Probabilistic planning (June 13, 2005)

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 costs 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.

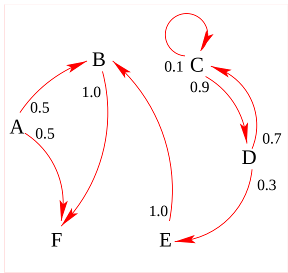
Probabilistic planning: Quality criteria for plans

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.

Probabilities for nondeterministic actions



	A	B	C	D	E	F
A	0	0.5	0	0	0	0.5
B	0	0	0	0	0	1.0
C	0	0	0.1	0.9	0	0
D	0	0	0.7	0	0.3	0
E	0	1.0	0	0	0	0
F	0	0	0	0	0	0

Probabilistic transition system

Definition
A probabilistic transition system is $\langle S, I, O, G, R\rangle$ 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. $R : O \times S \to \mathcal{R}$ is a function from actions and states to real numbers, indicating the reward associated with an action in a given state.

Probabilistic transition system
Notation

Notation: Applicable actions
$O(s)$ denotes the set of actions that are applicable in s.

Notation: Probabilities of successor states
$p(s'|s, o)$ denotes the probability o assigns to s' as a successor state of s.

Probabilistic operators
Example

Let $o = \langle \neg a, (0.2a|0.8b) \wedge (0.4c|0.6\top)\rangle$. Compute the successors of $s \models \neg a \wedge \neg b \wedge \neg c$ with respect to o.
Active effects:

$$\begin{aligned}
[0.2a|0.8b]_s &= \{\langle 0.2, \{a\}\rangle, \langle 0.8, \{b\}\rangle\} \\
[0.4c|0.6\top]_s &= \{\langle 0.4, \{c\}\rangle, \langle 0.6, \emptyset\rangle\} \\
[(0.2a|0.8b) \wedge (0.4c|0.6\top)]_s &= \{\langle 0.08, \{a, c\}\rangle, \langle 0.32, \{b, c\}\rangle, \\
&\quad \langle 0.12, \{a\}\rangle, \langle 0.48, \{b\}\rangle\}
\end{aligned}$$

Successor states of s with respect to o are

$$\begin{aligned}
s_1 &\models a \wedge \neg b \wedge c, \text{(probability 0.08)} \\
s_2 &\models \neg a \wedge b \wedge c, \text{(probability 0.32)} \\
s_3 &\models a \wedge \neg b \wedge \neg c, \text{(probability 0.12)} \\
s_4 &\models \neg a \wedge b \wedge \neg c \text{(probability 0.48)}.
\end{aligned}$$

Probabilistic operators
Definition

Definition
An operator is a pair $\langle c, e\rangle$ 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. $e_1 \wedge \cdots \wedge e_n$ is an effect if $e_1, \ldots, e_n$ are effects (the special case with $n = 0$ is the empty effect $\top$.)
3. $c \rhd e$ is an effect if c is a formula and e is an effect.
4. $p_1e_1|\cdots|p_ne_n$ is an effect if $n \geq 2$ and $e_1, \ldots, e_n$ for $n \geq 2$ are effects and $p_1, \ldots, p_n$ 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
Semantics

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

1. $[a]_s = \{\langle 1, \{a\}\rangle\}$ and $[\neg a]_s = \{\langle 1, \{\neg a\}\rangle\}$ for $a \in A$.
2. $[e_1 \wedge \cdots \wedge e_n]_s$
   $= \{\langle \prod_{i=1}^{n} p_i, \bigcup_{i=1}^{n} M_i\rangle | \langle p_1, M_1\rangle \in [e_1]_s, \ldots, \langle p_n, M_n\rangle \in [e_n]_s\}$.
3. $[z \rhd e]_s = [e]_s$ if $s \models z$ and otherwise $[z \rhd e]_s = \{\langle 1, \emptyset\rangle\}$.
4. $[p_1 e_1 | \cdots | p_n e_n]_s = \bigcup_{i \in \{1,\ldots,n\}} \{\langle p_i \cdot p, e\rangle | \langle p, e\rangle \in [e_i]_s\}$

Remark
In (4) the union of sets is defined so that for example $\{\langle 0.2, \{a\}\rangle\} \cup \{\langle 0.2, \{a\}\rangle\} = \{\langle 0.4, \{a\}\rangle\}$: same sets of changes are combined by summing their probabilities.

Probabilistic operators
Semantics

Definition
Let $\langle c, e\rangle$ be an operator and s a state.

