Advanced Topics on Artificial Intelligence

Alban Grastien

The Australian National University

Second Semester, 2020

Moving on

- Yesterday:
  - Policy Iteration (PI)
  - how to create/represent MDPs
- Today: more sophisticated algorithms
  - RTDP
  - LAO*

Real-Time Dynamic Programming RTDP

and variants

Asynchronous VI (aka Find & Revise)

Asynchronous VI

- Choose an arbitrary value function $V : S \to \mathbb{R}$
- **repeat**
  - Choose state s
  - $V[s] := \min_{a \in A(s)} \Sigma_{s' \in S} P(s, a, s') \times \Big(C(s, a, s') + \gamma V[s'] \Big)$
- **until** some condition holds

Compared to Value Iteration:

- At each iteration, VI backs up all states ($V^{t+1} := BV$)
- At each iteration, A-VI backs up the value for only one state

Correctness

When is Asynchronous VI correct?

1. If no state is starved, then Asynchronous VI converges to V^*
   - A state is starved if it will eventually never be backed up
2. The states that are not reachable do not need to be backed up
3. If the value of state s does not change when backed up, you may also not back it up
4. If a state is provably not reachable under the optimal policy, then it does not need to be backed-up (more on that later).

Little Robot: Asynchronous VI

	A	B	C	D	E
4	0	0	0	0	0
3	0	0	1	0	0
2	0	0		0	100
1	0	50	0	0	0
0	0	0	0	0	0

- Reward of getting to $= 100$
- Discount factor $= .9$

- The value of which state should we rather backup?

Little Robot: Asynchronous VI

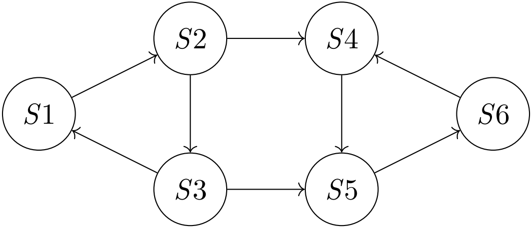
4	0	0	0	0	0
3	0	0	1	0	0
2	0	0		0	100
1	0	50	0	0	0
0	0	0	0	0	0
	A	B	C	D	E

- Reward of getting to $= 100$
- Discount factor $= .9$

- The value of which state should we rather backup?

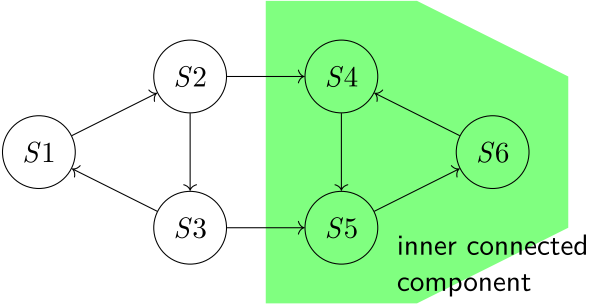
Topological VI

- Identifies connected components.
- Backs up the value of the inner components first.



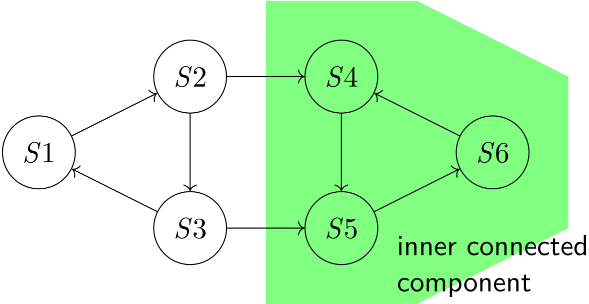
Topological VI

- Identifies connected components.
- Backs up the value of the inner components first.



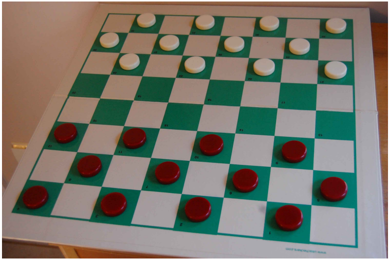
Topological VI

- Identifies connected components.
- Backs up the value of the inner components first.



- Occurs in problems with limited resources → cf. Tinsley vs Chinook

English Draughts, aka American Checkers



(source Wikipedia)

- A game where you try to take all pieces from the opponent.
- If a piece is taken, it cannot reappear.

→ The positions with k pieces forms one or several connected components.

- The game was solved in 2007 at the Uni. of Alberta. It took 18 years.
- The researchers who solved it started from positions with 4 pieces, up.

