Fall 2023

Dynamic programming

What makes sequential decision making hard?

Cathy Wu

6.7920: Reinforcement Learning: Foundations and Methods

Readings

1. DPOC 3.3-3.4

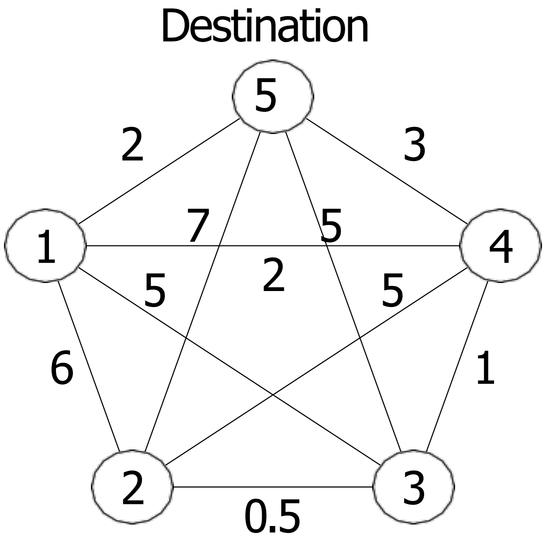
Outline

1. Solving finite-horizon decision problems

Outline

1. **Solving finite-horizon decision problems**
   a. **Example: shortest path routing**
   b. Dynamic programming algorithm
   c. Sequential decision making as shortest path
   d. Forward DP

Example: Shortest Path Problem

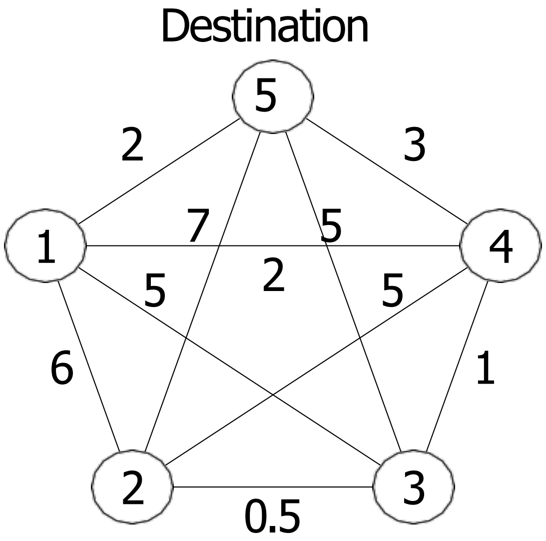


Destination is node 5.

Sequential decision problem

- Start state s_0: city 2
- Action a_0: take link between city 2 and city 3
- State s_1: city 3
- Action a_1: take link between city 3 and city 5
- State s_2: city 5
- …

Solving Shortest Path



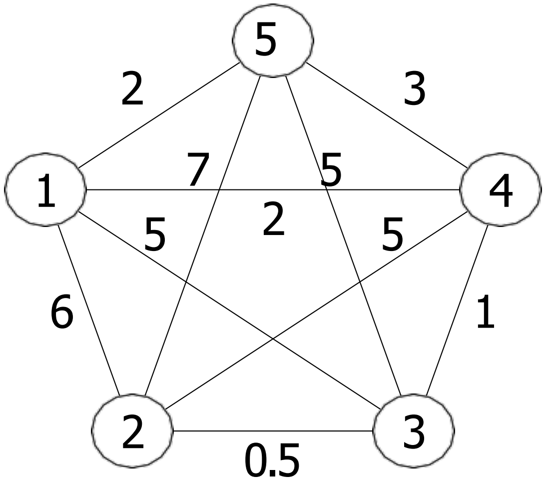
Destination is node 5.

Assumption: all cycles have non-negative length.

- **Naive approach:** enumerate all possibilities.
  - From a starting city s_0, choose any remaining city ($N - 1$ choices). Choose any next remaining city ($N - 2$ choices). ...
    Until there is only 1 option remaining.
  - Add up the edge costs.
  - Select the best sequence (lowest total cost).
  - $O(N!)$. ☹

