Back to our section problem,
can we do the search faster than UCS?

Use A!*

https://qiao.github.io/PathFinding.js/visual/

Recap of A* Search from Lecture

A heuristic $h(s)$ is any estimate of FutureCost(s).

Run uniform cost search with **modified edge costs**:

$$\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)$$

A heuristic h is **consistent** if

- $\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

If h is consistent, A* returns the minimum cost path.

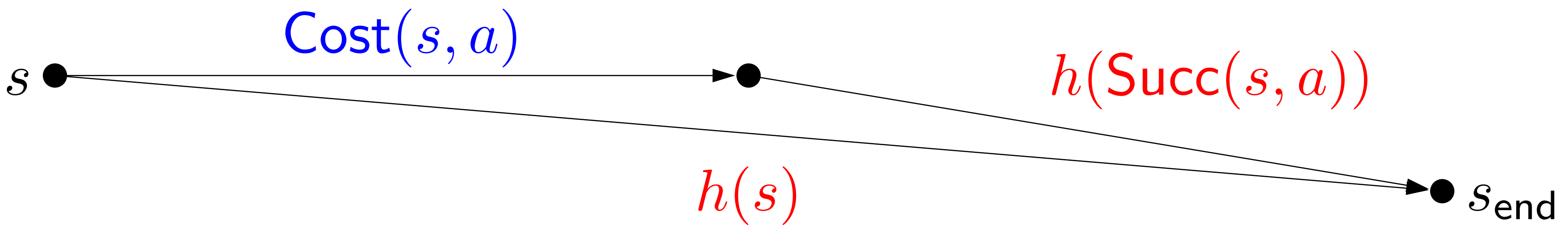
Consistent heuristics

Definition: consistency

A heuristic h is **consistent** if

- $\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

Condition 1: needed for UCS to work (triangle inequality).

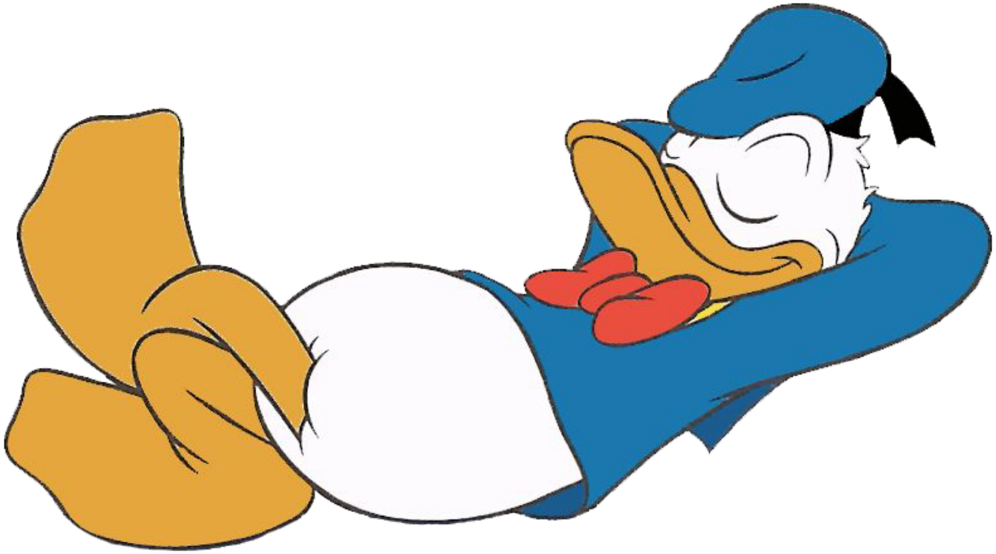


Condition 2: $\text{FutureCost}(s_{\text{end}}) = 0$ so match it.

Finding a Heuristic by **Relaxation**

→ try to solve an easier (less constrained) version of the problem

→ attain a problem that **can be solved more efficiently**



Relaxation, more formally:

Definition: relaxed search problem

A **relaxation** P' of a search problem P has costs that satisfy:

$$\text{Cost}'(s,a) \leq \text{Cost}(s,a).$$

Tradeoff

Efficiency:

$h(s) = \text{FutureCost}_{\text{rel}}(s)$ must be easy to compute

Closed form, easier search, independent subproblems

Tightness:

heuristic $h(s)$ should be close to $\text{FutureCost}(s)$

Don't remove too many constraints

Which heuristic would you use to solve our problem more efficiently?

Hint: Relaxation!

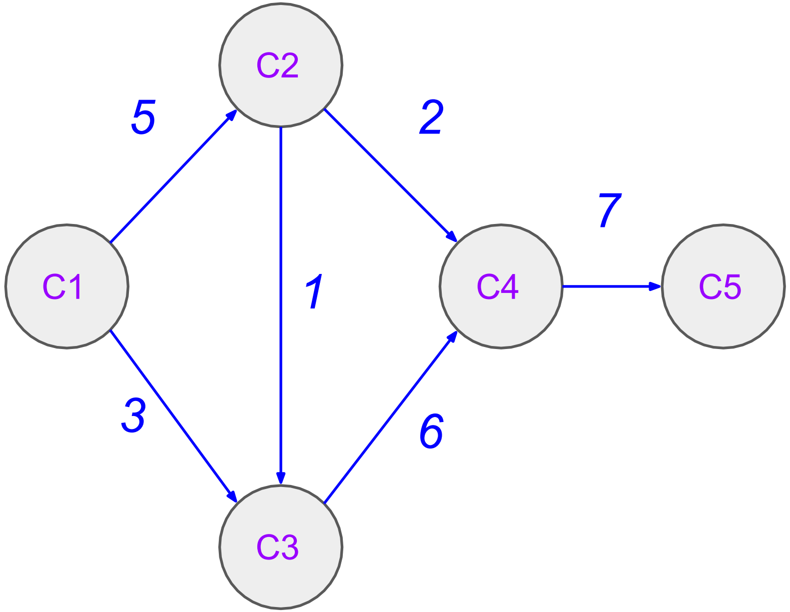
Section Problem

There exists ***N* cities**, labeled from *1* to *N*.

