Planning and Optimization

G3. Heuristic Search: Real-Time Dynamic Programming

Gabriele Röger and Thomas Keller

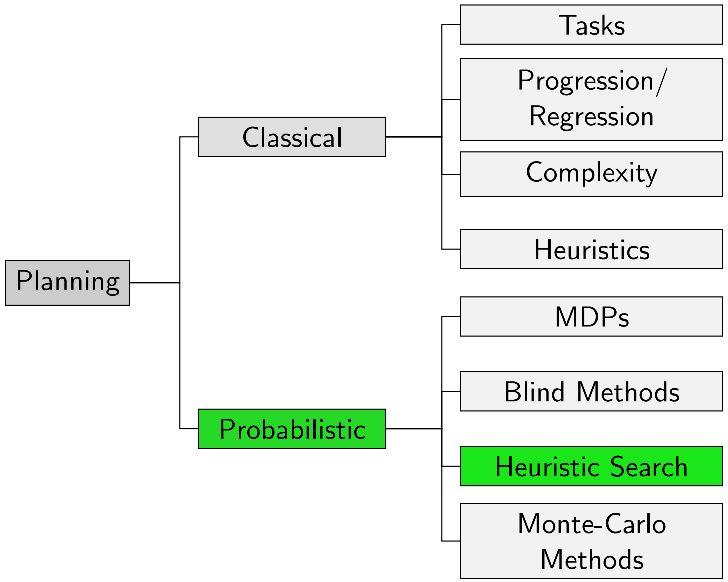
Universität Basel

December 5, 2018

Planning and Optimization

December 5, 2018 — G3. Heuristic Search: Real-Time Dynamic Programming

Content of this Course



G3.1 Motivation

Motivation

- AO* and LAO* find optimal solution without considering all states
- Value iteration has to repeatedly backup all states
- But VI computes complete policy, while AO* and LAO* compute executable policy for a given initial state
- And AO* and LAO* require admissible heuristic for guidance

⇒ Is this also possible for a VI-like algorithm if we provide it with admissible heuristic and accept executable policy as result?

G3.2 Asynchronous VI

Asynchronous Value Iteration

- Updating all states simultaneously is called synchronous backup
- Asynchronous VI performs backups for individual states
- Different approaches lead to different backup orders
- Can significantly reduce computation
- Guaranteed to converge if all states are selected repeatedly

⇒ Optimal VI with asynchronous backups possible

Example: Asynchronous Value Iteration

	1	2	3	4
5	4.49	2.0	1.0	$s_\star$ 0.0
4	5.49	3.0	8.49	2.49
3	6.49	4.0	5.0	4.98
2	8.98	6.49	6.0	7.47
1	s_0 8.49	7.49	7.0	9.49

$V^\star$

- cost of 1 for all actions except for moving away from (3,4) where cost is 3
- get stuck when moving away from gray cells with prob. 0.6

Example: Asynchronous Value Iteration

5	4.49	2.0	1.0	$s_\star$ 0.0
4	5.49	3.0	★ 8.49	2.49
3	6.49	4.0	5.0	4.98
2	8.98	6.49	6.0	7.47
1	s_0 8.49	7.49	7.0	9.49
	1	2	3	4

$\hat{V}^{41} \approx V^\star$

Demo: Result for VI variant that performs backup on each state with probability 0.5

In-place Value Iteration

- Synchronous value iteration creates new copy of value function (two are required simultaneously)

$$\hat{V}^{i+1}(s) := \min_{\ell \in L(s)} c(\ell) + \sum_{s' \in S} T(s, \ell, s') \cdot \hat{V}^i(s')$$

- In-place value iteration only requires a single copy of value function

$$\hat{V}(s) := \min_{\ell \in L(s)} c(\ell) + \sum_{s' \in S} T(s, \ell, s') \cdot \hat{V}(s')$$

- In-place VI is asynchronous because some backups are based on "old" values, some on "new" values

Example: In-place Value Iteration

5	4.49	2.0	1.0	$s_\star$ 0.0
4	5.49	3.0	★ 8.49	2.49
3	6.49	4.0	5.0	4.98
2	8.98	6.49	6.0	7.47
1	s_0 8.49	7.49	7.0	9.49
	1	2	3	4