Solving Shortest Path

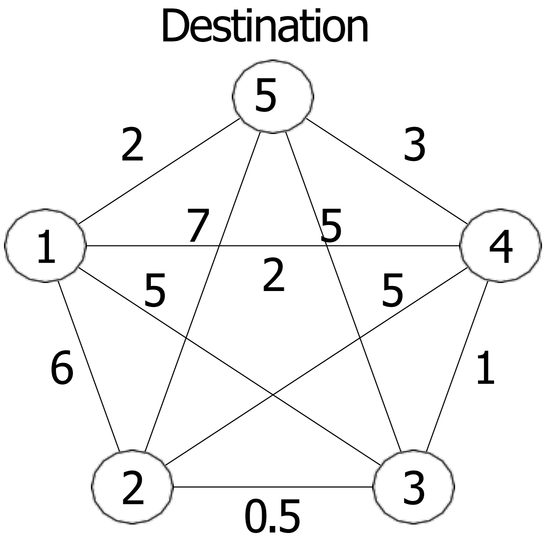
Destination

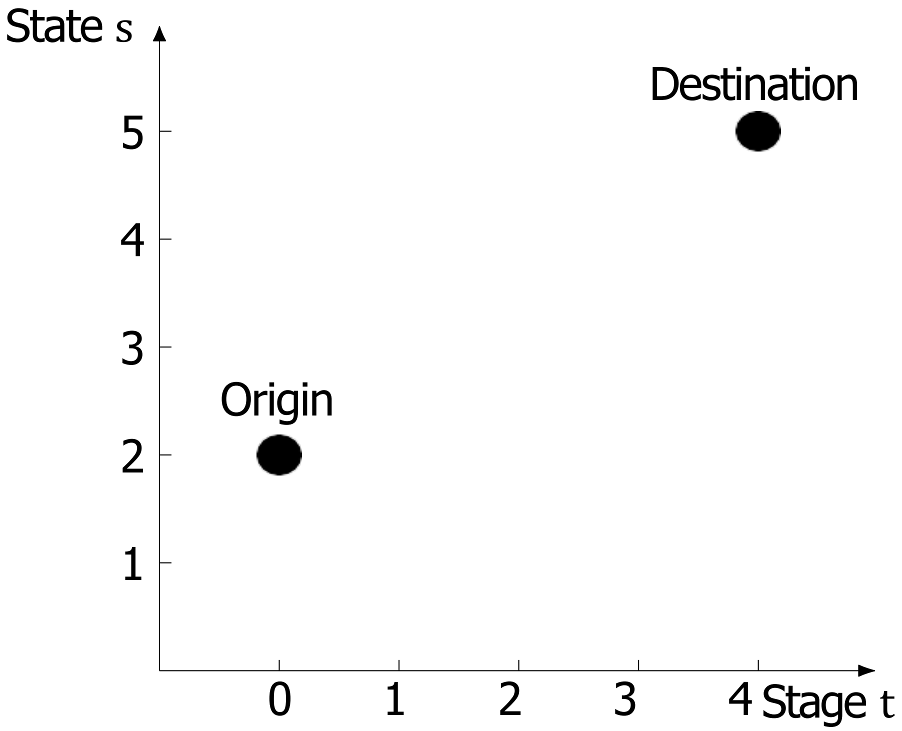


Destination is node 5.

- **Issue**: repeated calculations of subsequences.
  - Dynamic programming: divide-and-conquer, or **the principle of optimality**.
  - Overall problem would be much easier to solve if a part of the problem were already solved.
  - Break a problem down into subproblems.

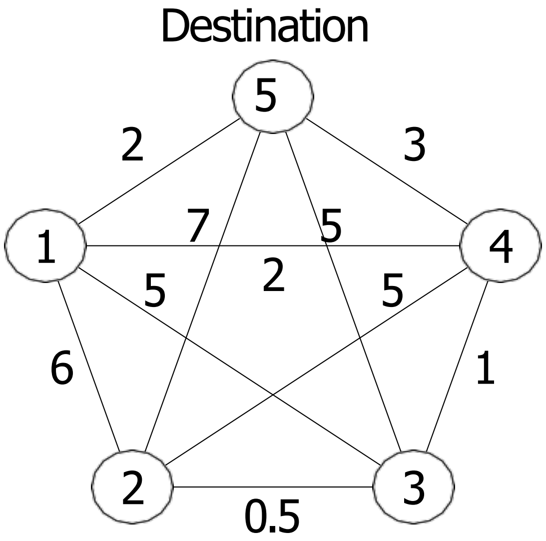
Solving Shortest Path



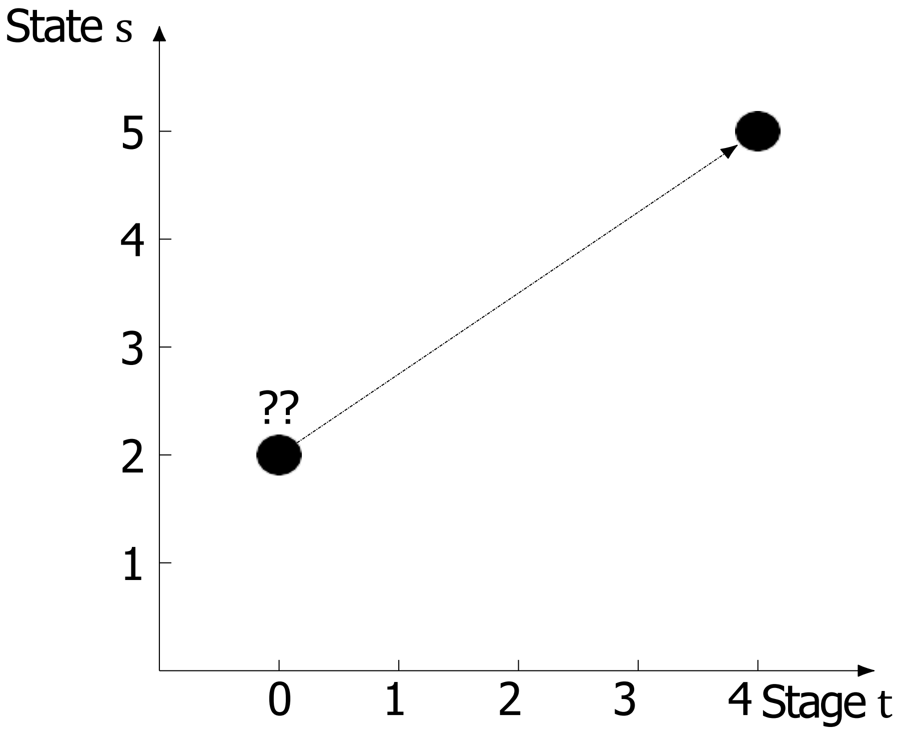


Destination is node 5.

