPUDD: VI for MDPs and POMDPs using ADDs to <u>represent</u> value functions and policies.
- ADDs: Generalization of BDDs.
- Derives from SPI which uses decision trees (unscalable) to represent π and V.
- Disjunctive structure in probability exploited by decision graphs.

Value Iteration (VI) Recap

- Bellman Equation: $V_\pi(s) = R(s) + \gamma \sum_{t \in \mathcal{S}} Pr(s, \pi(s), t).V_\pi(t)$

Value Iteration (VI) Recap

- Bellman Equation: $V_\pi(s) = R(s) + \gamma \sum_{t \in \mathcal{S}} Pr(s, \pi(s), t).V_\pi(t)$
- VI eqn: $V^{n+1}(s) = R(s) + \max_{a \in \mathcal{A}} \left\{ \gamma \sum_{t \in \mathcal{S}} Pr(s, a, t).V^n(t) \right\}$

Value Iteration (VI) Recap

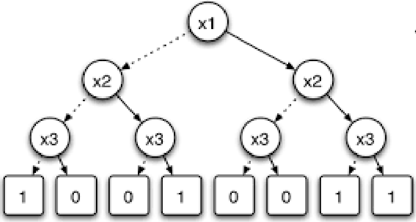
- Bellman Equation: $V_\pi(s) = R(s) + \gamma \sum_{t \in \mathcal{S}} Pr(s, \pi(s), t).V_\pi(t)$
- VI eqn: $V^{n+1}(s) = R(s) + \max_{a \in \mathcal{A}} \left\{ \gamma \sum_{t \in \mathcal{S}} Pr(s, a, t).V^n(t) \right\}$
- For some finite n, a's that maximize VI eqn form an opt π and V^n approximates its value.

Value Iteration (VI) Recap

- Bellman Equation: $V_\pi(s) = R(s) + \gamma \sum_{t \in \mathcal{S}} Pr(s, \pi(s), t).V_\pi(t)$
- VI eqn: $V^{n+1}(s) = R(s) + \max_{a \in \mathcal{A}} \left\{ \gamma \sum_{t \in \mathcal{S}} Pr(s, a, t).V^n(t) \right\}$
- For some finite n, a's that maximize VI eqn form an opt π and V^n approximates its value.
- Stopping criterion: $||V^{n+1} - V^n|| < \frac{\epsilon(1-\gamma)}{2\gamma}$
  $||X|| = \max\{|x| : x \in X\}$. Resulting π is ϵ-opt and V^{n+1} is within $\epsilon/2$ of V^*

ADD

- BDD: A function, $f : \mathcal{B}^n \rightarrow \mathcal{B}$
- ADD: Generalize BDD, $f : \mathcal{B}^n \rightarrow \mathcal{R}$.
    - Terminal node: $f(.) = c$
    - Non-terminal node: $f(x_1 \ldots x_n) = x_1.f_{then}(x_2 \ldots x_n) + \overline{x_1}.f_{else}(x_2 \ldots x_n)$



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 1: Binary Decision Diagram

ADD Representation of MDP

- State space: $\mathsf{X} = \{X_1 \ldots X_n\}$. Can be extended to multi-valued variables.

ADD Representation of MDP

- State space: $\mathrm{X} = \{X_1 \ldots X_n\}$. Can be extended to multi-valued variables.
- Action space: DBN. $\mathrm{X} = \{X_1, \ldots X_n\} \xrightarrow{a} \mathrm{X}' = \{X_1' \ldots X_n'\}$

ADD Representation of MDP

- State space: $\mathrm{X} = \{X_1 \dots X_n\}$. Can be extended to multi-valued variables.
- Action space: DBN. $\mathrm{X} = \{X_1, \dots X_n\} \xrightarrow{a} \mathrm{X}' = \{X_1' \dots X_n'\}$
- Directed arcs from variables in X to variables in X′ denote direct causal relationship.