$\hat{V}^{18} \approx V^\star$

Demo: Result for in-place value iteration

G3.3 Real-Time Dynamic Programming

Motivation: Real-Time Dynamic Programming

- Asynchronous VI still requires to backup all states repeatedly for optimality
- Real-Time Dynamic Programming (RTDP) uses admissible heuristic
- for optimal policy
- that is executable in initial state
- Proposed by Barto, Bradtke & Singh (1995)

Real-Time Dynamic Programming

- RTDP updates only states relevant to the agent
- Originally motivated from agent that acts in environment
- by following greedy policy w.r.t. current state-value estimates.
- Performs Bellman backup in each encountered state
- Uses admissible heuristic for states not updated before

Trial-based Real-Time Dynamic Programming

- We consider the offline version here
  $\Rightarrow$ interaction with environment is simulated in trials
- in real world, outcome of action application cannot be chosen
  $\Rightarrow$ in simulation, outcomes are sampled according to probabilities

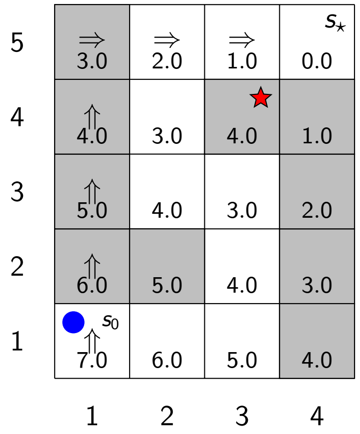
Real-Time Dynamic Programming

RTDP for SSP $\mathcal{T}$

while more trials required:
  $s := s_0$
  while $s \notin S_\star$:
    $\hat{V}(s) := \min_{\ell \in L(s)} c(\ell) + \sum_{s' \in S} T(s, \ell, s') \cdot \hat{V}(s')$
    $s :\sim \text{succ}(s, a_{\hat{V}}(s))$

Note: If $\hat{V}(s')$ is used on the right hand side of line 4 or 5 but has has not been assigned (by line 4) before, $h(s)$ is used instead