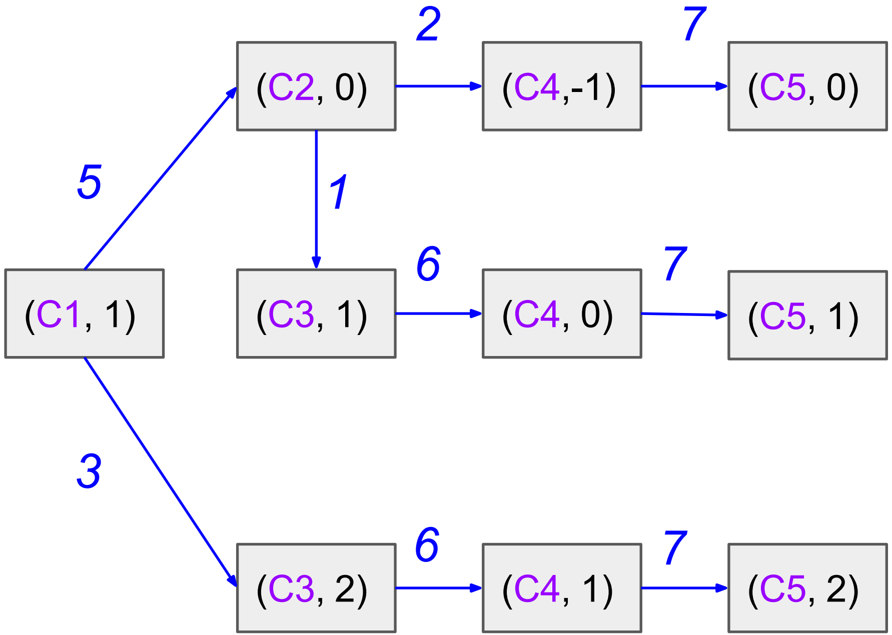
There are one-way roads connecting some pairs of cities. The road connecting city *i* and city *j* takes ***c(i,j)*** time to traverse. However, one can **only travel from a city with smaller label to a city with larger label** (each road is one-directional).

From city *1*, we want to travel to city *N*. What is the **shortest time** required to make this trip, given the **constraint** that we should visit **more odd-labeled cities than even labeled cities**?

Original Graph



State Graph



State $s = (i, d)$ (current city, #odd-#even)

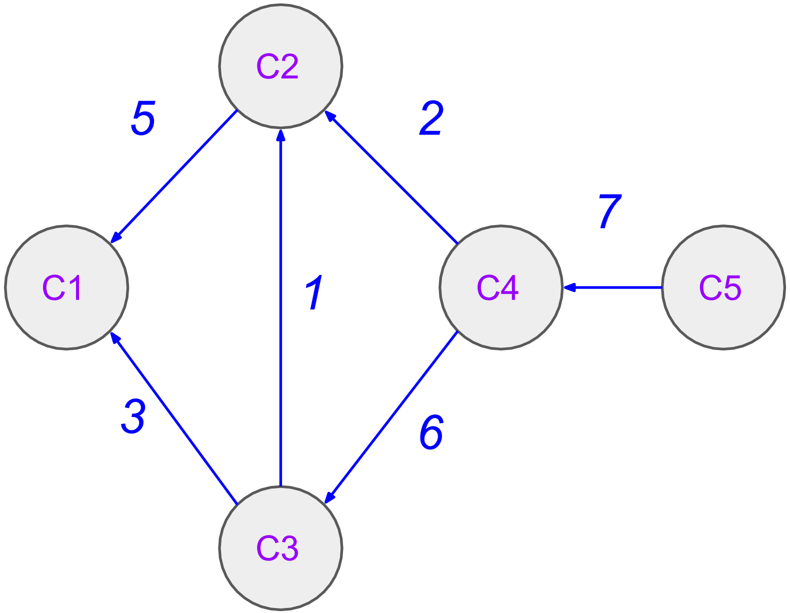
Heuristic for our problem

Remove the constraint that we visit more odd cities than even cities.

$h(s) = h((i, d))$ = length of shortest path from city i to city N

Note that the modified shortest path problem has $\boldsymbol{O(N)}$ states instead of $\boldsymbol{O(N^2)}$.

How to compute *h*?

