### Probabilistic planning (June 13, 2005)

#### Motivation

Probabilistic transition systems

#### Example

**Definition**

Probability distribution of states under a plan

**Evaluation of performance**

**Examples**

**Definition**

Algorithms

- Finite executions
- Value iteration
- Policy iteration
- Goal-directed problems

**Implementation**

- Algebraic decision diagrams ADDs
- Value iteration with ADDs

---

#### Probabilistic planning: Quality criteria for plans

The purpose of a plan may vary.

1. Reach goals with probability 1.
2. Reach goals with the highest possible probability.
3. Reach goals with the smallest possible expected cost.
4. Gain highest possible expected rewards (over a finite or an infinite execution).

For each objective a different algorithm is needed.

---

#### Probabilistic transition system

**Definition**

A probabilistic transition system is \((S, I, O, G, R)\) where

1. \(S\) is a finite set of states,
2. \(I\) is a probability distribution over \(S\),
3. \(O\) is a finite set of actions = partial functions that map each state to a probability distribution over \(S\),
4. \(G \subseteq S\) is the set of goal states, and
5. \(R : O \times S \rightarrow R\) is a function from actions and states to real numbers, indicating the reward associated with an action in a given state.

---

#### Probabilistic operators

**Example**

Let \(a = \{\neg a, 0.2a(0.8b) \land (0.4c(0.6c))\}\). Compute the successors of \(s = a \land \neg b \land \neg c\) with respect to \(a\).

Active effects:

- \([0.2a(0.8b)]_a = \{0.2(\{a\}, 0.8(\{b\})\}\}
- \([0.4c(0.6c)]_a = \{0.4(\{c\}), 0.6(\{b\})\}\}
- \([0.2a(0.8b) \land (0.4c(0.6c))]_a = \{0.08(\{a, c\}), 0.32(\{b, c\}), 0.12(\{a\}), 0.48(\{b\})\}\}

Successor states of \(s\) with respect to \(a\) are:

- \(s_1 = a \land \neg b \land c\) (probability 0.08)
- \(s_2 = \neg a \land b \land c\) (probability 0.32)
- \(s_3 = a \land \neg b \land \neg c\) (probability 0.12)
- \(s_4 = \neg a \land b \land \neg c\) (probability 0.48).

---

### Motivation for introducing probabilities

- Reaching the goals is often not sufficient: it is important that the expected costs do not outweigh the benefit of reaching the goals.
  1. Objective: maximize benefits - costs.
  2. Measuring expected benefits requires considering the probabilities of effects.
- Plans that guarantee achieving goals often do not exist. Then it is important to find a plan that maximizes success probability.

---

#### Probabilities for nondeterministic actions

**Notation**

Applicable actions

\(O(s)\) denotes the set of actions that are applicable in \(s\).

**Notation: Applicable actions**

Effect probabilities

\(p(x|s, a)\) denotes the probability \(a\) assigns to \(x\) as a successor state of \(s\).

**Notation: Probabilities of successor states**

- **Definition**
  
  An operator is a pair \((c, e)\) where \(c\) is a propositional formula (the precondition), and \(e\) is an effect. Effects are recursively defined as follows.
  
  1. \(a\) and \(\neg a\) for state variables \(a \in A\) are effects.
  2. \(e_1 \land \cdots \land e_n\) if \(e_1, \ldots, e_n\) are effects (the special case with \(n = 0\) is the empty effect \(\top\)).
  3. \(c \Rightarrow e\) is an effect if \(c\) is a formula and \(e\) is an effect.
  4. \(p_1 e_1 + \cdots + p_n e_n\) if \(p_1, \ldots, p_n\) are real numbers such that \(p_1 + \cdots + p_n = 1\) and \(0 \leq p_i \leq 1\) for all \(i \in \{1, \ldots, n\}\).

Operators map states to probability distributions over their successor states.
Probabilistic operators

Definition (Active effects)
Assign effects \( e \) a set of pairs of numbers and literal sets.