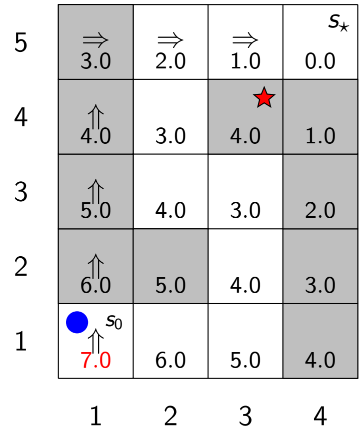
Example: RTDP



Before 1st trial

Used heuristic: shortest path assuming agent never gets stuck

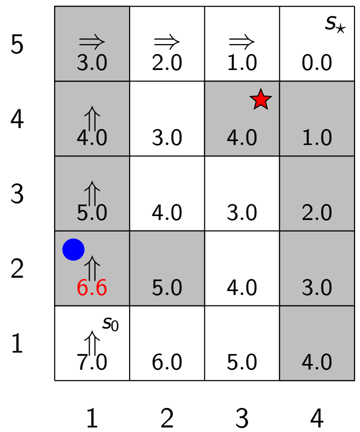
Example: RTDP



Step 1

Used heuristic: shortest path assuming agent never gets stuck

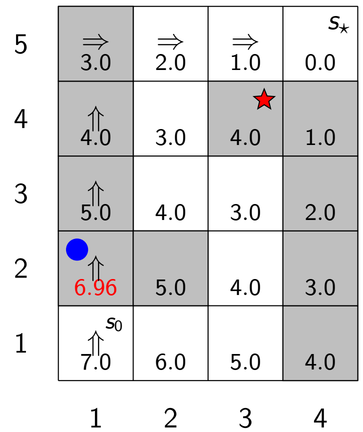
Example: RTDP



Step 2

Used heuristic: shortest path assuming agent never gets stuck

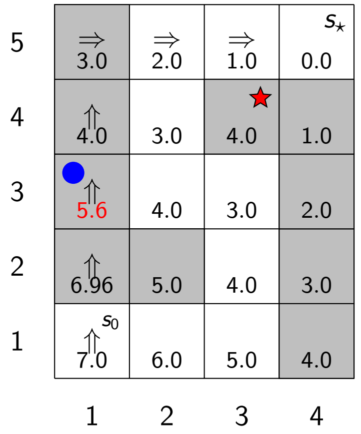
Example: RTDP



Step 3

Used heuristic: shortest path assuming agent never gets stuck

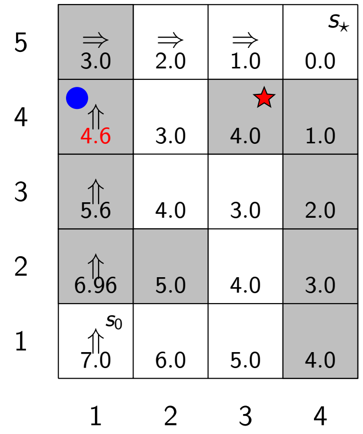
Example: RTDP



Step 4

Used heuristic: shortest path assuming agent never gets stuck

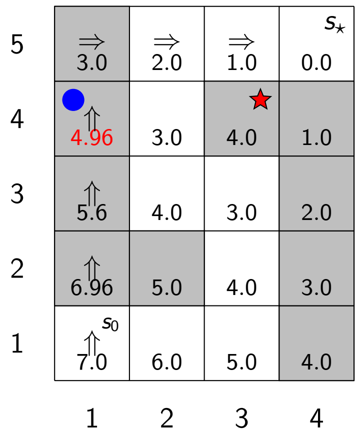
Example: RTDP



Step 5

Used heuristic: shortest path assuming agent never gets stuck

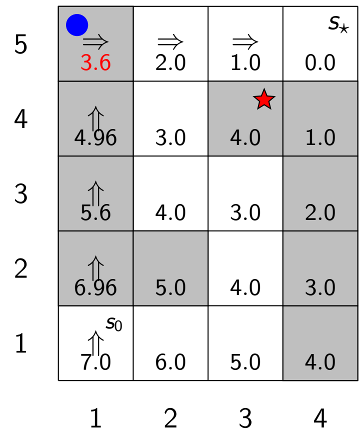
Example: RTDP



Step 6

Used heuristic: shortest path assuming agent never gets stuck

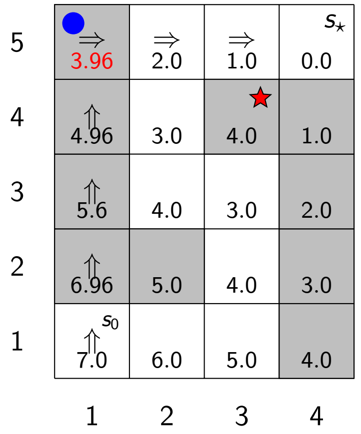
Example: RTDP



Step 7

Used heuristic: shortest path assuming agent never gets stuck

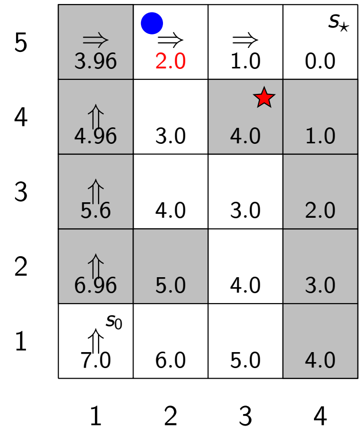
Example: RTDP



Step 8

Used heuristic: shortest path assuming agent never gets stuck

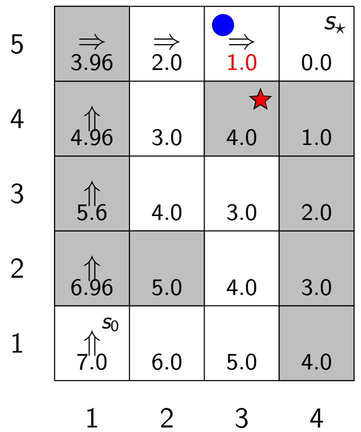
Example: RTDP



Step 9

Used heuristic: shortest path assuming agent never gets stuck

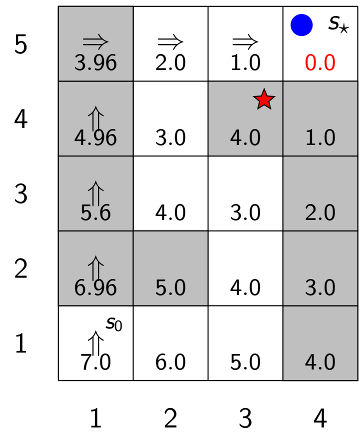
Example: RTDP