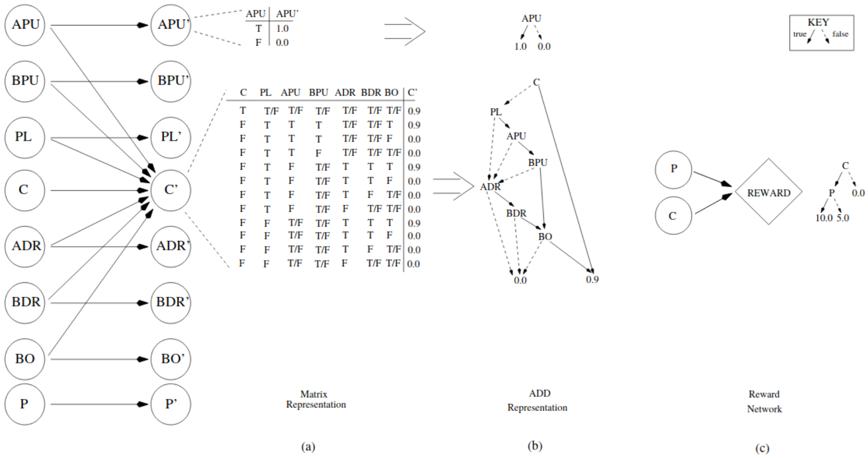
ADD Representation of MDP

- State space: $\mathrm{X} = \{X_1 \dots X_n\}$. Can be extended to multi-valued variables.
- Action space: DBN. $\mathrm{X} = \{X_1, \dots X_n\} \xrightarrow{a} \mathrm{X}' = \{X'_1 \dots X'_n\}$
- Directed arcs from variables in X to variables in X′ denote direct causal relationship.
- CPT for each post-action variable X'_i defines a conditional distribution $P^a_{X'_i}$ over X'_i: $P^a_{X'_i}(X_1 \dots X_n)$.

Example

- Process planning problem: A factory agent is tasked to connect 2 objects A and B.
- One way the agent can connect is take take action *bolt*.
- State C (objects connected) is independent of variable P (objects painted).
- If obj A is punched (APU) after bolting depends only on whether it was punched before bolting.
- Use ADDs to represent the functions $P^a_{X'_i}$ (to capture regularities in the CPTs)
- ADDs also exploit context-specific independence in the distributions.

Example



C	PL	APU	BPU	ADR	BDR	BO	C'
T	T/F	T/F	T/F	T/F	T/F	T/F	0.9
F	T	T	T	T/F	T/F	T	0.9
F	T	T	T	T/F	T/F	F	0.0
F	T	T	F	T/F	T/F	T/F	0.0
F	T	F	T/F	T	T	T	0.9
F	T	F	T/F	T	T	F	0.0
F	T	F	T/F	T	F	T/F	0.0
F	T	F	T/F	F	T/F	T/F	0.0
F	F	T/F	T/F	T	T	T	0.9
F	F	T/F	T/F	T	T	F	0.0
F	F	T/F	T/F	T	F	T/F	0.0
F	F	T/F	T/F	F	T/F	T/F	0.0

Figure 2: Small FACTORY example: (a) action network for action *bolt*; (b) ADD representation of CPTs (action diagrams); and (c) immediate reward network and ADD representation of the reward table.

Example

- Regularity in CPT: $Pr_{C'}^{bolt}(C, PL, APU, BPU, ADR, BDR, BO) = [C + \overline{C}[(PL \cdot \overline{APU} \cdot \overline{PL}) \cdot ADR \cdot BDR + PL \cdot APU \cdot BPU] \cdot BO] \cdot 0.9$
- Reward function as ADD: $R(C, P) = C \cdot P \cdot 10 + C \cdot \overline{P} \cdot 5$
- Disjunctive structure exploited by ADD. Eg., CPT for C': Variety of distinct conditions each give give rise to successfully connecting the 2 parts. (Similar to paths)
- ADDs more compact than trees (and tables): 7 internal nodes and 2 leaves vs 11 internal nodes and 12 leaves. Std matrix: 128 parameters.
- ADDs more compact than trees *most* times but not always.

SPUDD Algorithm: Overview

- Avoids explicit enumeration of the state space. (Similar to SDD)

SPUDD Algorithm: Overview

- Avoids explicit enumeration of the state space. (Similar to SDD)
- Classical VI but uses ADDs to represent Vs and CPTs.

SPUDD Algorithm: Overview

- Avoids explicit enumeration of the state space. (Similar to SDD)
- Classical VI but uses ADDs to represent Vs and CPTs.
- Savings both in space and computational time.