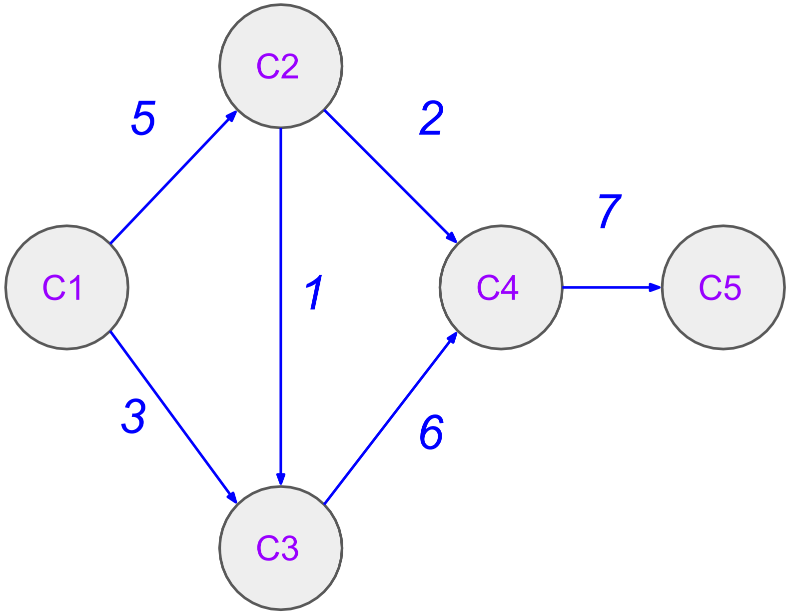


Reverse all edges, then perform UCS starting at C5 until C1 is found.

→ O(n log n) time (where n is # states whose distance to city CN is no farther than the distance of city C1 to city CN)

city	C1	C2	C3	C4	C5
h	14	9	13	7	0

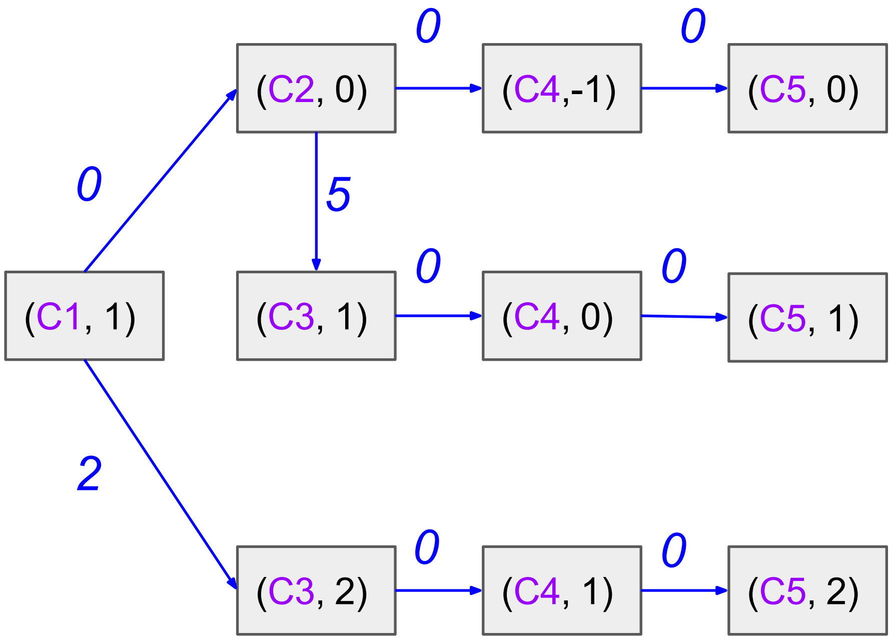
Original Graph



city	C1	C2	C3	C4	C5
h	14	9	13	7	0

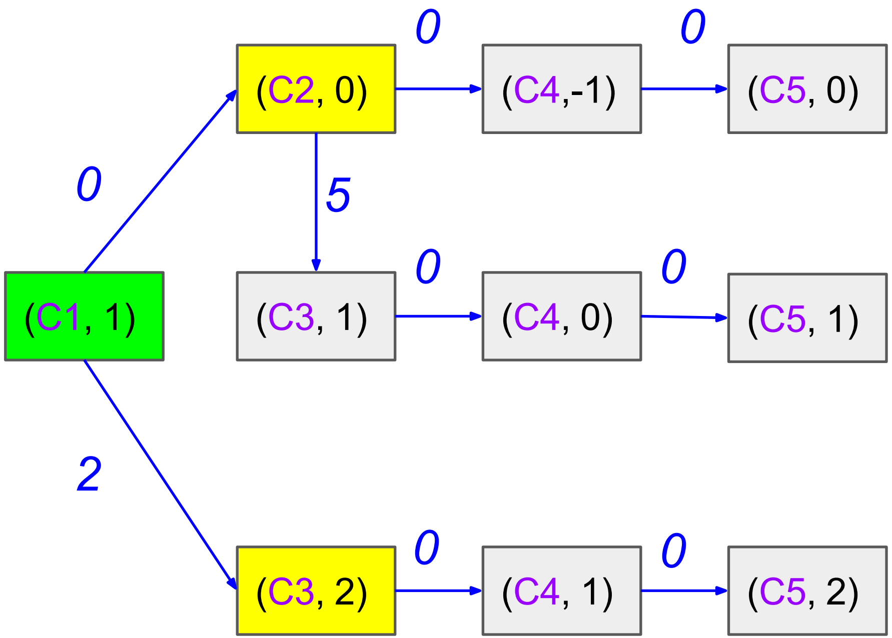
Modified State Graph

(updated edge costs)