Step 10

Used heuristic: shortest path assuming agent never gets stuck

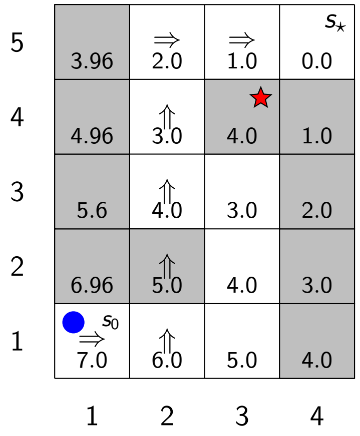
Example: RTDP



End of 1st trial

Used heuristic: shortest path assuming agent never gets stuck

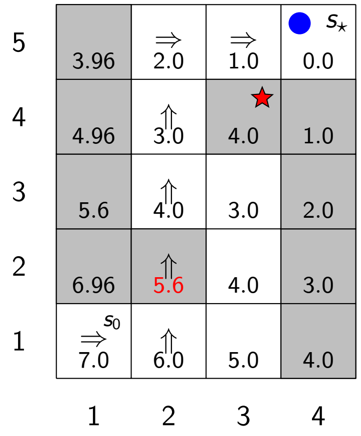
Example: RTDP



Before 2nd trial

Used heuristic: shortest path assuming agent never gets stuck

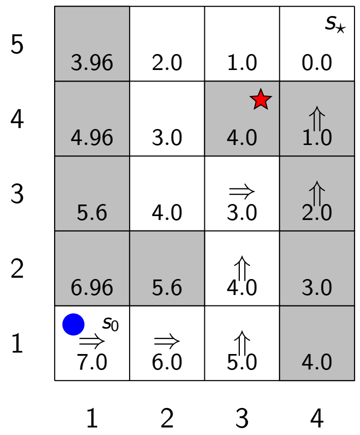
Example: RTDP



End of 2nd trial

Used heuristic: shortest path assuming agent never gets stuck

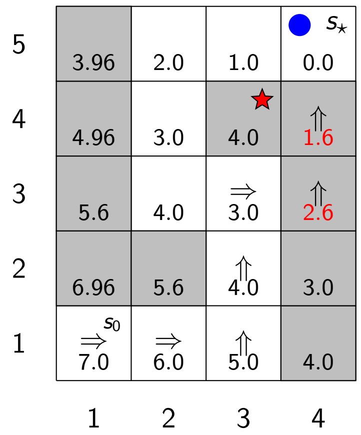
Example: RTDP



Before 3rd trial

Used heuristic: shortest path assuming agent never gets stuck

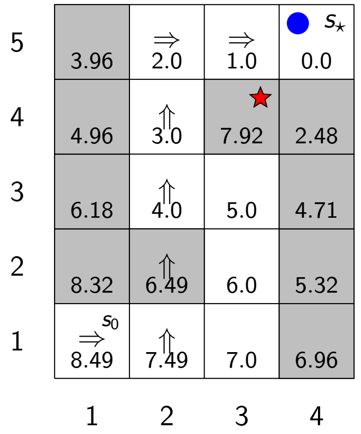
Example: RTDP



End of 3rd trial

Used heuristic: shortest path assuming agent never gets stuck

Example: RTDP



End of 13th trial

Used heuristic: shortest path assuming agent never gets stuck

RTDP: Theoretical Properties

Theorem

Using an admissible heuristic, RTDP converges to an optimal solution without (necessarily) computing state-value estimates for all states.

Proof omitted.

G3.4 Labeled Real-Time Dynamic Programming

Motivation

Issues of RTDP:

- states are updated after state-value estimate has converged
- no termination criterion $\Rightarrow$ algorithm is underspecified

Most popular algorithm to overcome these shortcomings:
Labeled RTDP (Bonet & Geffner, 2003)