1. \([-\alpha] = \{(1, \alpha)\}\) and \([\alpha] = \{(1, -\alpha)\}\) for \( \alpha \in A \).
2. \([e_1 \land \ldots \land e_n] = \left( \bigcap_{i=1}^{n} e_i \right) \cap \left( \bigcup_{i=1}^{n} e_i \right) \) for \( e_i \in E \).
3. \([e \lor j] = \{j\}\) if \( j \) is a literal and otherwise \([e \lor \{\neg \} j] = \{1, 0\}\).
4. \([p \land e] = \{p\} \land \{e\}\) if \( p \in \text{states} \) and \([p \land e] = \{e\}\) otherwise.

Remark
In (4) the union of sets is defined so that for example \( \{(0.2, \{\alpha\})\} \cup \{(0.2, \{\beta\})\} = \{(0.4, \{\alpha, \beta\})\} \) and 

\[
\bar{a} \text{ is one of the state after a successfull}\hfill \\
\bar{a} = \{a \}
\]

Probabilistic succint transition systems

Definition
A succint probabilistic transition system is \( \langle A, I, O, G, W \rangle \) where

1. \( A \) is a finite set of state variables, \( A \)
2. \( I = \{(p_1, \phi_1), \ldots, (p_n, \phi_n)\} \) where \( 0 < p_i \leq 1 \) and \( \phi_i \) is a formula over \( A \) for every \( i \in \{1, \ldots, n\} \) and \( \sum_{i=1}^{n} \phi_i = 1 \) describes the initial probability distribution over the states, \( I \)
3. \( G \) is a finite set of operators over \( A \), \( G \)
4. \( O \) is a formula over \( A \) describing the goal states, and \( O \)
5. \( W \) is a function from operators to sets of pairs \((\phi, r)\) where \( \phi \) is a formula over \( A \) and \( r \) is a real number: reward of executing \( e \) in \( s \) is \( r \) if there is \((\phi, r) \in W(\phi)\) such that \( s \models \phi \) and otherwise the reward is 0. \( W \)

Stationary probabilities under a plan

The probability of the transition system being in given states can be computed by matrix multiplication from the probability distribution for the initial states and the transition probabilities of the plan.

\[
\begin{array}{cccc}
J & M & JM & JM^2 \\
\hline
A & 0.8 & 0.2 & 0.8 & 0.2 \\
B & 0.6 & 0.0 & 0.0 & 0.0 \\
C & 0.0 & 0.0 & 0.2 & 0.8 \\
D & 0.0 & 0.0 & 1.0 & 0.0 \\
E & 0.0 & 1.0 & 0.0 & 0.0 \\
\end{array}
\]

Stationary probabilities under a plan

\[
J = (0.9)(0.1)(0.0)(0.0)
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
0.9 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.9 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0 \\
0.0 & 0.0 & 1.0 & 0.0 \\
\end{array}
\]

Probabilities of states under a plan (periodic)

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
0.0 & 0.9 & 0.0 & 0.1 \\
0.1 & 0.0 & 0.9 & 0.0 \\
0.0 & 0.0 & 0.9 & 0.1 \\
0.0 & 0.1 & 0.0 & 0.9 \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
0.9 & 0.0 & 0.0 & 0.1 \\
0.0 & 0.9 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.9 & 0.0 \\
0.0 & 0.1 & 0.0 & 0.9 \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
0.0 & 0.0 & 0.9 & 0.1 \\
0.1 & 0.0 & 0.0 & 0.9 \\
0.0 & 0.0 & 0.1 & 0.9 \\
0.0 & 0.1 & 0.0 & 0.9 \\
\end{array}
\]

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
0.0 & 0.0 & 0.0 & 1.0 \\
0.0 & 0.0 & 0.0 & 1.0 \\
0.0 & 0.0 & 0.0 & 1.0 \\
0.0 & 0.0 & 0.0 & 1.0 \\
\end{array}
\]

Probabilitydistribution of states under a plan

The probability of the transition system being in given states can be computed by matrix multiplication from the probability distribution for the initial states and the transition probabilities of the plan.

\[
\begin{array}{cccc}
J & M & JM & JM^2 \\
\hline
A & 0.8 & 0.2 & 0.8 & 0.2 \\
B & 0.6 & 0.0 & 0.0 & 0.0 \\
C & 0.0 & 0.0 & 0.2 & 0.8 \\
D & 0.0 & 0.0 & 1.0 & 0.0 \\
E & 0.0 & 1.0 & 0.0 & 0.0 \\
\end{array}
\]