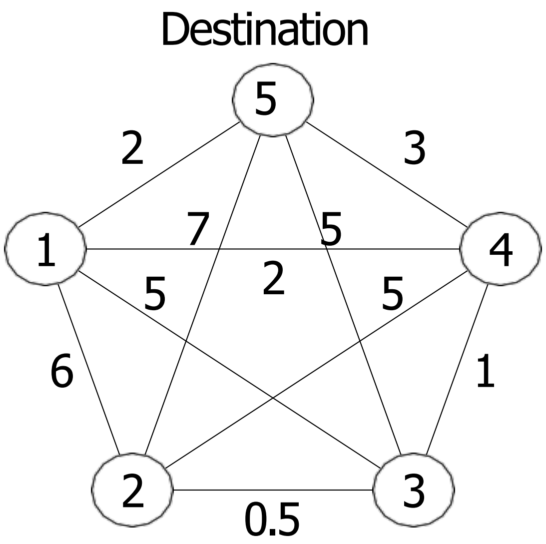
Solving Shortest Path



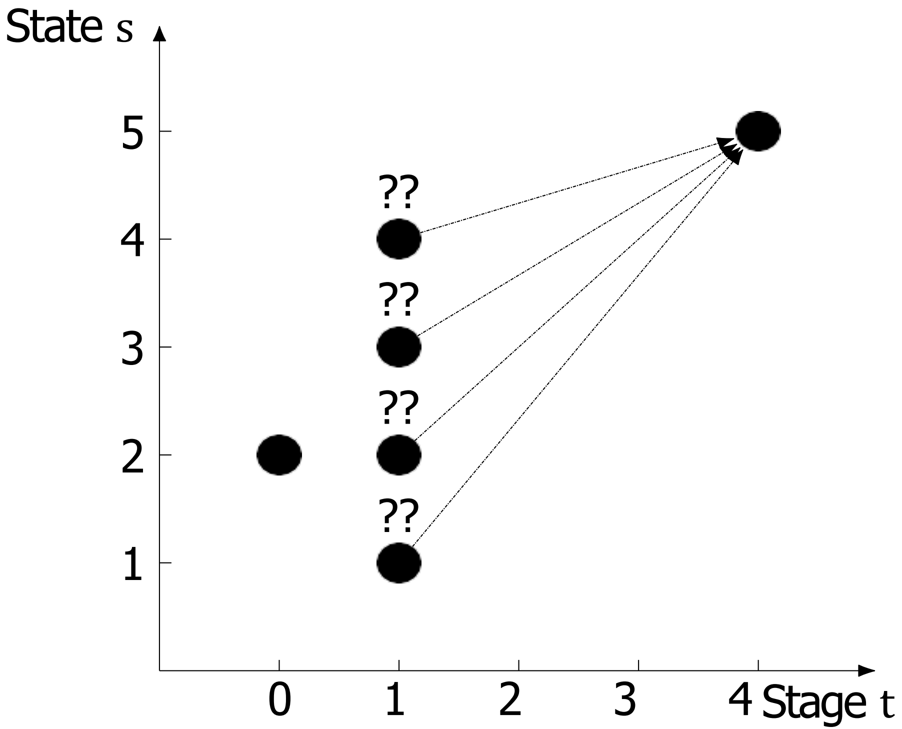
Destination is node 5.



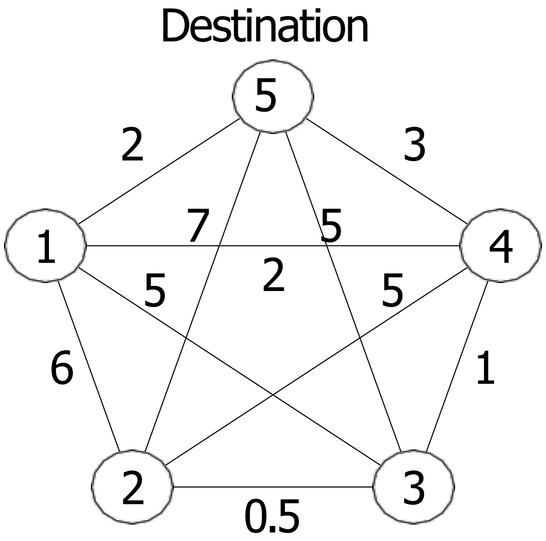
Solving Shortest Path



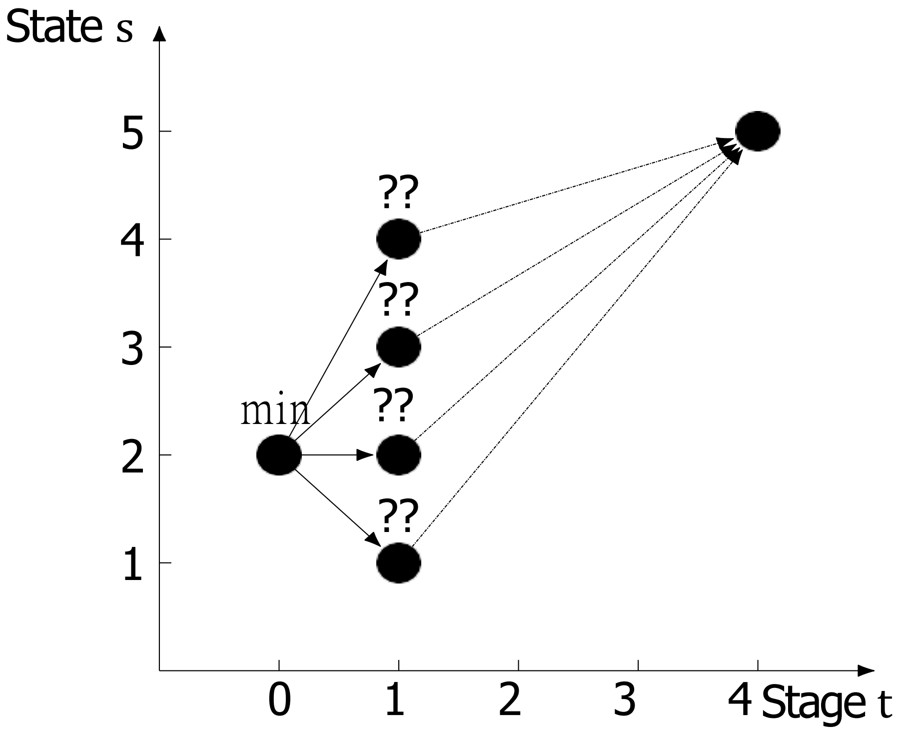
Destination is node 5.