Labeled RTDP: Idea

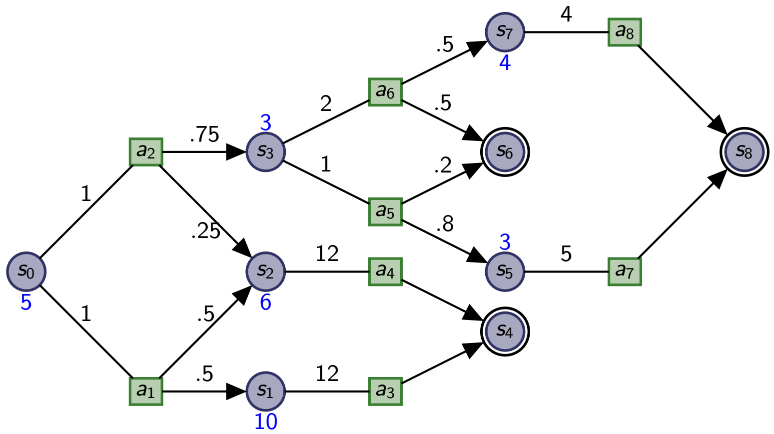
The main idea of Labeled RDTP is to label states as solved

- Labeling procedure different for cycylic and acyclic SSPs (following slides)
- Each trial terminates when solved state is encountered
  $\Rightarrow$ solved states no longer updated
- LRTDP terminates when the initial state is labeled as solved
  $\Rightarrow$ well-defined termination criterion

Solved States in Acyclic SSPs

- In acyclic SSPs, a state s is solved if
  - s is a goal state, or
  - all successor states of the greedy action $a_{\hat{V}}(s)$ are solved
- States are labeled as solved via backward induction

Labeled RTDP: Acyclic Example (Blackboard)



$h(s) = 0$ for goal states, otherwise in blue above or below s

Solved States in SSPs with Cycles

- States are solved if the difference of the state-value estimate to the Q-value of the greedy action (the residual) is small
- In presence of cycles, all states in strongly connected component must be solved simultaneously
- Labeled RTDP uses sub-algorithm `CheckSolved` to check if all states in SCC are solved

CheckSolved Procedure

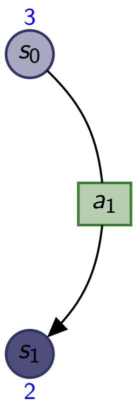
- CheckSolved is called on all states that were encountered in a trial in reverse order
- CheckSolved checks the residual of all states reachable under the greedy policy and
- labels all those states as solved if the residual is smaller than some ϵ
- Otherwise, CheckSolved performs (additional) backup on reachable states for faster convergence

Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

visited: s_0

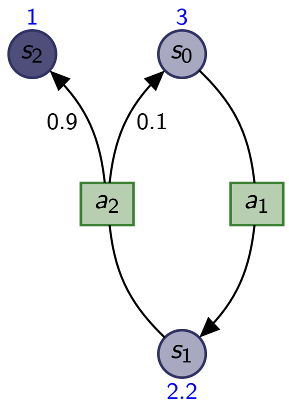
Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

visited: s_0, s_1



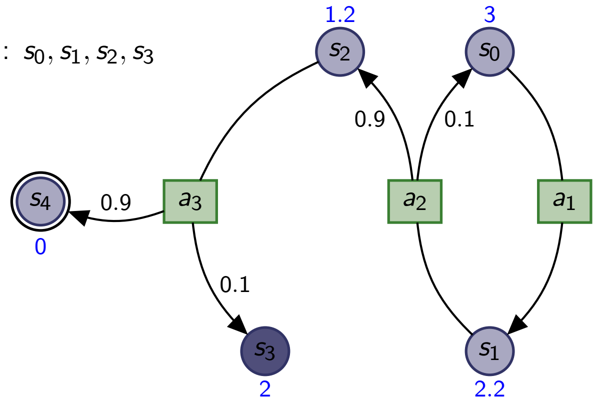
Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

visited: s_0, s_1, s_2



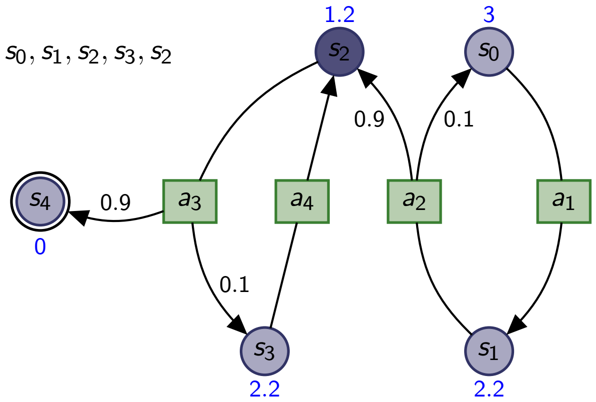
Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