Remark
In (4) the union of sets is defined so that for example \( \{(0.2, \{\alpha\})\} \cup \{(0.2, \{\beta\})\} = \{(0.4, \{\alpha, \beta\})\} \) and 

\[
\bar{a} \text{ is one of the state after a successfull}\hfill \\
\bar{a} = \{a \}
\]
Evaluation of performance

Average rewards

- A waiter/waitress robot
  - induceth(costsoffoodandbeveragesbroughttocustomers,
brokenplatesandglasses,...
  - bringswards:collectsmoneyfromcustomers.
- Thiscanbeviewedasainfinitesequencesofrewards-0.0,3.1,6.9,-0.80,-1.2,2.6,12.8,-1.1,2.1,-10.0.
- Owner'sobjective:theplantherobotfollowsmustmaximizethe
  average reward.

Evaluation of performance

Discounted rewards

- A company decides every month the pricing of its products and
  performs other actions affecting its costs and profits.
- Since there is a lot of uncertainty about distant future, the
  company's short-term performance (next 1-4 years) is more
  important than long-term performance (after 5 years) and distant
  future (after 10 years) is almost completely left out of all
  calculations.
- This can be similarly viewed as an infinite sequence -1.1,2.1,-
  10.0,4.5,-0.6,-1.0,3.6,18.4,... but the reward at point \(t+1\)
is discounted by a factor \(\lambda \in ]0,1[\) in comparison to reward at \(t\) to
reflect the importance of short-term performance.

Rewards/costs produced by a plan

An infinite sequence of expected rewards \(r_1, r_2, r_3, \ldots\) can be
evaluated in alternative ways:

1. **total rewards**: sum of all rewards \(r_1 + r_2 + \cdots\)
2. **average rewards** \(\lim_{N \to \infty} \frac{\sum r_i}{N}\)
3. **discounted rewards** \(r_1 + \lambda r_2 + \lambda^2 r_3 + \cdots + \lambda^{k-1} r_k + \cdots\)

For infinite executions the sums \(\sum_{i=0}^{\infty} r_i\) are typically **infinite** and
discounting is necessary to make them finite. The geometric series
has a finite sum \(\sum_{i=0}^{\infty} \lambda^i = \frac{1}{1-\lambda}\) for every \(\lambda < 1\) and \(c\).

Optimal rewards over a finite execution

- Objective: obtain highest possible rewards over a finite execution
  of length \(N\) (goals are ignored).
- Solution by dynamic programming:
  1. Value of a state at last stage \(N\) is the **best immediate reward**.
  2. Value of a state at stage \(i\) is obtained from values of states at stage
     \(i+1\).
- Since the executions are finite, it is possible to sum all rewards
  and no discounting is needed.
- Since efficiency degrades with long executions, this algorithm
  is not practical for very high \(N\).

Optimal rewards over a finite execution

- **Examples**

Optimal rewards over a finite execution

- **Algorithm**

Probabilistic planning with full observability

- Several algorithms:
  1. dynamic programming (finite executions)
  2. value iteration (discounted rewards, infinite execution)
  3. policy iteration (discounted rewards, infinite execution)

- Some of these algorithms can be easily implemented without
  explicitly representing the state space (e.g. by using algebraic
decision diagrams ADDs).

Optimal rewards over a finite execution

- **Algorithm**

The optimum values \(v_s(t)\) for states \(s \in S\) at time \(t \in \{1, \ldots, N\}\) fulfill the following equations.

\[
v_s(t) = \max_{a \in O(s)} R(s, a) + \sum_{s' \in S} p(s'|s, a) v_{s'}(t+1)
\]

for \(i \in \{1, \ldots, N-1\}\)
Optimal plans over a finite execution

**Algorithm**

Actions for states $s \in S$ at times $t \in \{1, \ldots, N\}$ are:

$$\pi(s, N) = \arg \max_{a \in O(s)} R(s, a)$$

$$\pi(s, i) = \arg \max_{a \in O(s)} \left( R(s, a) + \sum_{s' \in S} p(s'|s, a)v_{i+1}(s') \right)$$

for $i \in \{1, \ldots, N - 1\}$

**Receding-horizon control**

Finite-horizon policies can be applied to infinite-execution problems as well: always take action $\pi(s, 1)$. This is known as receding-horizon control.

