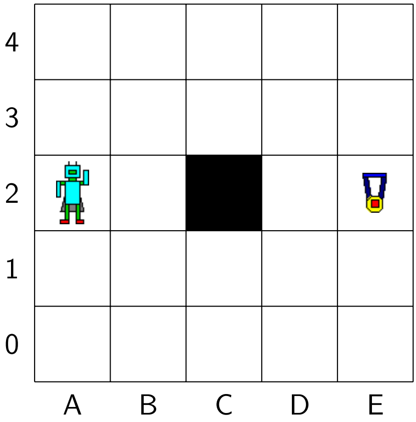
Advanced Topics on Artificial Intelligence

Alban Grastien

The Australian National University

Second Semester, 2020

Formally: Little Robot



- Four actions: Up, Down, Left, Right (50% chance of going Down-Right).
- Each action costs 1
- Reaching  ends the game.
- Trying to minimise total cost

Greedy Policy

Let $V : S \to \mathcal{Q}^+$ be a value function, that estimates the value of each state.

The greedy policy π_V chooses the action that guarantees the smallest expected cost according to a one-step lookahead:

$$\pi_V(s) \stackrel{def}{=} \arg\min_{a \in A(s)} \quad \Sigma_{s' \in S} \left(P(s,a,s') \cdot (C(s,a,s') + \gamma \times V(s')) \right)$$

(ties broken arbitrarily)

Little Robot: Greedy Policy

4	60	50	40	30	20
3	50	40	30	20	10
2	40	30		10	0
1	50	40	30	20	10
0	60	50	40	30	20
	A	B	C	D	E

- Cost of each action is 1. Reward is 0. $\gamma = 1$.
- Initialise value function with values defined from obstacle-free distance

Little Robot: Greedy Policy

4	60	50	40	30	20
3	50	40	30	20	10
2	40	30		10	0
1	50	40	30	20	10
0	60	50	40	30	20
	A	B	C	D	E

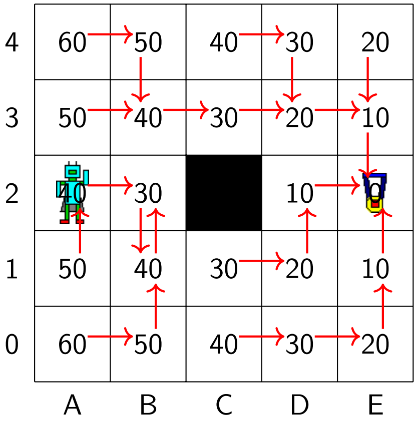
- Cost of each action is 1. Reward is 0. $\gamma = 1$.
- Initialise value function with values defined from obstacle-free distance
- Consider A2:
- Up: $C(Up) + V(A3) = 51$
- Right: $C(Right) + (.5\,V(B2) + .5\,V(B1)) = 36$
- Down: $C(Down) + V(A1) = 51$

Little Robot: Greedy Policy

4	60	50	40	30	20
3	50	40	30	20	10
2	 40	30		10	 0
1	50	40	30	20	10
0	60	50	40	30	20
	A	B	C	D	E

- Cost of each action is 1. Reward is 0. $\gamma = 1$.
- Initialise value function with values defined from obstacle-free distance
- Consider A2:
- Up: $C(Up) + V(A3) = 51$
- Right: $C(Right) + (.5\,V(B2) + .5\,V(B1)) = 36$
- Down: $C(Down) + V(A1) = 51$

Little Robot: Greedy Policy



- Cost of each action is 1. Reward is 0. $\gamma = 1$.
- Initialise value function with values defined from obstacle-free distance
- Consider A2:
- Up: $C(Up) + V(A3) = 51$
- Right: $C(Right) + (.5\,V(B2) + .5\,V(B1)) = 36$
- Down: $C(Down) + V(A1) = 51$

Bellmann Backup

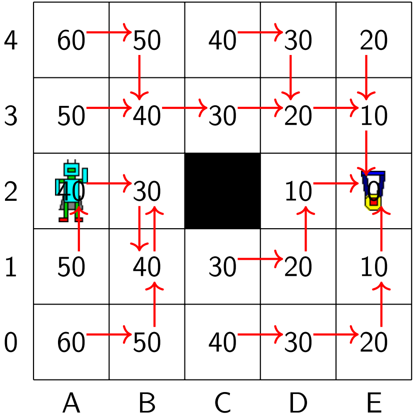
- The idea of Bellmann backup is to update the value of each state with its lookahead.
- It is expected that the new value will be more accurate.