visited: s_0, s_1, s_2, s_3



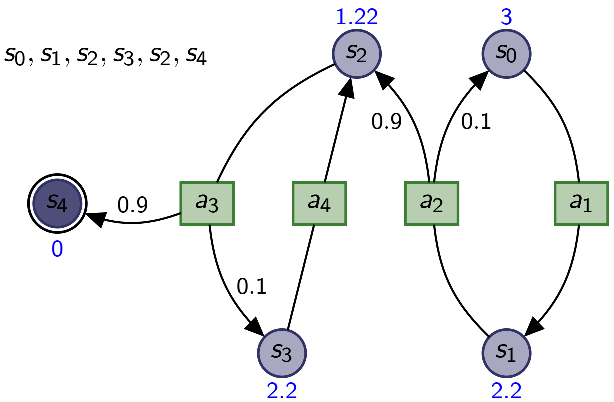
Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

visited: s_0, s_1, s_2, s_3, s_2



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

visited: $s_0, s_1, s_2, s_3, s_2, s_4$

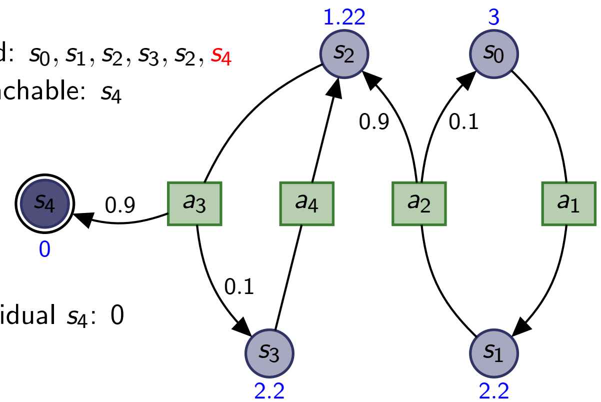


Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$

reachable: s_4

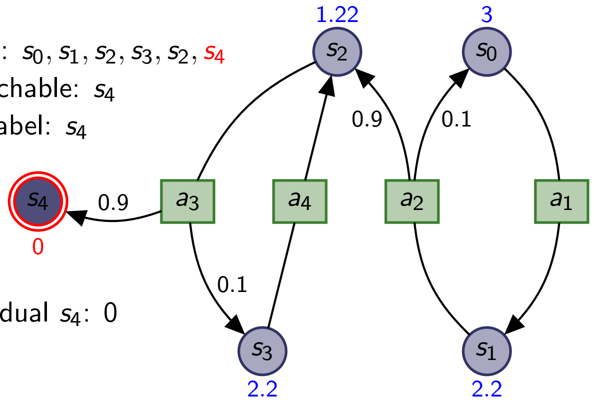
residual s_4: 0



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$
reachable: s_4
label: s_4

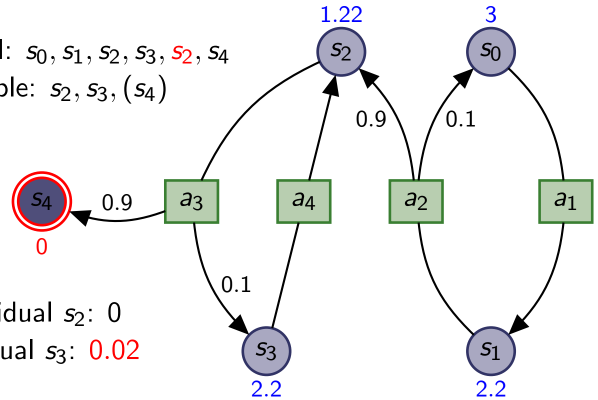
residual s_4: 0



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

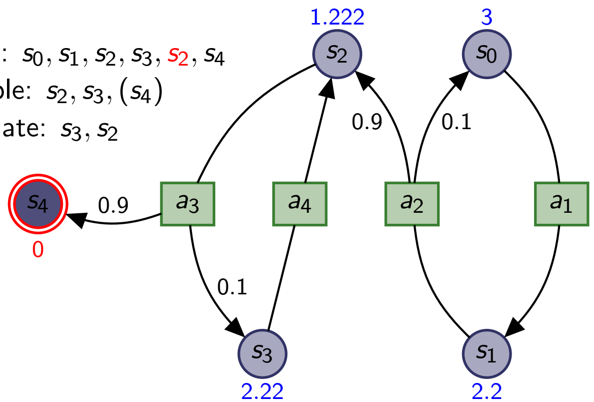
check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$
reachable: $s_2, s_3, (s_4)$