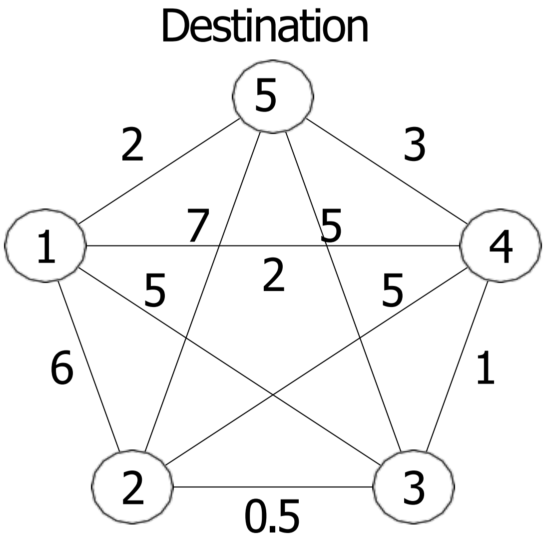
Solving Shortest Path



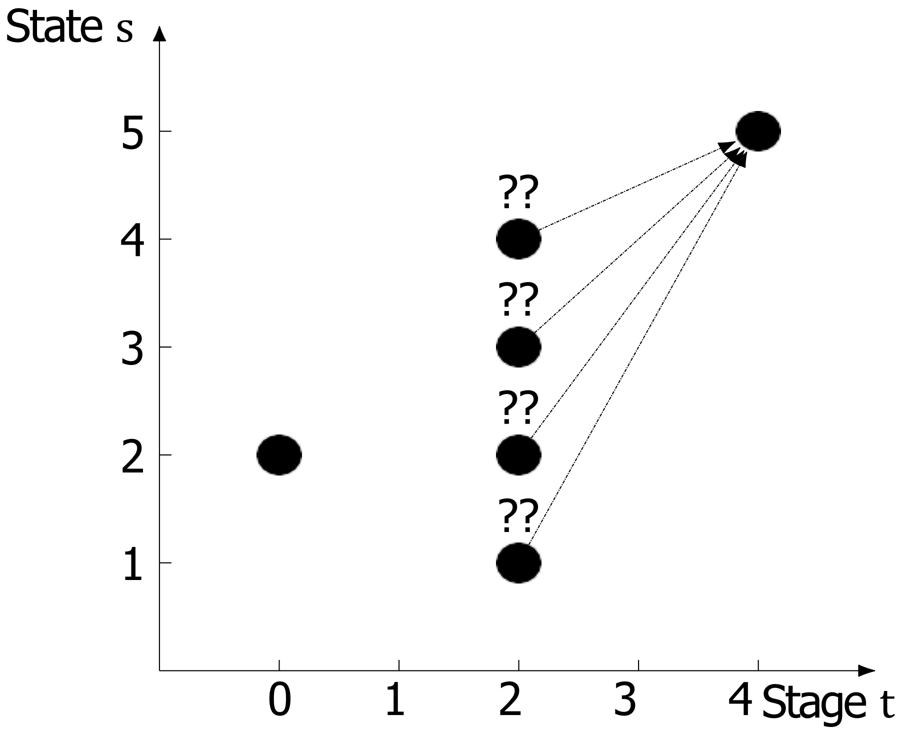
Destination is node 5.



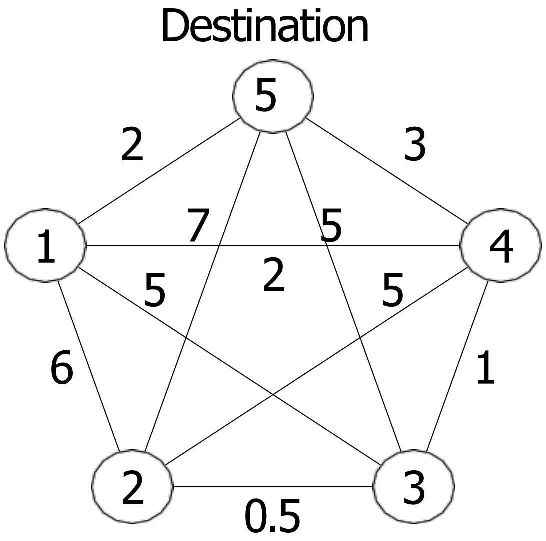
Solving Shortest Path



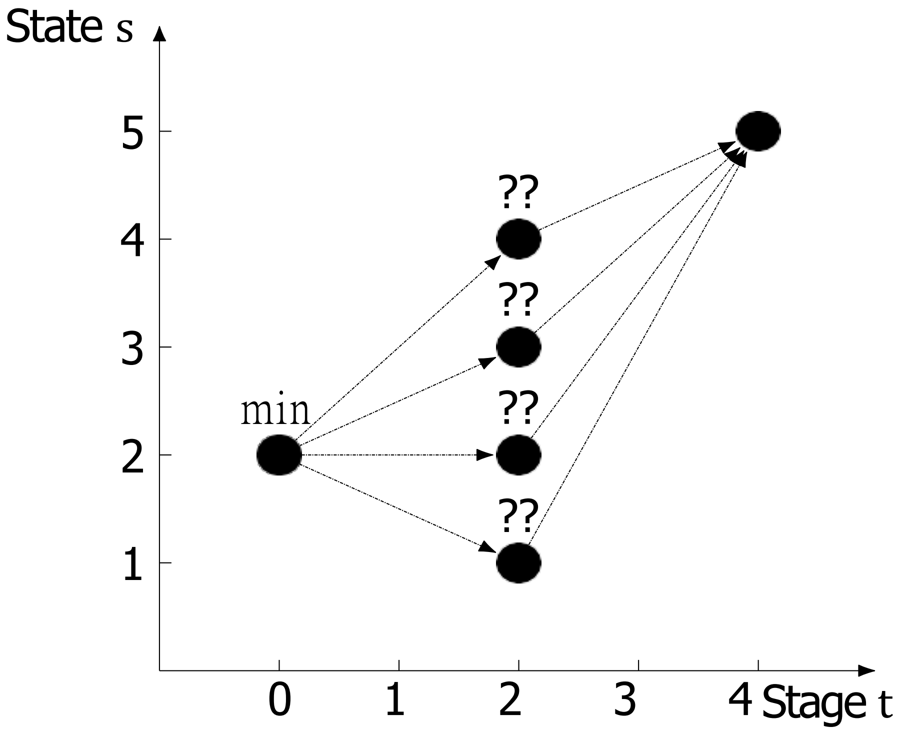
Destination is node 5.