Given value function V, the Bellmann backup is a new value function, BV, defined as the cost of the next action + the (discounted) value of the next state, weighted by the outcome of the action

If s is a goal state, then $BV(s) \stackrel{def}{=} 0$. Otherwise:

$$BV(s) \stackrel{def}{=} \min_{a \in A(s)} \Sigma_{s' \in S} P(s, a, s') \Big(C(s, a, s') + \gamma \cdot V(s') \Big)$$

Little Robot: Bellmann Backup

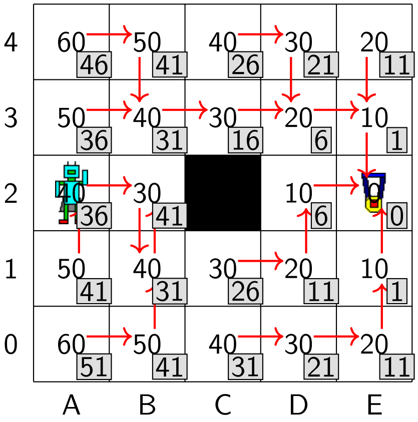


- $\gamma = 1$
- Cost of actions is 1

then:

- $BV(E1) = C(Up) + V(E2) = 1$
- $BV(D2) = C(Right) + (.5\,V(E2) + .5\,V(E1)) = 6$

Little Robot: Bellmann Backup



- $\gamma = 1$
- Cost of actions is 1

then:

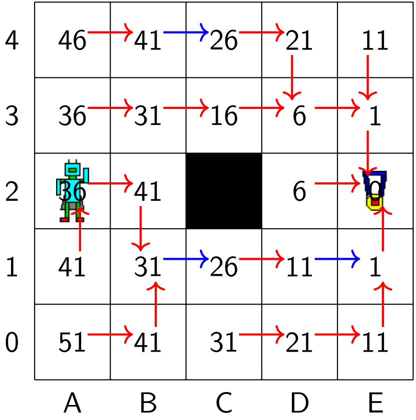
- $BV(E1) = C(Up) + V(E2) = 1$
- $BV(D2) = C(Right) + (.5\,V(E2) + .5\,V(E1)) = 6$

Little Robot: Greedy Backup Policy

4	46	41	26	21	11
3	36	31	16	6	1
2	36	41		6	0
1	41	31	26	11	1
0	51	41	31	21	11
	A	B	C	D	E

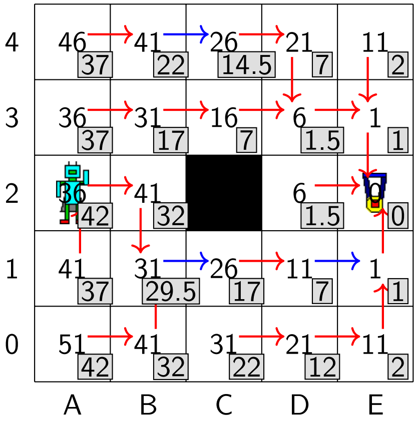
- Replacing V with BV
- If several actions are tied in a state, we stick to the same policy.

Little Robot: Greedy Backup Policy



- Replacing V with BV
- If several actions are tied in a state, we stick to the same policy.
- In blue: the new decisions.

Little Robot: Greedy Backup Policy

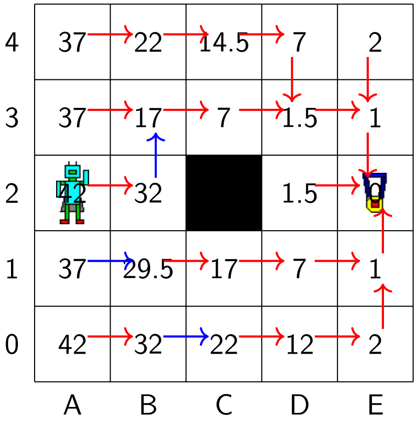


- Replacing V with BV
- If several actions are tied in a state, we stick to the same policy.
- In blue: the new decisions.