State $s = (i, d)$ (current city, #odd-#even)

Simulation of UCS (A*)

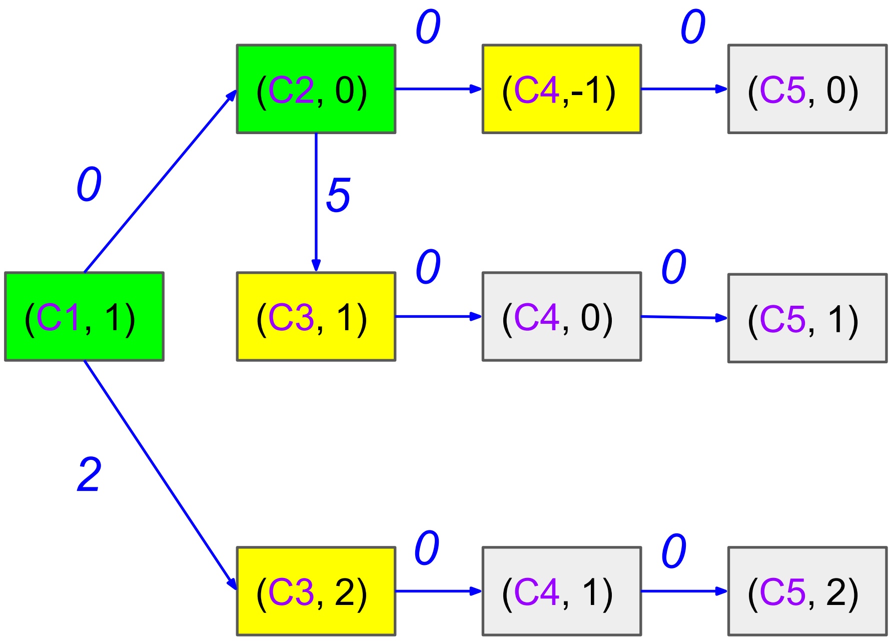


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0

Frontier:
(C2, 0) : 0
(C3, 2) : 2

Simulation of UCS (A*)

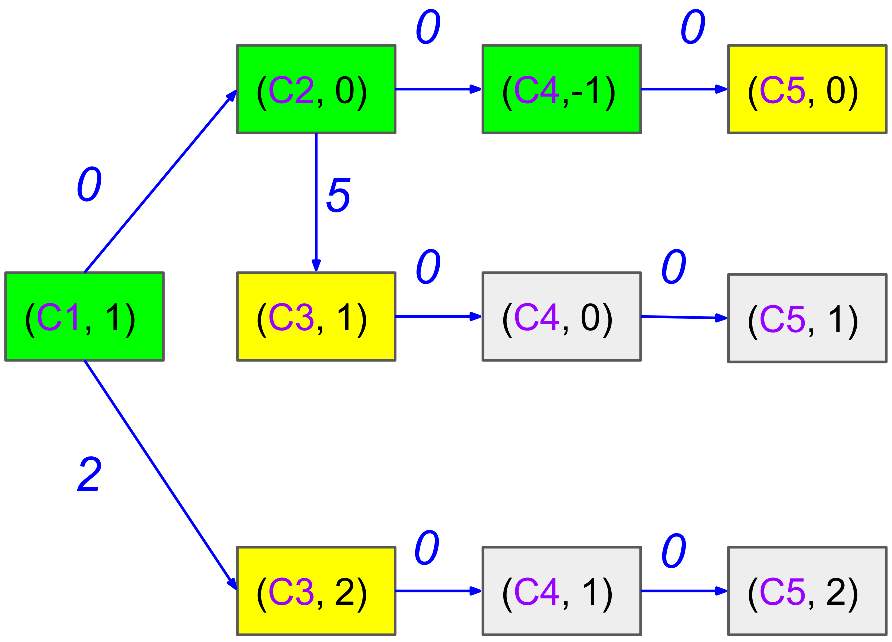


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0

Frontier:
(C4, -1) : 0
(C3, 2) : 2
(C3, 1) : 5

Simulation of UCS (A*)

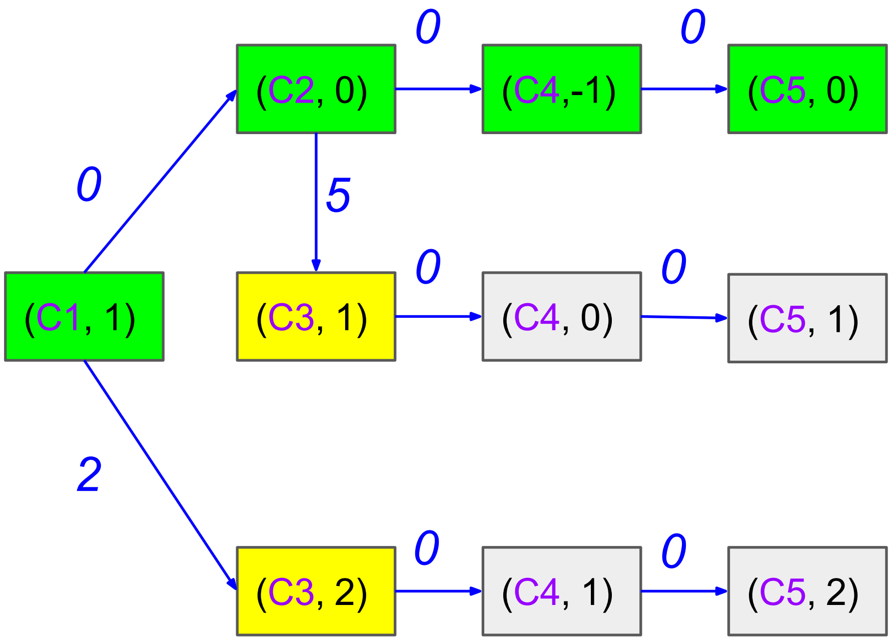


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0

Frontier:
(C5, 0) : 0
(C3, 2) : 2
(C3, 1) : 5

Simulation of UCS (A*)