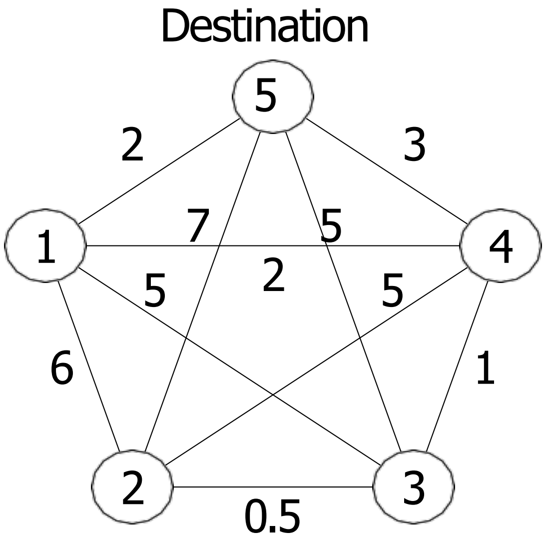
Solving Shortest Path



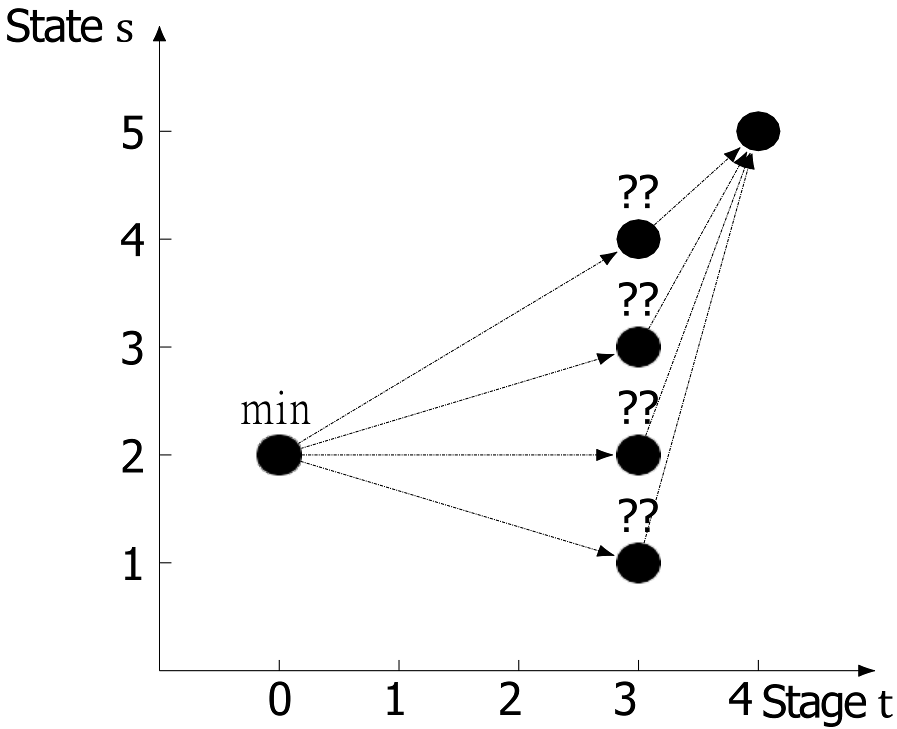
Destination is node 5.



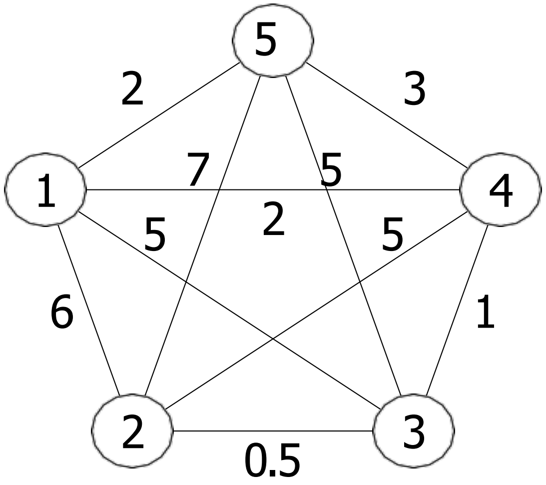
Solving Shortest Path



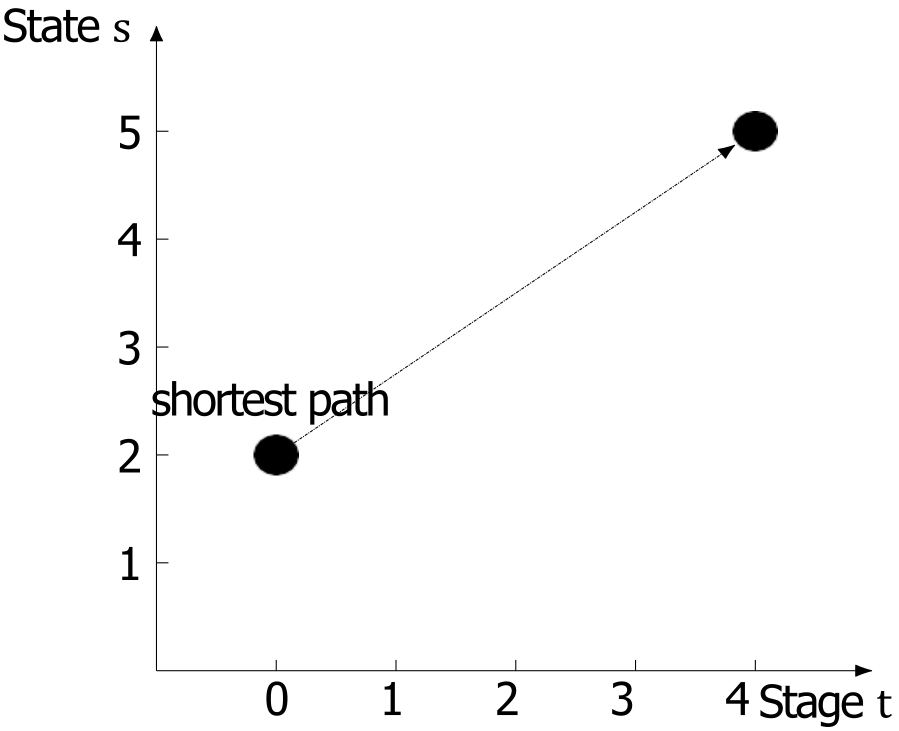
Destination is node 5.