SPUDD Algorithm: Overview

- Avoids explicit enumeration of the state space. (Similar to SDD)
- Classical VI but uses ADDs to represent Vs and CPTs.
- Savings both in space and computational time.
- V at each step is represented as an ADD. ($V^0 = R$)

SPUDD Algorithm: Overview

- Avoids explicit enumeration of the state space. (Similar to SDD)
- Classical VI but uses ADDs to represent Vs and CPTs.
- Savings both in space and computational time.
- V at each step is represented as an ADD. ($V^0 = R$)
- Exploit ADD structure of V^i and MDP representation to get ADD structure for V^{i+1}.

SPUDD Algorithm

- Variables in V^i are replaced by their primed counterparts (post-action vars).

SPUDD Algorithm

- Variables in V^i are replaced by their primed counterparts (post-action vars).
- Goal: For each a, compute ADD for V_a^{i+1}: Exp value of performing action a.

SPUDD Algorithm

- Variables in V^i are replaced by their primed counterparts (post-action vars).
- Goal: For each a, compute ADD for V_a^{i+1}: Exp value of performing action a.
  - Negative action diagrams: $\overline{P^a_{X'_i}}(X_1 \ldots X_n) = 1 - P^a_{X'_i}(X_1 \ldots X_n)$: Probability that a will make X'_i false.

SPUDD Algorithm

- Variables in V^i are replaced by their primed counterparts (post-action vars).
- Goal: For each a, compute ADD for V_a^{i+1}: Exp value of performing action a.
  - Negative action diagrams: $\overline{P^a_{X_i'}}(X_1 \dots X_n) = 1 - P^a_{X_i'}(X_1 \dots X_n)$: Probability that a will make X_i' false.
  - Dual action diagrams: $Q^a_{X_i'}(X_i'; X_1 \dots X_n) = X_i' \cdot P^a_{X_i'}(X_1 \dots X_n) + \overline{X_i'} \cdot \overline{P^a_{X_i'}}(X_1 \dots X_n)$

SPUDD Algorithm

- Variables in V^i are replaced by their primed counterparts (post-action vars).
- Goal: For each a, compute ADD for V_a^{i+1}: Exp value of performing action a.
  - Negative action diagrams: $\overline{P^a_{X_i'}}(X_1 \dots X_n) = 1 - P^a_{X_i'}(X_1 \dots X_n)$: Probability that a will make X_i' false.
  - Dual action diagrams: $Q^a_{X_i'}(X_i'; X_1 \dots X_n) = X_i' \cdot P^a_{X_i'}(X_1 \dots X_n) + \overline{X_i'} \cdot \overline{P^a_{X_i'}}(X_1 \dots X_n)$
- Intuitively, Q denotes $P(X_i' = x_i' | X_1 = x_1 \dots X_n = x_n)$ (under action a)

SPUDD Algorithm

- Variables in V^i are replaced by their primed counterparts (post-action vars).
- Goal: For each a, compute ADD for V_a^{i+1}: Exp value of performing action a.
  - Negative action diagrams: $\overline{P^a_{X_i'}}(X_1 \ldots X_n) = 1 - P^a_{X_i'}(X_1 \ldots X_n)$: Probability that a will make X_i' false.
  - Dual action diagrams: $Q^a_{X_i'}(X_i'; X_1 \ldots X_n) = X_i' \cdot P^a_{X_i'}(X_1 \ldots X_n) + \overline{X_i'} \cdot \overline{P^a_{X_i'}}(X_1 \ldots X_n)$
- Intuitively, Q denotes $P(X_i' = x_i' | X_1 = x_1 \ldots X_n = x_n)$ (under action a)
- To generate $V_a^{i+1}(s)$: Combine $V_a^i(t)$ with probability of reaching t from s.

SPUDD Algorithm

- To get V_a^{i+1}: Multiply dual action diagrams X_j' by V'^i and then eliminate X_j'.

SPUDD Algorithm

- To get V_a^{i+1}: Multiply dual action diagrams X_j' by V'^i and then eliminate X_j'.
- $Q^a_{X_j'} \cdot V'^i = f(X_1' \ldots X_n', X_1 \ldots X_n)$

SPUDD Algorithm

- To get V_a^{i+1}: Multiply dual action diagrams X'_j by V'^i and then eliminate X'_j.
- $Q^a_{X'_j} \cdot V'^i = f(X'_1 \dots X'_n, X_1 \dots X_n)$
- $f(x'_1, \dots, x'_n, x_1 \dots x_n) = V'^i(x'_1 \dots x'_n) P(x'_j | x_1 \dots x_n)$