- o is applicable in s if $s \models c$ and for every $E \in [e]_s$ the set $\bigcup\{M | \langle p, M\rangle \in E, p > 0\}$ is consistent.
- The successor states of s under operator with effect e are ones that are obtained from s by making the literals in M for some $\langle p, M\rangle \in [e]_s$ true and retaining the truth-values of state variables not occurring in M.
- The probability of a successor state is the sum of the probabilities p for $\langle p, M\rangle \in [e]_s$ that lead to it.

Probabilistic succinct transition systems

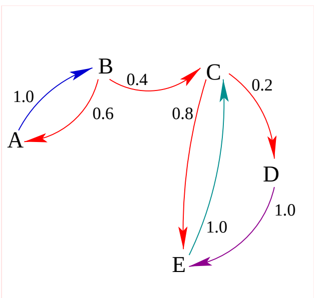
Definition
A succinct probabilistic transition system is $\langle A, I, O, G, W\rangle$ where

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

Stationary probabilities under a plan

- To measure the performance of a plan it is necessary to compute the probabilities of different executions.
- If there is an infinite number of infinite-length executions then infinitely many of them have probability 0.
- Probability of execution $s_0, s_1, s_2, \ldots, s_n$ is obtained as the product of the initial probability of s_0 multiplied by the probabilities of reaching s_i from s_{i-1} with $\pi(s_{i-1})$ for all $i \in \{1, \ldots, n\}$.
- It is often possible to associate a unique probability with each state: the stationary probability of the state after a sufficiently high number of execution steps = probability that at any time point the current state is that state.
- Some cases there is no unique stationary probability, in which case the plan executions are periodic.

Stationary probabilities under a plan



$J = (0.9, 0.1, 0, 0, 0)$

	A	B	C	D	E
A	0	1.0	0	0	0
B	0.6	0	0.4	0	0
C	0	0	0	0.2	0.8
D	0	0	0	0	1.0
E	0	0	1.0	0	0

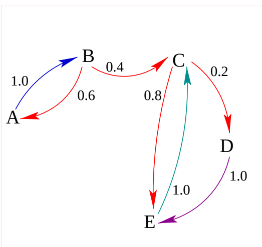
Stationary probabilities under a plan

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.

J	probability distribution initially
JM	after 1 action
JMM	after 2 actions
$JMMM$	after 3 actions
$\vdots$	
JM^i	after i actions

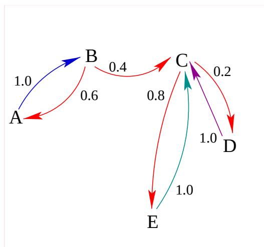
A probability distribution P over states $s_1, \ldots, s_n$ is represented as an n-element row vector $(P(s_1), \ldots, P(s_n))$.

Stationary probabilities under a plan



t	A	B	C	D	E
0	0.900	0.100	0.000	0.000	0.000
1	0.060	0.900	0.040	0.000	0.000
2	0.540	0.060	0.360	0.008	0.032
3	0.036	0.540	0.056	0.072	0.296
4	0.324	0.036	0.512	0.011	0.117
5	0.022	0.324	0.131	0.102	0.421
6	0.194	0.022	0.550	0.026	0.207
7	0.013	0.194	0.216	0.110	0.467
8	0.117	0.013	0.544	0.043	0.283
9	0.008	0.117	0.288	0.109	0.479
10	0.070	0.008	0.525	0.058	0.339
$\vdots$					
	0.000	0.000	0.455	0.091	0.455
	0.000	0.000	0.455	0.091	0.455

Probabilities of states under a plan (periodic)



t	A	B	C	D	E
0	0.900	0.100	0.000	0.000	0.000
1	0.060	0.900	0.040	0.000	0.000
2	0.540	0.060	0.360	0.008	0.032
3	0.036	0.540	0.064	0.072	0.288
4	0.324	0.036	0.576	0.013	0.051
5	0.022	0.324	0.078	0.115	0.461
6	0.194	0.022	0.706	0.016	0.063
7	0.013	0.194	0.087	0.141	0.564
$\vdots$					
	0.000	0.000	0.900	0.020	0.080
	0.000	0.000	0.100	0.180	0.720
	0.000	0.000	0.900	0.020	0.080
	0.000	0.000	0.100	0.180	0.720