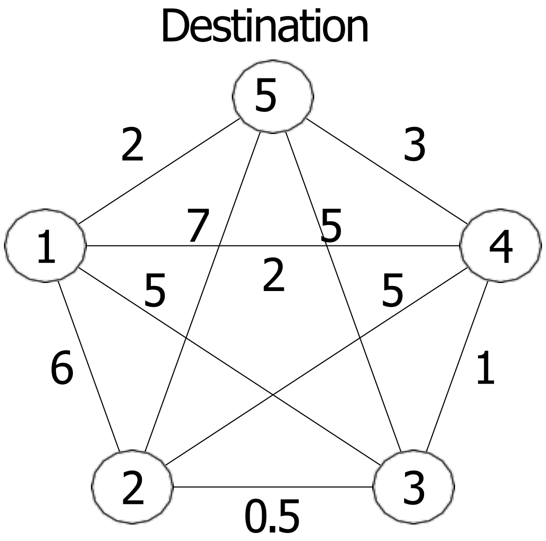
Solving Shortest Path



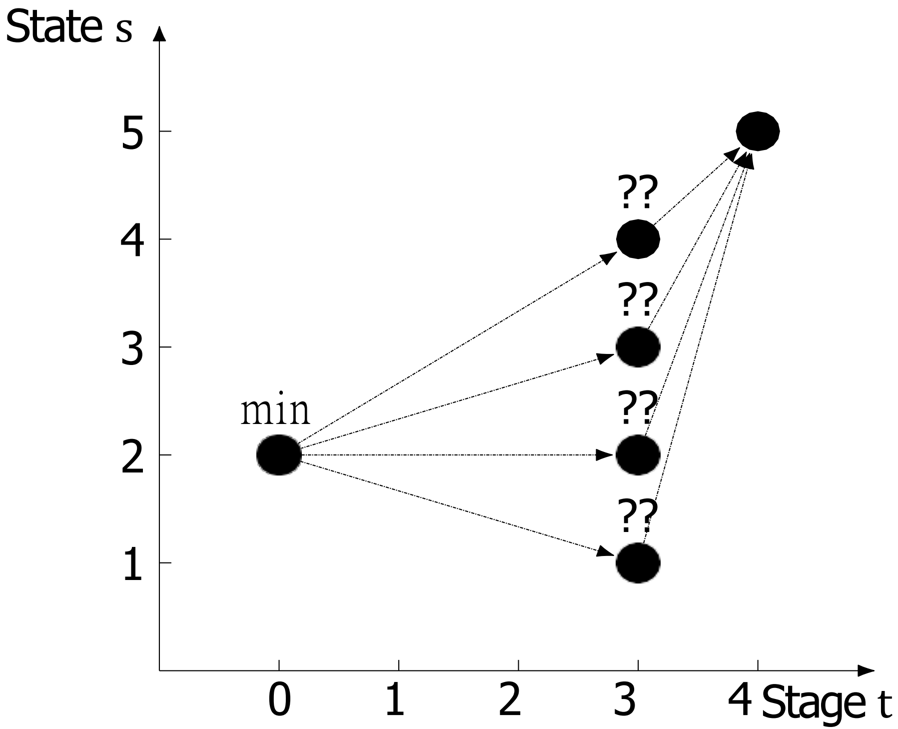
Destination is node 5.



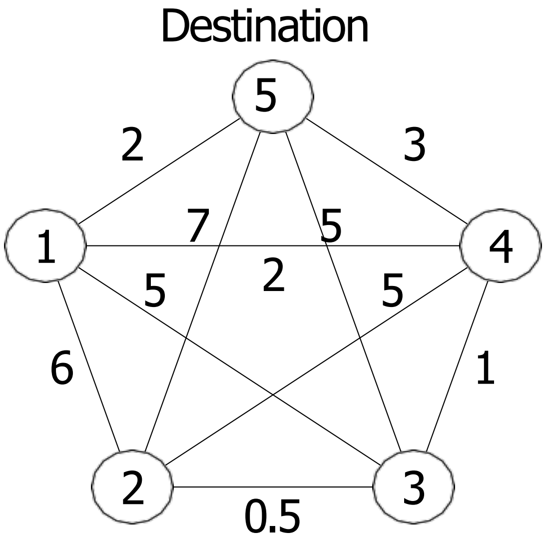
Solving Shortest Path



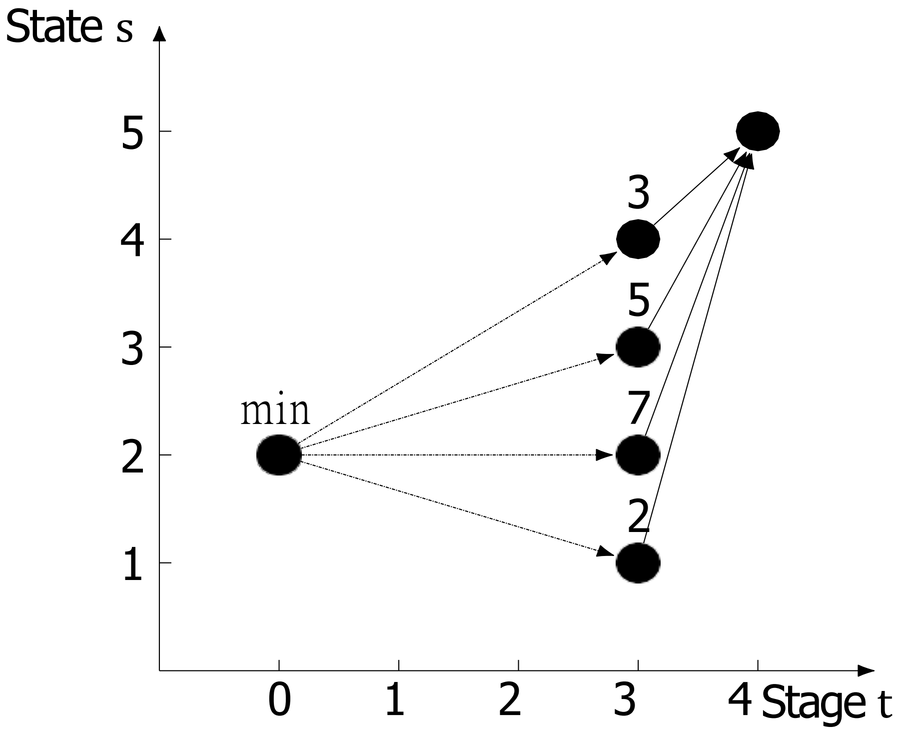
Destination is node 5.