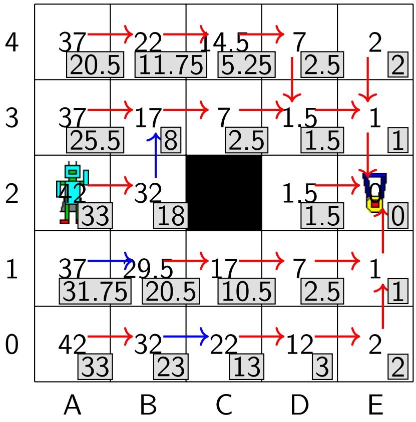
Little Robot: Going On

4	37	22	14.5	7	2
3	37	17	7	1.5	1
2	 42	32		1.5	 0
1	37	29.5	17	7	1
0	42	32	22	12	2
	A	B	C	D	E

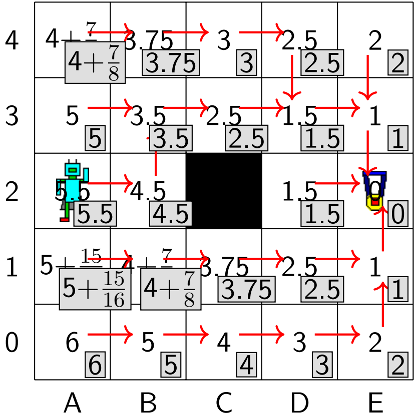
Little Robot: Going On



Little Robot: Going On



Little Robot: Eventually



Bellmann Backup and Perfect Value Function V^*

Assuming either $\gamma \neq 1$ or every action has a positive cost except in goal states.

Theorem

V^* enjoys the following property:

$$\forall s \in S. \quad BV^*(s) = V^*(s)$$

Furthermore, $V^*(s)$ is the only value function that enjoys this property.

Proof (sketch)

Assume $V_1 \neq V_2$ and yet $BV_1 = V_1$ and $BV_2 = V_2$. Assume that all actions have a positive cost. Assume $\gamma = 1$.

1. Let s be one state that maximises $\delta := V_2(s)/V_1(s)$ where $\delta > 1$.
2. Let a be the optimal action in s according to V_1.
3. We have

$$
\begin{aligned}
V_2(s) &\leq Q_2(s,a) \\
&\leq \Sigma_{s' \in S} \left(P(s,a,s') \times (C(s,a,s') + V_2(s')) \right) \\
&\leq \Sigma_{s' \in S} \left(P(s,a,s') \times (C(s,a,s') + \delta V_1(s')) \right) \\
&< \Sigma_{s' \in S} \left(P(s,a,s') \times (\delta C(s,a,s') + \delta V_1(s')) \right) \\
&< \delta \Sigma_{s' \in S} \left(P(s,a,s') \times (C(s,a,s') + V_1(s')) \right) \\
&< \delta V_1(s)
\end{aligned}
$$

Bellmann Error

The **Bellmann error** of a value function is the maximum absolute difference before the value of a state and the backup value of the same state:

$$BE(V) \stackrel{def}{=} \max_s \quad |BV(s) - V(s)|$$

Theorem

The Bellmann error of a function is 0 iff this function is the perfect value function.

Iterative Backup

- For an SSP derived from an MDP with discount factor $\gamma \in [0, 1)$, the Bellmann error satisfies the following inequality:

$$BE(BV) \leq (1 - \gamma) BE(V).$$

- In other words, V^* can be obtained by applying infinitely many backups to any value function:

$$V^* = \lim_{i \to \infty} \underbrace{B \dots B}_{i} V.$$

- If the Bellmann error of V is below $\frac{\varepsilon(1-\gamma)}{2\gamma}$, then V is within ε of V^*:

$$V^*(s) \in [V(s) - \varepsilon, V(s) + \varepsilon] \quad \text{for all state } s.$$

Value Iteration

If the goal is reachable with a non-zero percent chance from any state, then $\underbrace{B \dots B}_{i} V$ converges as i increases.

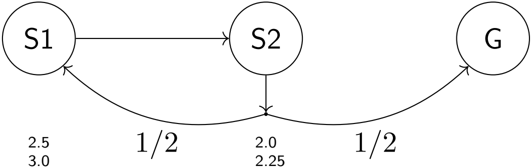
Value-Iteration

- Input: small number ε
- $t := 0$
- Choose an arbitrary value function V^t
- **repeat**
  - $t \mathrel{+}= 1$
  - $V^t := BV^{t-1}$
- **while** $BE(V^{t-1}) > \varepsilon$
- **return** V^t

Convergence

Notice that in general, the optimal value V^* is never reached. Example:

- there is only one action per state;
- $P(S1, a, S2) = 1$, $P(S2, a, S1) = P(S2, a, G) = 1/2$.
- the goal is G (value 0) and each action has cost 1.



2.5	2.0
3.0	2.25
3.25	2.5
3.5	2.625
3.625	2.75
3.75	2.8125
3.8125	2.875
3.875	2.90625
3.90625	2.9375
3.9375	2.953125

Properties and Drawbacks

- Fairly expensive
- Diverges when there are dead-ends
- Struggles to find the best first action (but good at finding the last good action)
- Allows for asynchronous backups $\rightarrow$ more about this later