Improving Asynchronous VI: Monotonicity

- $V(s) \leq V^*(s)$ for all $s \Rightarrow V(s) \leq BV(s) \leq V^*(s)$ for all s
- $V(s) \geq V^*(s)$ for all $s \Rightarrow V(s) \geq BV(s) \geq V^*(s)$ for all s

- The **policy graph** of π is the set of states reachable while following π.

- If
  1. the state s does not appear in the policy graph of π_V and
  2. $V(s') \leq V^*(s')$ for all states s' (in a minimisation context)

  then it is not necessary to backup this state (currently).

Little Robot: Monotonicity

	A	B	C	D	E
4	4.62	3.75	3	2.5	2
3	5	3.5	2.5	1.5	1
2	5.68	4.5		1.5	0
1	5.93	4.87	3.75	2.5	1
0	6	5	4	3	2

- Cost once in  $= 0$
- Cost of actions: 1

Little Robot: Monotonicity

	A	B	C	D	E
4	4.62	3.75	3	2.5	2
3	5	3.5	2.5	1.5	1
2	5.68	4.5		1.5	0
1	5.93	4.87	3.75	2.5	1
0	6	5	4	3	2

- Cost once in  $= 0$
- Cost of actions: 1
- Purple: policy graph

Little Robot: Monotonicity

	A	B	C	D	E
4	4.62 **4.5**	3.75 **3.65**	3 **2.75**	2.5 **2.25**	2 **1.5**
3	5 **4.8**	3.5	2.5	1.5	1
2	5.68	4.5		1.5	0
1	5.93 **5.5**	4.87	3.75	2.5	1
0	6 **5**	5 **4.5**	4	3	2

- Cost once in  $= 0$
- Cost of actions: 1

- Purple: policy graph
- Red: Estimates of the policy value for non-policy states. If we know that the red values are underestimate, it is not necessary to backup these states.

Real-Time Dynamic Programming

- RTDP builds on these properties.
- It only explores and backs up the current policy graph.
- Using an admissible heuristic, it can safely ignore the other states.

Real-Time Dynamic Programming

RTDP

- $V := h$
- Perform n trials

Trial:

- $s := s_0$
- **while** $s \notin G$
  - Backup state s:

    $V(s) := \min_{a \in A(s)} \Sigma_{s' \in S}\ P(s, a, s') \times \Big(C(s, a, s') + \gamma V(s') \Big)$
  - Choose greedy action:

    $a := \arg\min_a\ \Sigma_{s' \in S}\ P(s, a, s') \times \Big(C(s, a, s') + \gamma V(s') \Big)$
  - $s' := simulate(s, a)$ // i.e., outcome of a is randomised

Properties

- A heuristic is $\boxed{\text{admissible}}$
  - if it underestimates when you try to minimise cost
  - if it overestimates when you try to maximise reward.

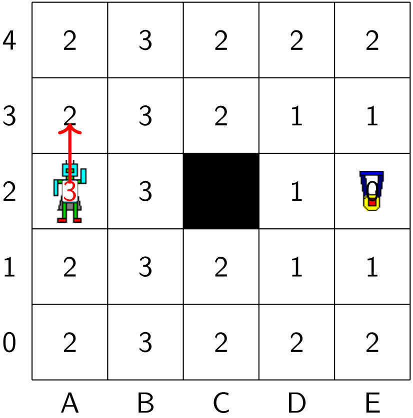
- If the heuristic h is admissible, then RTDP converges towards the optimal policy

Illustrating RTDP

	A	B	C	D	E
4	2	3	2	2	2
3	2	3	2	1	1
2	2	3		1	0
1	2	3	2	1	1
0	2	3	2	2	2

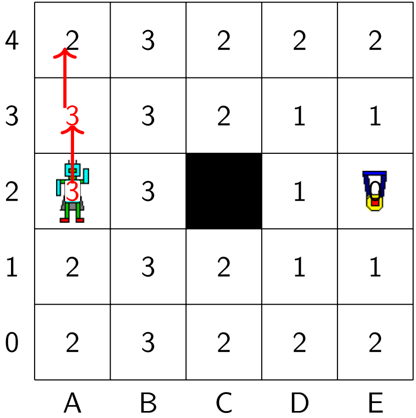
- Start with an **admissible heuristic**.