SPUDD Algorithm

- To get V_a^{i+1}: Multiply dual action diagrams X_j' by V'^i and then eliminate X_j'.
- $Q^a_{X_j'} \cdot V'^i = f(X_1' \ldots X_n', X_1 \ldots X_n)$
- $f(x_1', \ldots, x_n', x_1 \ldots x_n) = V'^i(x_1' \ldots x_n')P(x_j'|x_1 \ldots x_n)$
- Elimination of X_j' (Summing over left and right subgraphs of the ADD for f)

SPUDD Algorithm

- To get V_a^{i+1}: Multiply dual action diagrams X'_j by V'^i and then eliminate X'_j.
- $Q^a_{X'_j} \cdot V'^i = f(X'_1 \ldots X'_n, X_1 \ldots X_n)$
- $f(x'_1, \ldots, x'_n, x_1 \ldots x_n) = V'^i(x'_1 \ldots x'_n) P(x'_j | x_1 \ldots x_n)$
- Elimination of X'_j (Summing over left and right subgraphs of the ADD for f)
- $g(X'_1 \ldots X'_{j-1}, X'_{j+1]} \ldots X'_n, X_1 \ldots X_n) = \sum_{x'_j} V'^i(X'_1 \ldots x'_j \ldots X'_n) P(x'_j | X_1 \ldots X_n)$

SPUDD Algorithm

- Repeat the previous step for each post-action variable X_j' that occurs in ADD for V'^i: Multiply by Q and eliminate the prime variable.

SPUDD Algorithm

- Repeat the previous step for each post-action variable X'_j that occurs in ADD for V'^i: Multiply by Q and eliminate the prime variable.
- After elimninating all prime variables, we get $h(X_1 \dots X_n) = \sum_{x'_1 \dots x'_n} V'^i(x'_1 \dots x'_n) P(x'_1 | X_1 \dots X_n) \dots P(x'_n | X_1 \dots X_n)$

SPUDD Algorithm

- Repeat the previous step for each post-action variable X'_j that occurs in ADD for V'^i: Multiply by Q and eliminate the prime variable.
- After elimnating all prime variables, we get $h(X_1 \dots X_n) = \sum_{x'_1 \dots x'_n} V'^i(x'_1 \dots x'_n) P(x'_1|X_1 \dots X_n) \dots P(x'_n|X_1 \dots X_n)$
- $R + h$: ADD representation of V_a^{i+1}

SPUDD Algorithm

- Repeat the previous step for each post-action variable X'_j that occurs in ADD for V'^i: Multiply by Q and eliminate the prime variable.
- After elimnating all prime variables, we get $h(X_1 \dots X_n) = \sum_{x'_1 \dots x'_n} V'^i(x'_1 \dots x'_n) P(x'_1|X_1 \dots X_n) \dots P(x'_n|X_1 \dots X_n)$
- $R + h$: ADD representation of V_a^{i+1}
- ADD for $V^{i+1} = \max_{a \in \mathcal{A}} V_a^{i+1}$

Summary and Conclusion

- SPUDD: VI for solving MDPs using ADDs.

Summary and Conclusion

- SPUDD: VI for solving MDPs using ADDs.
- ADDs capture regularities in the system dynamics, reward and value: Compact representation of the problem. (vs explicit matrix and decision tree methods)

Summary and Conclusion

- SPUDD: VI for solving MDPs using ADDs.
- ADDs capture regularities in the system dynamics, reward and value: Compact representation of the problem. (vs explicit matrix and decision tree methods)
- Drawback: Boolean variables only. (Multi-valued variables can be split into Boolean variables)

Summary and Conclusion

- SPUDD: VI for solving MDPs using ADDs.
- ADDs capture regularities in the system dynamics, reward and value: Compact representation of the problem. (vs explicit matrix and decision tree methods)
- Drawback: Boolean variables only. (Multi-valued variables can be split into Boolean variables)
- Ordering of variables fixed; dynamic ordering could reduce size. (Done by SDD)

Summary and Conclusion

- SPUDD: VI for solving MDPs using ADDs.
- ADDs capture regularities in the system dynamics, reward and value: Compact representation of the problem. (vs explicit matrix and decision tree methods)
- Drawback: Boolean variables only. (Multi-valued variables can be split into Boolean variables)
- Ordering of variables fixed; dynamic ordering could reduce size. (Done by SDD)
- Extensions to other dynamic programming algorithms.