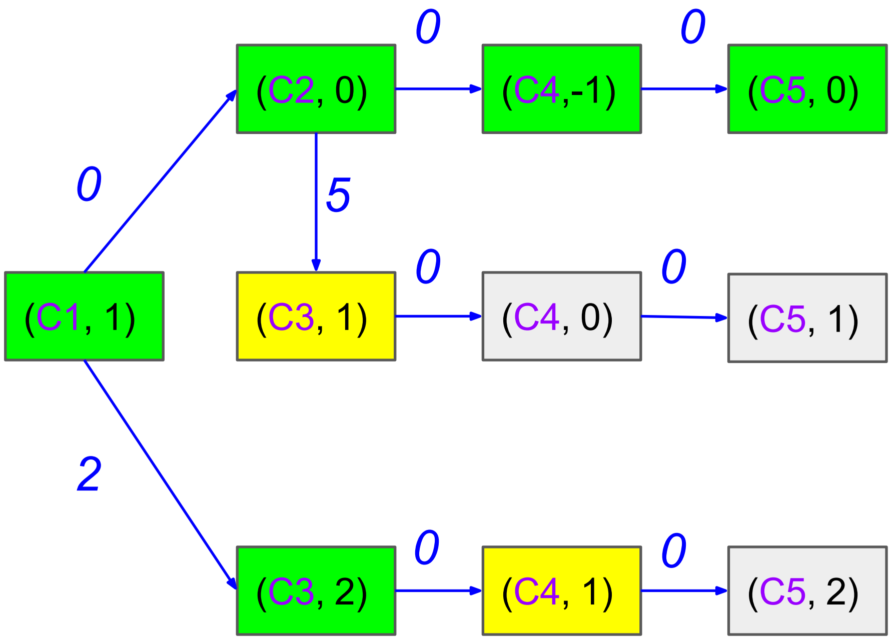


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0

Frontier:
(C3, 2) : 2
(C3, 1) : 5

Simulation of UCS (A*)

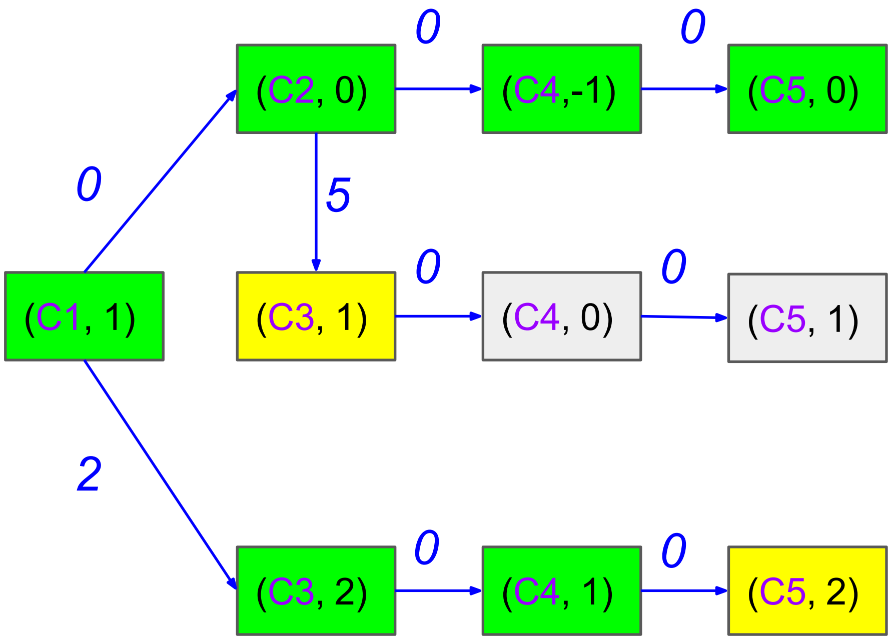


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2

Frontier:
(C4, 1) : 2
(C3, 1) : 5

Simulation of UCS (A*)