residual s_2: 0
residual s_3: 0.02



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

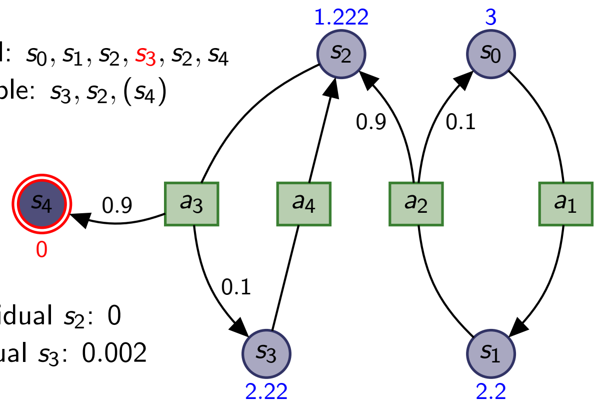
check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$
reachable: $s_2, s_3, (s_4)$
update: s_3, s_2



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$
reachable: $s_3, s_2, (s_4)$

residual s_2: 0
residual s_3: 0.002

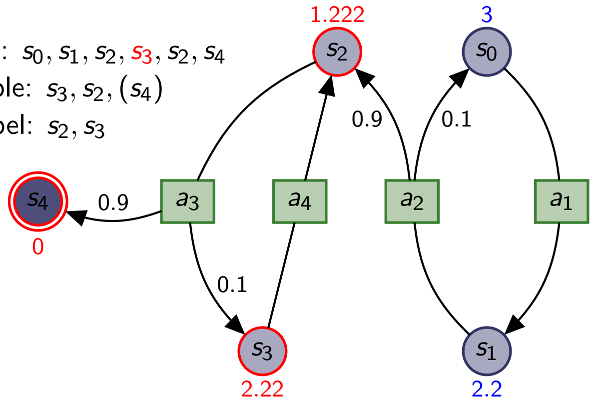


Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$

reachable: $s_3, s_2, (s_4)$

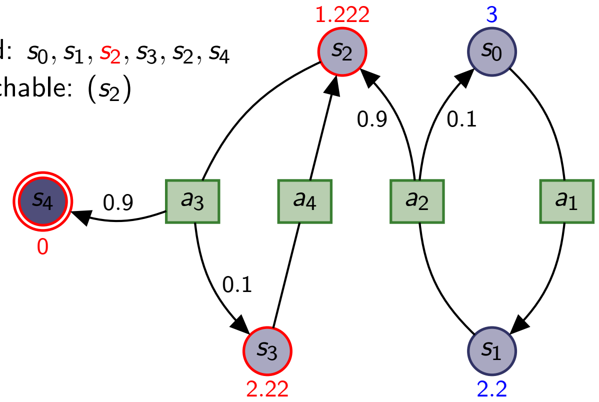
label: s_2, s_3



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$

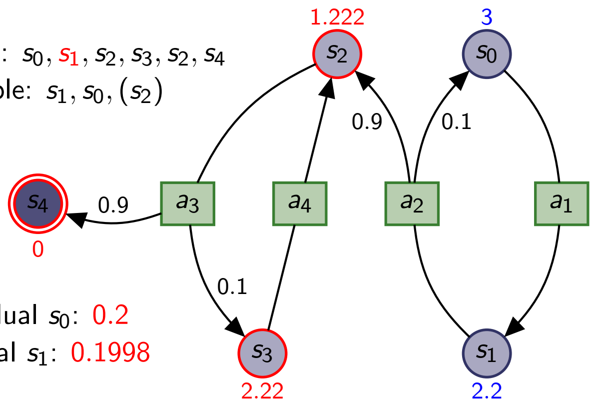
reachable: (s_2)