---

Optimality / Bellman equations

**Infinite executions**

Values $v(s)$ of states $s \in S$ are the discounted sum of the expected rewards obtained by choosing the best possible actions in $s$ and in its successors.

$$v(s) = \max_{a \in O(s)} \left( R(s, a) + \sum_{s' \in S} \lambda p(s'|s, a)v(s') \right)$$

$\lambda$ is the discount constant: $0 < \lambda < 1$.

**Value iteration**

Example

Let $\lambda = 0.6$.

$$v^*(A) = v^*(B) = v^*(C) = v^*(D) = v^*(E)$$

<table>
<thead>
<tr>
<th>$s$</th>
<th>$v_0(s)$</th>
<th>$v_1(s)$</th>
<th>$v_2(s)$</th>
<th>$v_3(s)$</th>
<th>$v_4(s)$</th>
<th>$v_5(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>2.760</td>
<td>2.760</td>
<td>2.760</td>
<td>2.760</td>
<td>1.878</td>
<td>1.878</td>
</tr>
<tr>
<td>3</td>
<td>1.656</td>
<td>1.656</td>
<td>1.656</td>
<td>1.656</td>
<td>1.656</td>
<td>1.656</td>
</tr>
<tr>
<td>4</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
</tr>
<tr>
<td>5</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
<td>1.994</td>
</tr>
<tr>
<td>6</td>
<td>1.796</td>
<td>1.796</td>
<td>1.796</td>
<td>1.796</td>
<td>1.796</td>
<td>1.796</td>
</tr>
<tr>
<td>7</td>
<td>1.310</td>
<td>1.078</td>
<td>1.078</td>
<td>1.078</td>
<td>1.078</td>
<td>1.078</td>
</tr>
<tr>
<td>8</td>
<td>1.878</td>
<td>1.878</td>
<td>1.878</td>
<td>1.878</td>
<td>1.878</td>
<td>1.878</td>
</tr>
<tr>
<td>9</td>
<td>1.132</td>
<td>1.132</td>
<td>1.132</td>
<td>1.132</td>
<td>1.132</td>
<td>1.132</td>
</tr>
</tbody>
</table>

---

The policy iteration algorithm

**Algorithm**

- The policy iteration algorithm finds optimal plans.
- Slightly more complicated to implement than value iteration: on each iteration
  - the value of the current plan is evaluated, and
  - the current plan is improved if possible.
- Number of iterations is smaller than with value iteration.
- Value iteration is usually in practice more efficient.

---

Value iteration

Plan evaluation by solving linear equations

Given a plan $\pi$, its value $v_\pi$ under discounted rewards with discount constant $\lambda$ satisfies the following equation. For every $s \in S$

$$v_\pi(s) = R(s, \pi(s)) + \sum_{s' \in S} \lambda p(s'|s, \pi(s))v_\pi(s')$$

This yields a system of $|S|$ linear equations and $|S|$ unknowns. The solution of these equations gives the value of the plan in each state.
Plan evaluation by solving linear equations

\[ \pi = \text{arg max} \ \frac{R(v(s)) + \sum_{s' \in S} p(s'|s,v) v(s')} \]

\[ \pi(A) = 1 \]
\[ \pi(B) = 0 \]
\[ \pi(C) = 1 \]
\[ \pi(D) = 5 \]
\[ \pi(E) = 0 \]

Consider the plan
\[ \pi(A) = R, \pi(B) = R, \pi(C) = R, \pi(D) = R, \pi(E) = B \]

\[ v_\pi(A) = -0.1 \lambda v_\pi(B) + 0.1 \lambda v_\pi(C) + 0.1 \lambda v_\pi(D) + 0.1 \lambda v_\pi(E) \]
\[ v_\pi(B) = R \]
\[ v_\pi(C) = 0 \]
\[ v_\pi(D) = 5 \]
\[ v_\pi(E) = 0 \]

The policy iteration algorithm

1. \( n := 0 \)
2. Let \( \pi_0 \) be any mapping from states \( s \in S \) to actions in \( O(s) \).
3. Compute \( v_{\pi_n}(s) \) for all \( s \in S \).
4. For all \( s \in S \)

\[ \pi_{n+1}(s) = \text{arg max} \ \frac{R(v(s)) + \sum_{s' \in S} p(s'|s,v) v_{\pi_n}(s')} \]

5. \( n := n + 1 \)
6. If \( n = 1 \) or \( v_{\pi_n} \neq v_{\pi_{n-1}} \) then go to 3.