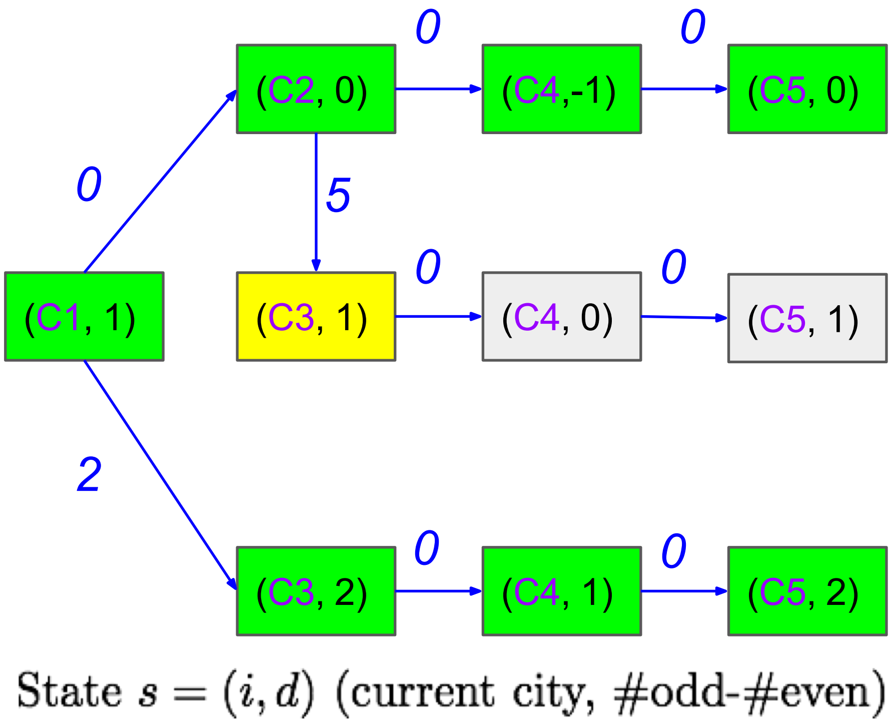


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1): 2

Frontier:
(C5, 2) : 2
(C3, 1) : 5

Simulation of UCS (A*)

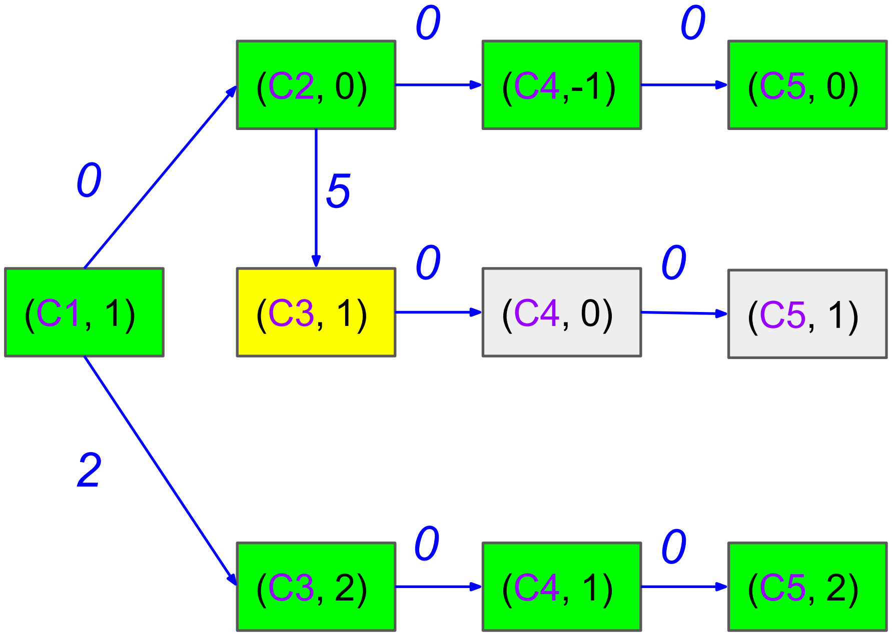


Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:
(C3, 1) : 5

STOP!

Simulation of UCS (A*)



State $s = (i, d)$ (current city, #odd-#even)

Explored:

(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:

(C3, 1) : 5

Actual Cost is 2 + $h(1)$ = 2 + 14 = 16

Comparison of States visited

UCS		UCS(A*)	
Explored:	Frontier:	Explored:	Frontier:
(C1, 1) : 0	(C5, 1) : 19	(C1, 1) : 0	(C3, 1) : 5
(C3, 2) : 3		(C2, 0) : 0	
(C2, 0) : 5		(C4, -1) : 0	
(C3, 1) : 6		(C5, 0) : 0	
(C4, -1) : 7		(C3, 2) : 2	
(C4, 1) : 9		(C4, 1) : 2	
(C4, 0) : 12		(C5, 2) : 2	
(C5, 0) : 14			
(C5, 2) : 16			

Comparison of States visited

UCS

Explored:
(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7
(C4, 1) : 9
(C4, 0) : 12
(C5, 0) : 14
(C5, 2) : 16

Frontier:
(C5, 1) : 19

UCS explored 9 states

UCS(A*)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:
(C3, 1) : 5

UCS(A*) explored 7 states

Summary

- ***States Representation/Modelling***
  - make state representation compact, remove unnecessary information
- ***DP***
  - underlying graph cannot have cycles
  - visit all reachable states, but no log overhead
- ***UCS***
  - actions cannot have negative cost
  - visit only a subset of states, log overhead
- ***A****
  - Introduce heuristic to guide search
  - ensure that relaxed problem can be solved more efficiently

Now let’s practice modeling our search problems!

MDPs: overview

Markov decision process

Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in$ States: starting state

Actions(s): possible actions from state s

$T(s'|s,a)$: probability of s' if take action a in state s

Reward(s,a,s'): reward for the transition (s,a,s')

IsEnd(s): whether at end

$0 \leq \gamma \leq 1$: discount factor (default: 1)

What is a solution?