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$

reachable: $s_1, s_0, (s_2)$



residual s_0: 0.2

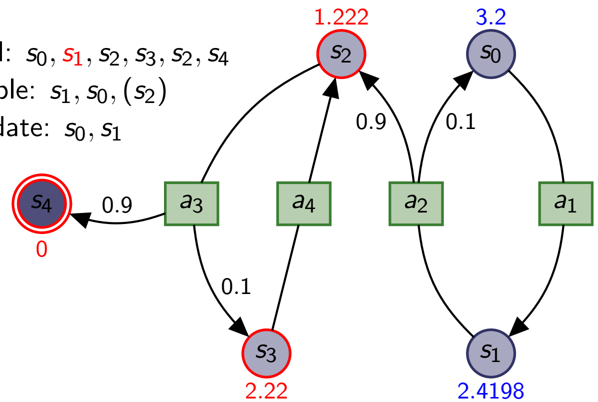
residual s_1: 0.1998

Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$

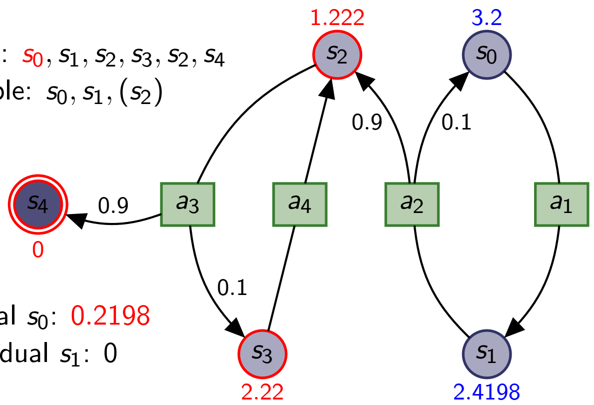
reachable: $s_1, s_0, (s_2)$

update: s_0, s_1



Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

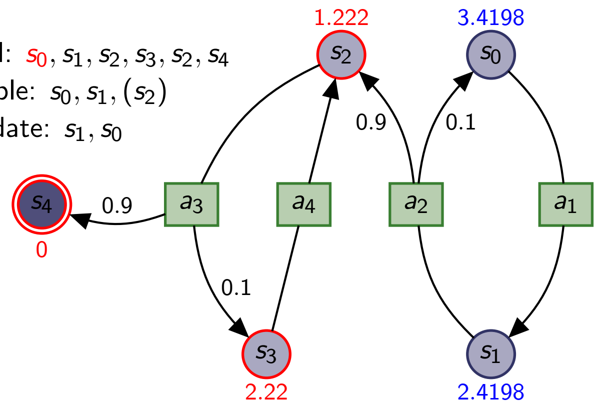
check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$
reachable: $s_0, s_1, (s_2)$



residual s_0: 0.2198
residual s_1: 0

Labeled RTDP: Cyclic Example ($\epsilon = 0.005$)

check_solved: $s_0, s_1, s_2, s_3, s_2, s_4$
reachable: $s_0, s_1, (s_2)$
update: s_1, s_0



Labeled Real-Time Dynamic Programming

Labeled RTDP for SSP $\mathcal{T}$

while s_0 is not solved:
    visit(s_0)

visit state s

if s is solved or $s \in S_\star$:
    return
$\hat{V}(s) := \min_{\ell \in L(s)} c(\ell) + \sum_{s' \in S} T(s, \ell, s') \cdot \hat{V}(s')$
$s' :\sim \text{succ}(s, a_{\hat{V}}(s))$
visit(s')
check_solved(s)

Note: If $\hat{V}(s')$ is used on the right hand side of line 3 or 4 in `visit(s)` but has has not been assigned before, $h(s)$ is used instead

Labeled RTDP: CheckSolved

check_solved for SSP $\mathcal{T}$

set ret := true, open, closed := stack
if s_0 not labeled **then** push $s0$ to open
while open is not empty:
    pop s from open and insert into closed
    if residual(s) $> \epsilon$
        ret := false
    else push all $s' \in \text{succ}(s, a_{\hat{V}}(s))$ to open
        that are not labeled and not in open or closed
if ret **then** label all s in closed
else perform backup on all s in closed

Labeled RTDP: Theoretical Properties

Theorem

Using an admissible heuristic, Labeled RTDP converges to an optimal solution without (necessarily) computing state-value estimates for all states.

Proof omitted.

Further RTDP Variants

Many variants exists, among them some interesting ones:

- Bounded RTDP (McMahan, Likhachecv & Gordon, 2005)
- Focused RTDP (Smith & Simmons, 2006)
- Bayesian RTDP (Sanner et al., 2009)

G3.5 Summary

Summary

- Asynchronous variants of value iteration are optimal as long as all states are selected repeatedly
- RTDP finds optimal solutions for SSPs
- and performs updates only on relevant states
- Labeled RTDP labels states as solved to stop updating converged states