Evaluation of performance

Average rewards

- A waiter/waitress robot
  - induces costs: cost of food and beverages brought to customers, broken plates and glasses, ...
  - brings rewards: collects money from customers.
- This can be viewed as an infinite sequence of rewards -6.0, 3.1, 6.9, -0.80, -1.2, 2.6, 12.8, -1.1, 2.1, -10.0,...
- Owner's objective: the plan the robot follows must maximize the 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 time point $i+1$ is discounted by a factor $\lambda \in]0..1[$ in comparison to reward at i 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 $\lim_{N\to\infty} \frac{\sum_{i=1}^{N} r_i}{N}$
3. discounted rewards $r_1 + \lambda r_2 + \lambda^2 r_3 + \ldots + \lambda^{k-1} r_k + \cdots$

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

Probabilistic planning with full observability

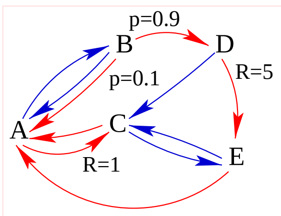
- Several algorithms:
  1. dynamic programming (finite executions)
  2. value iteration (discounted rewards, infinite execution)
  3. policy iteration (discounted rewards, infinite execution)
- Some of these algorithms can be easily implemented without explicitly representing the state space (e.g. by using algebraic decision diagrams ADDs).

Optimal rewards over a finite execution

- 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

Example

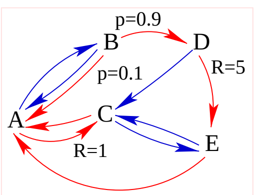


$$\begin{bmatrix} 0 & 0 & 1.0 & 0 & 0 \\ 0.1 & 0 & 0 & 0.9 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 1.0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1.0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 \end{bmatrix}$$

Optimal rewards over a finite execution

Example



i	$v_i(A)$	$v_i(B)$	$v_i(C)$	$v_i(D)$	$v_i(E)$
9	1.00	0.00	0.00	5.00	0.00
8	1.00	4.60	1.00	5.00	1.00
7	4.60	4.60	1.00	6.00	1.00
6	4.60	5.86	4.60	6.00	4.60
5	5.86	5.86	4.60	9.60	4.60
4	5.86	9.23	5.86	9.60	5.86
3	9.23	9.23	5.86	10.86	5.86
2	9.23	10.70	9.23	10.86	9.23
1	10.70	10.70	9.23	14.23	9.23

Optimal rewards over a finite execution

Algorithm

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

$$\begin{aligned} v_N(s) &= \max_{o \in O(s)} R(s,o) \\ v_i(s) &= \max_{o \in O(s)} \left(R(s,o) + \textstyle\sum_{s' \in S} p(s'|s,o) v_{i+1}(s')\right) \\ &\quad \text{for } i \in \{1, \ldots, N-1\} \end{aligned}$$

Optimal plans over a finite execution
Algorithm

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

$$\begin{aligned} \pi(s,N) &= \arg\max_{o \in O(s)} R(s,o) \\ \pi(s,i) &= \arg\max_{o \in O(s)} \left(R(s,o) + \sum_{s' \in S} p(s'|s,o) v_{i+1}(s')\right) \\ & \quad \text{for } i \in \{1, \ldots, N-1\} \end{aligned}$$

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.

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

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

The value iteration algorithm

- Value iteration is the simplest algorithm for finding close-to-optimal plans for infinite executions and discounted rewards.
- Idea:
  1. Start with an arbitrary value function.
  2. Compute better and better approximations of the optimal value function by using Bellman's equation.
  3. From a good approximation construct a plan.
- Plans extracted from a very-close-to-optimal value function are typically optimal.
- Parameter ϵ: Algorithm terminates when value function changes less than $\frac{\epsilon(1-\lambda)}{2\lambda}$: difference to optimal value function is $< \epsilon$.

The value iteration algorithm
Definition

1. $n := 0$
2. Choose any value function v_0.
3. For every $s \in S$

$$v_{n+1}(s) = \max_{o \in O(s)} \left(R(s,o) + \sum_{s' \in S} \lambda p(s'|s,o) v_n(s')\right).$$

   Go to step 4 if $|v_{n+1}(s) - v_n(s)| < \frac{\epsilon(1-\lambda)}{2\lambda}$ for all $s \in S$.
   Otherwise set $n := n + 1$ and go to step 3.
4. Construct a plan: for every $s \in S$

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

The value iteration algorithm
Properties

Theorem

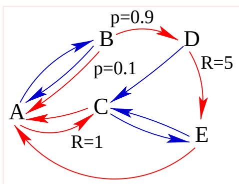
Let v_π be the value function of the plan produced by the value iteration algorithm, and let v^* be the value function of the optimal plan(s). Then $|v^*(s) - v_\pi(s)| \leq \epsilon$ for all $s \in S$.