Illustrating RTDP



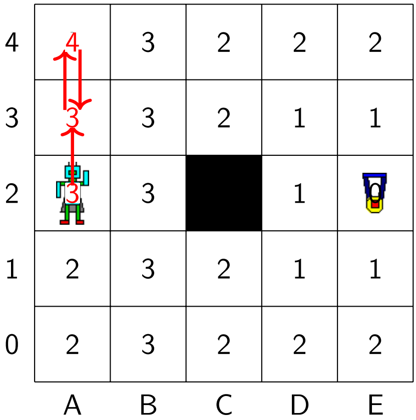
- Start with an **admissible heuristic**.
- Start trial 1:
  - Update A2, choose best action (here: Up or Down), and simulate it

Illustrating RTDP



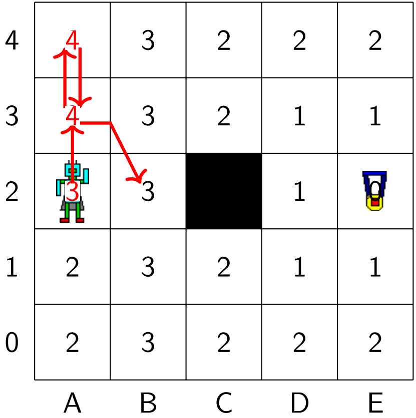
- Start with an **admissible heuristic**.
- Start trial 1:
  - Update A2, choose best action (here: Up or Down), and simulate it
  - Update A3, and choose best action (here: Up), and simulate it

Illustrating RTDP



- Start with an **admissible heuristic**.
- Start trial 1:
  - Update A2, choose best action (here: Up or Down), and simulate it
  - Update A3, and choose best action (here: Up), and simulate it
  - Update A4, and choose best action (here: Down or Right), and simulate it

Illustrating RTDP



- Start with an **admissible heuristic**.
- Start trial 1:
  - Update A2, choose best action (here: Up or Down), and simulate it
  - Update A3, and choose best action (here: Up), and simulate it
  - Update A4, and choose best action (here: Down or Right), and simulate it
  - Update A3, and choose best action (here: Down or Right), and simulate it (here, leads to B2)

RTDP's Performance

- RTDP is a "good" any-time algorithm:
  - Compared to VI or PI, it finds a good (sub-optimal) policy early
- By using the greedy policy, RTDP concentrates on the states that are likely to be in the optimal policy graph
- By using simulation, RTDP concentrates on the states that are more likely to actually be reached

L-RTDP[1]

- Asynchronous VI is very efficient when it does not backup the states that have a good estimate.

- A state is $\boxed{\text{solved}}$ if all states s' in the greedy space rooted at s have a Bellman error below ε.
- Backing up these states will not improve (significantly) the policy, while being time-consuming.

- L-RTDP identifies these solved states, and avoids them when simulating the effect of the action.
- Doing this is **not** unbiased (as opposed to Reinforcement Learning).

[1] B. Bonet and H. Geffner. "Labeled RTDP: improving the convergence of real-time dynamic programming". In: *13th International Conference on Automated Planning and Scheduling*. 2003, pp. 12–21.

Example

4	4.5	3.65	2.75	2.25	1.5
3	4.8	3.48	2.49	1.49	1
2	5.2	4.48		1.5	0
1	5	4.5	3.4	2.2	1
0	5	4.5	4	3	2
	A	B	C	D	E

- Current values

Example

4	4.5	3.65	2.75	2.25	1.5
3	4.8	3.48	2.49	1.49	1
2	 5.2	4.48		1.5	0
1	5	4.5	3.4	2.2	1
0	5	4.5	4	3	2
	A	B	C	D	E

- Current values
- Solved states

Example

4	4.5	3.65	2.75	2.25	1.5
3	4.8	3.48	2.49	1.49	1
2	5.2	4.48		1.5	0
1	5	4.5	3.4	2.2	1
0	5	4.5	4	3	2
	A	B	C	D	E

- Current values
- Solved states
- Because B2 is solved, the effect of Right in A2 will always lead to B1 during simulation.

Remark

- Determining what states are solved is not a trivial task
- Search for Tarjan's algorithm

Bounding the policy

- It sometimes happens that the policy is not settled in parts of the greedy space that have a small probability of being reached.
- “What **will** I do if I end up in this situation that is unlikely to happen?”
  - I don’t need to answer this question to decide the optimal action **now**.

Bounded-RTDP[2]

- Bounded-RTDP keeps two value functions V_ℓ and V_u such that $V_\ell(s) \leq V^*(s) \leq V_u(s)$ for all states.
  - Remember: the monotonicity property works if V is an upper bound or a lower bound
- Bounded-RTDP avoids states such that $(V_u(s) - V_\ell(s))/V_u(s) \leq \tau$ for some τ.
- Problem: How to compute (useful) upper bounds of $V^*(s)$?
  - It's harder than for admissible heuristics
  - Look at the paper for hints

[2] H. B. McMahan, M. Likhachev, and G. Gordon. "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees". In: *22nd International Conference on Machine Learning (ICML-05)*. 2005, pp. 569–576.