Search problem: path (sequence of actions)

MDP:

Definition: policy

A **policy** π is a mapping from each state $s \in$ States to an action $a \in$ Actions(s).

Example: volcano crossing

s	$\pi(s)$
(1,1)	S
(2,1)	E
(3,1)	N
...	...

MDPs: policy evaluation

Discounting

Definition: utility

Path: $s_0, a_1 r_1 s_1, a_2 r_2 s_2, \dots$ (action, reward, new state).

The **utility** with discount γ is

$$u_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$$

Discount $\gamma = 1$ (save for the future):

[stay, stay, stay, stay]: $4 + 4 + 4 + 4 = 16$

Discount $\gamma = 0$ (live in the moment):

[stay, stay, stay, stay]: $4 + 0 \cdot (4 + \cdots) = 4$

Discount $\gamma = 0.5$ (balanced life):

[stay, stay, stay, stay]: $4 + \frac{1}{2} \cdot 4 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 4 = 7.5$

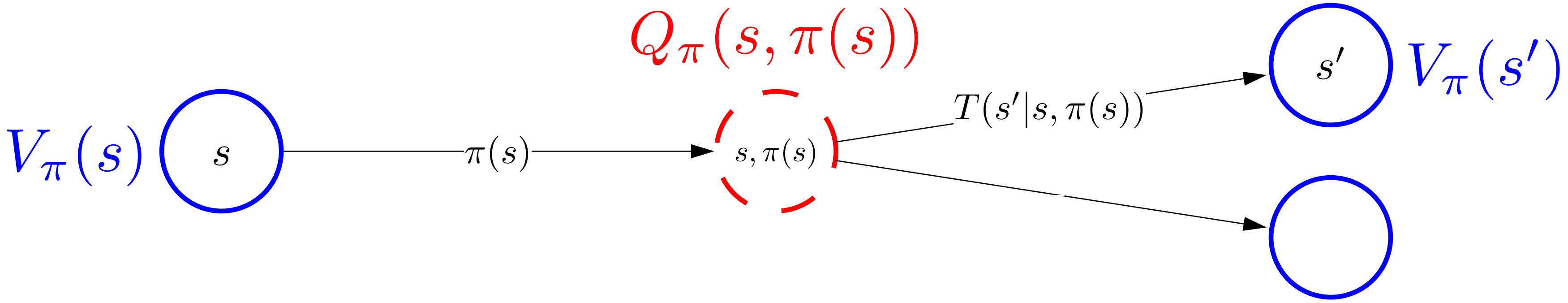
Policy evaluation

Definition: value of a policy

Let $V_\pi(s)$ be the expected utility received by following policy π from state s.

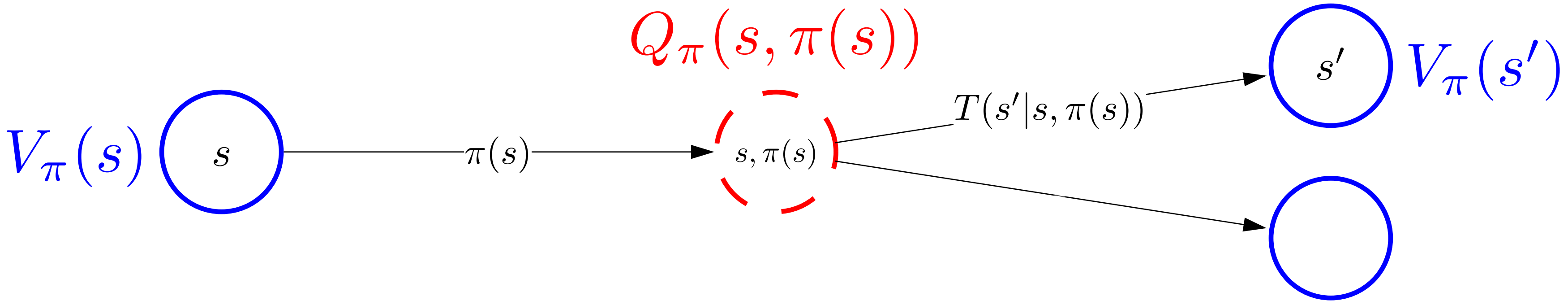
Definition: Q-value of a policy

Let $Q_\pi(s, a)$ be the expected utility of taking action a from state s, and then following policy π.



Policy evaluation

Plan: define recurrences relating value and Q-value



$$V_\pi(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

$$Q_\pi(s, a) = \sum_{s'} T(s'|s, a)[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$$

Policy evaluation

Key idea: iterative algorithm

Start with arbitrary policy values and repeatedly apply recurrences to converge to true values.

Algorithm: policy evaluation

Initialize $V_\pi^{(0)}(s) \leftarrow 0$ for all states s.