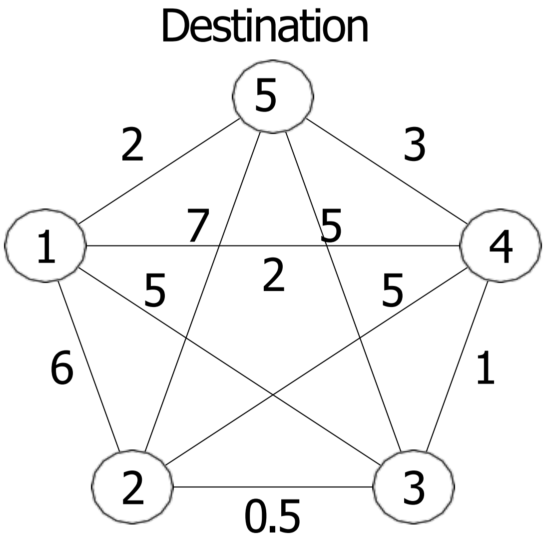
Solving Shortest Path



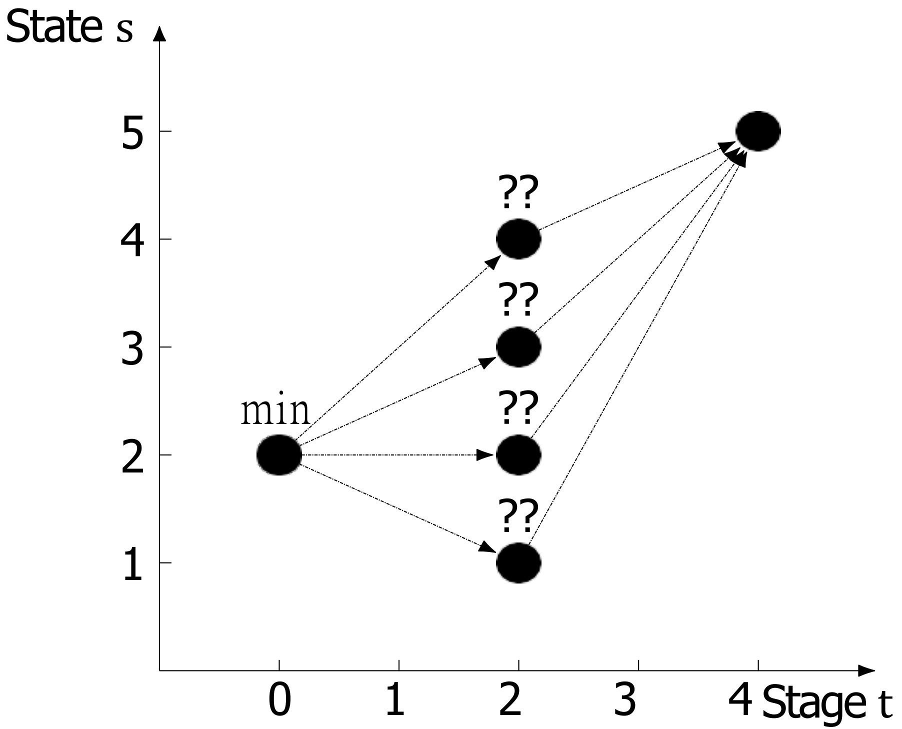
Destination is node 5.