The policy iteration algorithm

Example

<table>
<thead>
<tr>
<th>ltr</th>
<th>(\pi(A))</th>
<th>(\pi(B))</th>
<th>(\pi(C))</th>
<th>(\pi(D))</th>
<th>(\pi(E))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

Bounded goal reachability with minimum cost

Define for all \( i \geq 0 \) the following value functions for the expected cost of reaching a goal state.

\[ v_0(s) = -\infty \text{ for } s \in S \setminus G \]
\[ v_0(s) = 0 \text{ for } s \in G \]
\[ v_{i+1}(s) = \max_{a \in O(s)} \left( R(s,a) + \sum_{s' \in S} p(s'|s,a) v_i(s') \right) \text{ for } s \in S \setminus G \]

This computation converges if for every \( \epsilon \) there is \( i \) such that

\[ |v_i(s) - v_{i+1}(s)| < \epsilon. \]

Notice

The above algorithm is guaranteed to converge only if all rewards are \( < 0 \). If some rewards are positive, the most rewarding behavior may be to loop without ever reaching the goals.

Goal reachability with highest probability

Define for all \( i \geq 0 \) the following value functions expressing the probability of eventually reaching a goal.

\[ v_0(s) = 0 \text{ for } s \in S \setminus G \]
\[ v_0(s) = 1 \text{ for } s \in G \]
\[ v_{i+1}(s) = \min_{a \in O(s)} \sum_{s' \in S} p(s'|s,a) v_i(s') \text{ for } s \in S \setminus G \]

Notice

The above algorithm converges to \( v \) such that \( v(s) = 1 \) iff \( s \in L \cup G \) where \( L \) is the set returned by \( \text{prune} \).
Implementation for big state spaces

Fact: The most trivial way of implementing the previous algorithms is feasible only for state space sizes of up to $10^6$ or $10^7$.

Problem: Every state in the state space has to be considered explicitly, even when it is not needed for the solution.

Solution
1. Use algorithms that restrict to the relevant part of the state space: Real-Time Dynamic Programming RTDP, ...
2. Use data structures that represent sets of states and probability distributions compactly: size of the data structure is not necessarily linear in the number of states, but could be logarithmic or less.

Algebraic decision diagrams

- Graph representation of functions from $\{0, 1\}^n \rightarrow \mathcal{R}$ that generalizes BDDs (functions $\{0, 1\}^n \rightarrow \{0, 1\}$)
- Every BDD is an ADD.
- Canonicity: Two ADDs describe the same function if and only if they are the same ADD.
- Applications: Computations on very big matrices including computing stationary probabilities of Markov chains; probabilistic verification; AI planning

Operations on ADDs

Operations $\odot$ for ADDs $f$ and $g$ are definable by $(f \odot g)(x) = f(x) \odot g(x)$.

<table>
<thead>
<tr>
<th>abc</th>
<th>$f \odot g$</th>
<th>$f + g$</th>
<th>$\max(f, g)$</th>
<th>$\top \cdot f$</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0 1 2 3 4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>001</td>
<td>1 2 3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>010</td>
<td>0 1 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>011</td>
<td>2 1 3 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>100</td>
<td>0 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>101</td>
<td>0 2 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>110</td>
<td>0 2 2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>111</td>
<td>3 4 3 2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>21</td>
</tr>
</tbody>
</table>

Operations on ADDs

Maximum

\[ \max(x, y) = (x \lor y) \land (x \land y) \]

Arithmetic abstraction

\[ (\exists x.f)(x) = (f[\top/p](x) + (f[\bot/p](x) \]

Operations on ADDs

Sum

\[ \text{Operations on ADDs} \]

Maximum

\[ \max(x, y) = (x \lor y) \land (x \land y) \]

Arithmetic abstraction

\[ (\exists x.f)(x) = (f[\top/p](x) + (f[\bot/p](x) \]

Implementation for big state spaces

Like binary decision diagrams (BDDs) can be used in implementing algorithms that use strong/weak preimages, there are data structures that can be used for implementing probabilistic algorithms for big state spaces.