For iteration $t = 1, \dots, t_{\text{PE}}$:

For each state s:

$$V_\pi^{(t)}(s) \leftarrow \underbrace{\sum_{s'} T(s' | s, \pi(s))[\text{Reward}(s, \pi(s), s') + \gamma V_\pi^{(t-1)}(s')]}_{Q^{(t-1)}(s, \pi(s))}$$

MDPs: value iteration

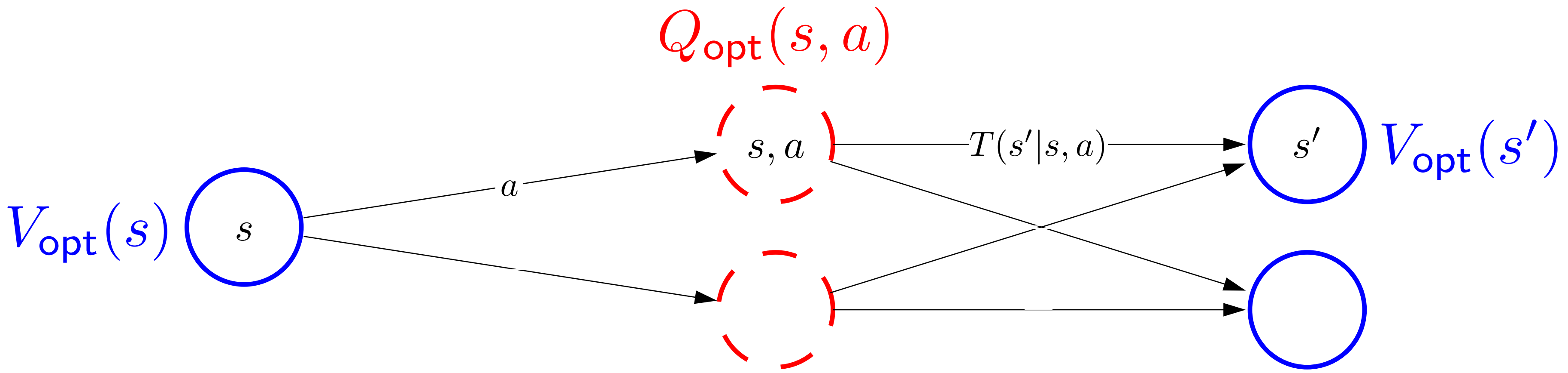
Optimal value and policy

Goal: try to get directly at maximum expected utility

Definition: optimal value

The **optimal value** $V_{\text{opt}}(s)$ is the maximum value attained by any policy.

Optimal values and Q-values



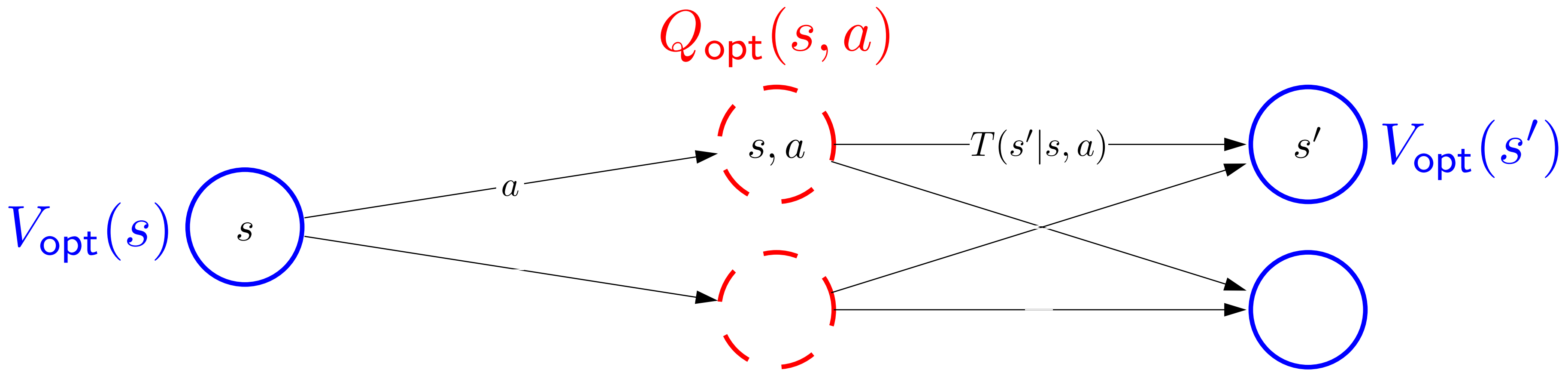
Optimal value if take action a in state s:

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')].$$

Optimal value from state s:

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a) & \text{otherwise.} \end{cases}$$

Optimal policies



Given Q_{opt}, read off the optimal policy:

$$\pi_{\text{opt}}(s) = \arg\max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s, a)$$

Value iteration

Algorithm: value iteration [Bellman, 1957]

Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.

For iteration $t = 1, \dots, t_{\text{VI}}$:

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{\text{opt}}^{(t-1)}(s,a)}$$

Time: $O(t_{\text{VI}} S A S')$

Convergence

Theorem: convergence

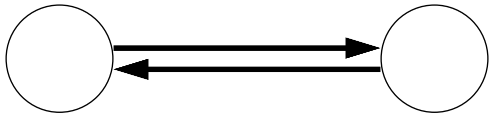
Suppose either

- discount $\gamma < 1$, or
- MDP graph is acyclic.

Then value iteration converges to the correct answer.

Example: non-convergence

discount $\gamma = 1$, zero rewards



Summary of algorithms

- **Policy evaluation**: (MDP, π) $\to V_\pi$

- **Value iteration**: MDP $\to (Q_{\text{opt}}, \pi_{\text{opt}})$

MDPs: reinforcement learning