Under full observability there is never a trade-off between the values of two states: if the optimal value for state s_1 is r_1 and the optimal value for state s_2 is r_2, then there is one plan that achieves these both.

Value iteration
Example

Let $\lambda = 0.6$.

i	$v_i(A)$	$v_i(B)$	$v_i(C)$	$v_i(D)$	$v_i(E)$
0	0.000	0.000	0.000	0.000	0.000
1	1.000	0.000	0.000	5.000	0.000
2	1.000	2.760	0.600	5.000	0.600
3	1.656	2.760	0.600	5.360	0.600
4	1.656	2.994	0.994	5.360	0.994
5	1.796	2.994	0.994	5.596	0.994
6	1.796	3.130	1.078	5.596	1.078
7	1.878	3.130	1.078	5.647	1.078
8	1.878	3.162	1.127	5.647	1.127
⋮					
19	1.912	3.186	1.147	5.688	1.147
20	1.912	3.186	1.147	5.688	1.147



The policy iteration 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.

Plan evaluation by solving linear equations

Given a plan π, its value v_π under discounted rewards with discount constant λ 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
Example

Consider the plan

$$\pi(A) = R, \pi(B) = R, \pi(C) = B, \pi(D) = R, \pi(E) = B$$

$$\begin{aligned}
v_\pi(A) &= R(A,R) + 0\lambda v_\pi(A) + 0\lambda v_\pi(B) + 1\lambda v_\pi(C) + 0\lambda v_\pi(D) + 0\lambda v_\pi(E)\\
v_\pi(B) &= R(B,R) + 0.1\lambda v_\pi(A) + 0\lambda v_\pi(B) + 0\lambda v_\pi(C) + 0.9\lambda v_\pi(D) + 0\lambda v_\pi(E)\\
v_\pi(C) &= R(C,B) + 0\lambda v_\pi(A) + 0\lambda v_\pi(B) + 0\lambda v_\pi(C) + 0\lambda v_\pi(D) + 1\lambda v_\pi(E)\\
v_\pi(D) &= R(D,R) + 0\lambda v_\pi(A) + 0\lambda v_\pi(B) + 0\lambda v_\pi(C) + 0\lambda v_\pi(D) + 1\lambda v_\pi(E)\\
v_\pi(E) &= R(E,B) + 0\lambda v_\pi(A) + 0\lambda v_\pi(B) + 1\lambda v_\pi(C) + 0\lambda v_\pi(D) + 0\lambda v_\pi(E)
\end{aligned}$$

$$\begin{aligned}
v_\pi(A) &= 1 + \lambda v_\pi(C)\\
v_\pi(B) &= 0 + 0.1\lambda v_\pi(A) + 0.9\lambda v_\pi(D)\\
v_\pi(C) &= 0 + \lambda v_\pi(E)\\
v_\pi(D) &= 5 + \lambda v_\pi(E)\\
v_\pi(E) &= 0 + \lambda v_\pi(C)
\end{aligned}$$

Plan evaluation by solving linear equations
Example

$$\begin{aligned}
v_\pi(A) - \lambda v_\pi(C) &= 1\\
-0.1\lambda v_\pi(A) + v_\pi(B) - 0.9\lambda v_\pi(D) &= 0\\
v_\pi(C) - \lambda v_\pi(E) &= 0\\
v_\pi(D) - \lambda v_\pi(E) &= 5\\
-\lambda v_\pi(C) + v_\pi(E) &= 0
\end{aligned}$$

Solving with $\lambda = 0.5$ we get

$$\begin{aligned}
v_\pi(A) &= 1\\
v_\pi(B) &= 2.3\\
v_\pi(C) &= 0\\
v_\pi(D) &= 5\\
v_\pi(E) &= 0
\end{aligned}$$

This is the value function of the plan.

The policy iteration algorithm
Definition

1. $n := 0$
2. Let π_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 $s \in S$

$$\pi_{n+1}(s) = \arg\max_{o \in O(s)} \left(R(s,o) + \sum_{s' \in S} \lambda p(s'|s,o) v_{\pi_n}(s') \right)$$

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

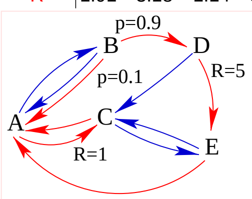
The policy iteration algorithm
Properties

Theorem
The policy iteration algorithm terminates after a finite number of steps and returns an optimal plan.