Problem: Algorithms do not use just sets and relations which can be represented as BDDs, but value functions \(\nu : S \rightarrow \mathcal{R}\) and non-binary transition matrices.

Solution: Use a generalization of BDDs called algebraic decision diagrams (or MTBDDs: multi-terminal BDDs.)
Matrix multiplication with ADDs (I)

Consider matrices $M_1$ and $M_2$, represented as mappings:

$$
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
\begin{pmatrix}
1 & 2 \\
2 & 1
\end{pmatrix}
$$

- $a'^0 M_1$
- $a'^1 M_2$

$$
\begin{pmatrix}
00 & 1 \\
01 & 2 \\
10 & 1 \\
11 & 1
\end{pmatrix}
\begin{pmatrix}
00 & 1 \\
01 & 2 \\
10 & 1 \\
11 & 1
\end{pmatrix}
$$

Matrix multiplication with ADDs (II)

$$
\begin{pmatrix}
00 & 1 & 1 & 1 \\
01 & 2 & 2 & 4 \\
01 & 2 & 1 & 2 \\
10 & 3 & 1 & 3 \\
10 & 1 & 2 & 6 \\
11 & 4 & 2 & 8 \\
11 & 4 & 1 & 4
\end{pmatrix}
\begin{pmatrix}
1 & 2 \\
2 & 1
\end{pmatrix}
= 
\begin{pmatrix}
00 & 5 \\
01 & 4 \\
11 & 10
\end{pmatrix}
$$

Implementation of value iteration with ADDs

- Start from $\langle A, I, O, G, W \rangle$.
- Variables in ADDs $A$ and $A' = \{ a' | a \in A \}$.
- Construct transition matrix ADDs from all $o \in O$ (next slide).
- Construct ADDs for representing rewards $\mathbb{W}(o), o \in O$.
- Functions $v_s$ are ADDs that map valuations of $A$ to $\mathbb{R}$.
- All computation is for all states (one ADD) simultaneously: big speed-ups possible.

Translation of reward functions into ADDs

- Let the rewards for $o = (c, e) \in O$ be represented by $\mathbb{W}(o) = \{ (\phi_1, r_1), \ldots, (\phi_n, r_n) \}$.
- We construct an ADD $R_s$ that maps each state to the corresponding rewards.
- This is by constructing the BDDs for $\phi_1, \ldots, \phi_n$ and then multiplying them with the respective numbers $r_1, \ldots, r_n$.
- $R_s = r_1 \cdot \phi_1 + \cdots + r_n \cdot \phi_n - \infty \cdot \neg c$

The value iteration algorithm without ADDs

1. $n := 0$
2. Choose any value function $v_0$.
3. For every $s \in S$
   $$v_{n+1}(s) = \max_{a \in D(s)} \left( R(s, a) + \sum_{c \in C(s)} \lambda p^{\text{state}}(c) v_n(s) \right)$$
   Go to step 4 if $|v_{n+1}(s) - v_n(s)| < \frac{1}{2^n}$ for all $s \in S$. Otherwise set $n := n + 1$ and go to step 3.

The value iteration algorithm with ADDs

Backup step for $v_{n+1}$ with $a$ as product of $\tau^{\text{state}}_A(o)$ and $v_n$:

$$
R_a + \lambda
\begin{pmatrix}
00 & 00 & 01 & 10 & 11 \\
01 & 00 & 01 & 10 & 11 \\
10 & 02 & 00 & 08 & 00 \\
11 & 10 & 00 & 00 & 00
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix} a'^0 & a'^1 & a'^2 & a'^3 & a'^4 \\ a'^0 & a'^1 & a'^2 & a'^3 & a'^4 \end{pmatrix} \\
\begin{pmatrix} v_n \end{pmatrix}
\end{pmatrix}
$$

Remark

The fact that the operator is not applicable in 11 is handled by having the immediate reward $-\infty$ in that state.
Summary

- Probabilities are needed when plan has to have low expected costs or a high success probability when success cannot be guaranteed.
- We have presented several algorithms based on dynamic programming.
- Most of these algorithms can be easily implemented by using Algebraic Decision Diagrams ADDs as a data structure for representing probability distributions and transition matrices.
- There are also other algorithms that do not always require looking at every state but restrict to states that are reachable from the initial states.