Proof idea.
There is only a finite number of different plans, and at each step a properly better plan is found or the algorithm terminates.
The number of iterations needed for finding an ϵ-optimal plan by policy iteration is never higher than the number of iterations needed by value iteration. □

The policy iteration algorithm
Example

itr.	$\pi(A)$	$\pi(B)$	$\pi(C)$	$\pi(D)$	$\pi(E)$	$v_\pi(A)$	$v_\pi(B)$	$v_\pi(C)$	$v_\pi(D)$	$v_\pi(E)$
1	R	R	R	R	R	1.56	3.09	0.93	5.56	0.93
2	B	R	R	R	R	1.91	3.18	1.14	5.68	1.14



- The previous three algorithms ignored the set of goal states and attempted to maximize the rewards.
- Reaching the goal states is an objective that may be combined with rewards and probabilities.
- Goal reachability with minimum costs and probability 1: Find a plan that guarantees reaching the goals with the minimum expected costs.
- Goal reachability with maximum probability: Find a plan that maximizes the probability that a goal state is reached.

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.

$$\begin{aligned}
v_0(s) &= -\infty \text{ for } s \in S \backslash G\\
v_0(s) &= 0 \text{ for } s \in G\\
v_{i+1}(s) &= \max_{o \in O(s)} \left(R(s,o) + \textstyle\sum_{s' \in S} p(s'|s,o) v_i(s') \right) \text{ for } s \in S \backslash G
\end{aligned}$$

This computation converges if for every ϵ 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.

$$\begin{aligned}
v_0(s) &= 0 \text{ for } s \in S \backslash G\\
v_0(s) &= 1 \text{ for } s \in G\\
v_{i+1}(s) &= \min_{o \in O(s)} \textstyle\sum_{s' \in S} p(s'|s,o) v_i(s') \text{ for } s \in S \backslash G
\end{aligned}$$

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 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.

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.

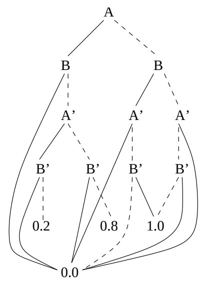
Problem: Algorithms do not use just sets and relations which can be represented as BDDs, but value functions $v : S \to \mathcal{R}$ and non-binary transition matrices.

Solution: Use a generalization of BDDs called algebraic decision diagrams (or MTBDDs: multi-terminal BDDs.)

Algebraic decision diagrams

- Graph representation of functions from $\{0,1\}^n \to \mathcal{R}$ that generalizes BDDs (functions $\{0,1\}^n \to \{0,1\}$)
- 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

An algebraic decision diagram



A mapping aba'b'$\to \mathcal{R}$

ab	$a'b'$ 00	$a'b'$ 01	$a'b'$ 10	$a'b'$ 11
00	1.0	0	0	0
01	0	1.0	0	0
10	0.8	0	0.2	0
11	0	0	0	0

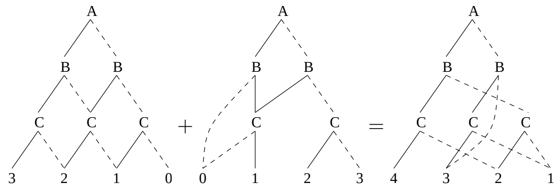
Operations on ADDs

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

abc	f	g	$f+g$	max(f,g)	$7 \cdot f$
000	0	3	3	3	0
001	1	2	3	2	7
010	1	0	1	1	7
011	2	1	3	2	14
100	1	0	1	1	7
101	2	0	2	2	14
110	2	0	2	2	14
111	3	1	4	3	21

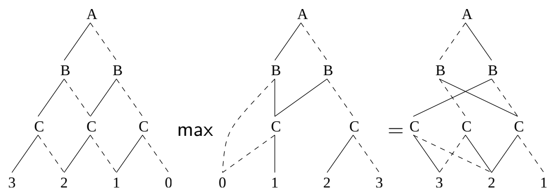
Operations on ADDs

Sum



Operations on ADDs

Maximum



Operations on ADDs

Arithmetic $\exists$ abstraction

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

abc	f
000	0
001	1
010	1
011	2
100	1
101	2
110	2
111	3

$\exists c.f$ is obtained by summing $f(x)$ and $f(x')$ when x and x' differ only on c:

ab	$\exists c.f$
00	1
01	3
10	3
11	5

Matrix multiplication with ADDs (I)

Consider matrices M_1 and M_2, represented as mappings:

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

aa'	M_1
00	1
01	2
10	3
11	4

$a'a''$	M_2
00	1
01	2
10	2
11	1

Matrix multiplication with ADDs (II)

$aa'a''$	M_1	M_2	$M_1 \cdot M_2$
000	1	1	1
001	1	2	2
010	2	2	4
011	2	1	2
100	3	1	3
101	3	2	6
110	4	2	8
111	4	1	4

aa''	$\exists a'.(M_1 \cdot M_2)$
00	5
01	4
10	11
11	10

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

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 $W(o), o \in O$.
- Functions v_i are ADDs that map valuations of A to $\mathcal{R}$.
- All computation is for all states (one ADD) simultaneously: big speed-ups possible.

Translation of operators into ADDs

Translation for operator $o = \langle c, e \rangle$ in normal form is $\tau_A^{\textit{prob}}(o) = c \wedge \tau_A^{\textit{prob}}(e)$ where

$$\begin{aligned} \tau_B^{\textit{prob}}(e) &= \tau_B(e) \text{ when } e \text{ is deterministic} \\ \tau_B^{\textit{prob}}(p_1 e_1 | \cdots | p_n e_n) &= p_1 \tau_B^{\textit{prob}}(e_1) + \cdots + p_n \tau_B^{\textit{prob}}(e_n) \\ \tau_B^{\textit{prob}}(e_1 \wedge \cdots \wedge e_n) &= \tau_{B \setminus (B_2 \cup \cdots \cup B_n)}^{\textit{prob}}(e_1) \cdot \tau_{B_2}^{\textit{prob}}(e_2) \cdot \ldots \cdot \tau_{B_n}^{\textit{prob}}(e_n) \\ & \quad \text{where } B_i = \textit{changes}(e_i) \text{ for all } i \in \{1, \ldots, n\} \end{aligned}$$

Nondeterministic choice and outermost conjunctions are respectively by arithmetic sum and multiplication.

Translation of reward functions into ADDs

- Let the rewards for $o = \langle c, e \rangle \in O$ be represented by $W(o) = \{\langle \phi_1, r_1 \rangle, \ldots, \langle \phi_n, r_n \rangle\}$.
- We construct an ADD R_o 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 $r_1, \ldots, r_n$:

$$R_o = r_1 \cdot \phi_1 + \cdots + r_n \cdot \phi_n - \infty \cdot \neg c$$

The value iteration algorithm without ADDs

1. $n := 0$
2. Choose any value function v_0.
3. For every $s \in S$

$$v_{n+1}(s) = \max_{o \in O(s)} \left(R(s, o) + \sum_{s' \in S} \lambda p(s'|s, o) v_n(s') \right).$$

   Go to step 4 if $|v_{n+1}(s) - v_n(s)| < \frac{\epsilon(1-\lambda)}{2\lambda}$ 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 o as product of $\tau_A^{\textit{prob}}(o)$ and v_n:

$$R_o + \lambda \left(\begin{array}{c|cccc} & a'b' & a'b' & a'b' & a'b' \\ ab & 00 & 01 & 10 & 11 \\ \hline 00 & \mathbf{1.0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 01 & \mathbf{0} & \mathbf{1.0} & \mathbf{0} & \mathbf{0} \\ 10 & \mathbf{0.2} & \mathbf{0} & \mathbf{0.8} & \mathbf{0} \\ 11 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array} \right) \left(\begin{array}{c|r} a'b' & v_n \\ \hline 00 & \mathbf{-5.1} \\ 01 & \mathbf{2.8} \\ 10 & \mathbf{10.2} \\ 11 & \mathbf{3.7} \end{array} \right)$$

Remark

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

The value iteration algorithm with ADDs

1. Assign $n := 0$ and let v_n be an ADD that is constant 0.
2. $$v_{n+1} := \max_{o \in O} \left(R_o + \lambda \cdot \exists A'.(\tau_A^{\textit{prob}}(o) \cdot (v_n[A'/A])) \right)$$
   Unsatisfied preconditions are handled by the immediate rewards $-\infty$.
3. If all terminal nodes of ADD $|v_{n+1} - v_n|$ are $< \frac{\epsilon(1-\lambda)}{2\lambda}$ then stop.
4. Otherwise, set $n := n + 1$ and repeat from step 2.

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.