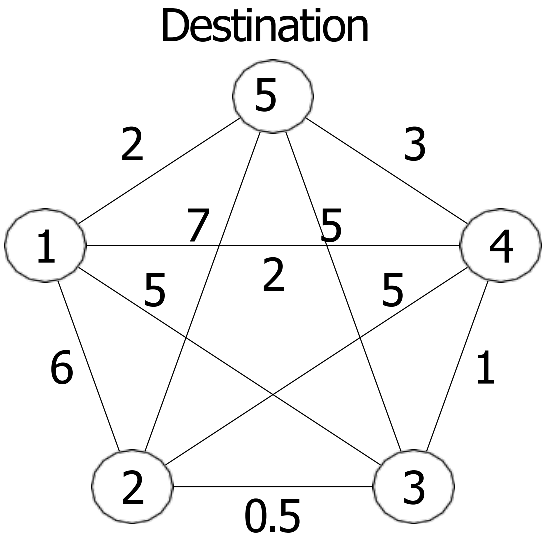
Solving Shortest Path



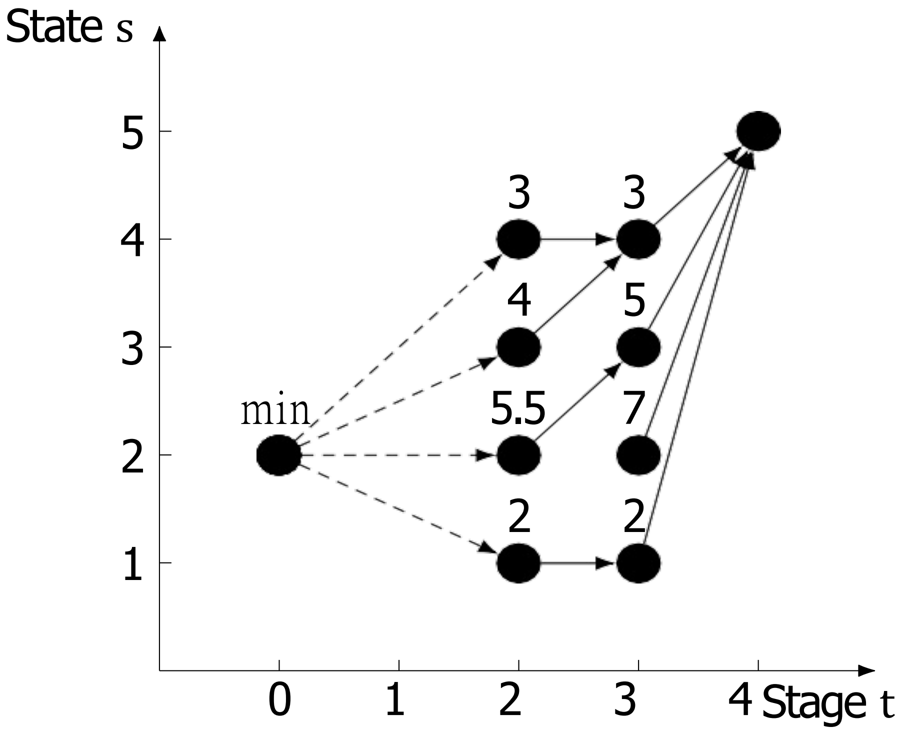
Destination is node 5.



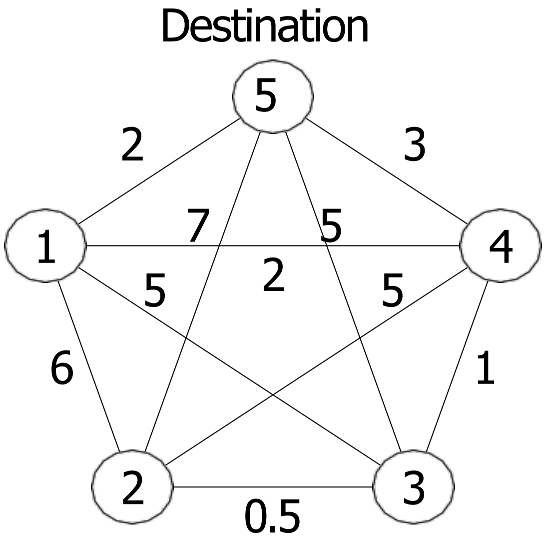
Solving Shortest Path



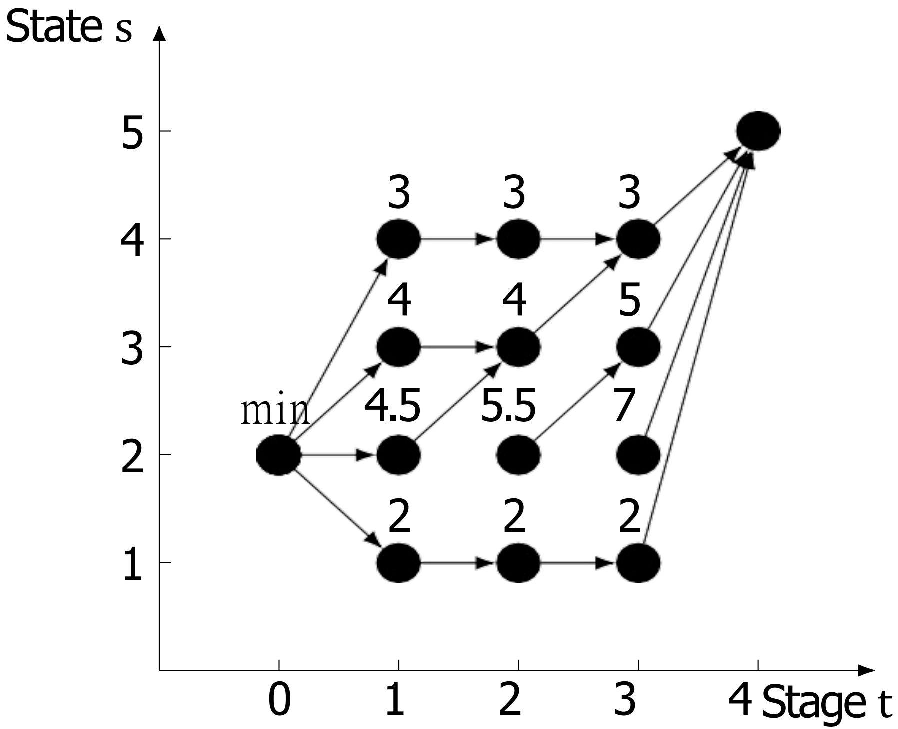
Destination is node 5.



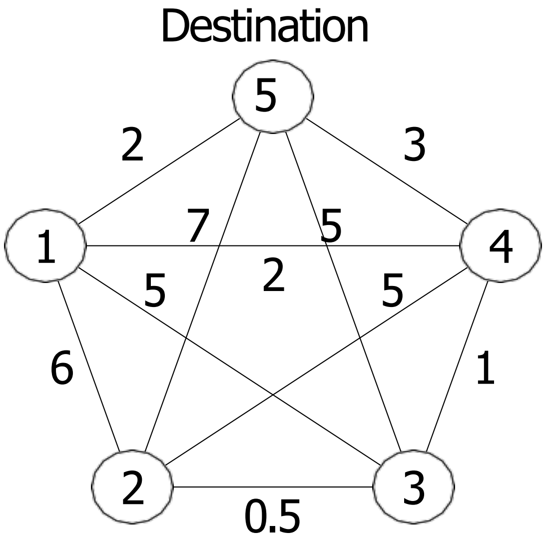
Solving Shortest Path



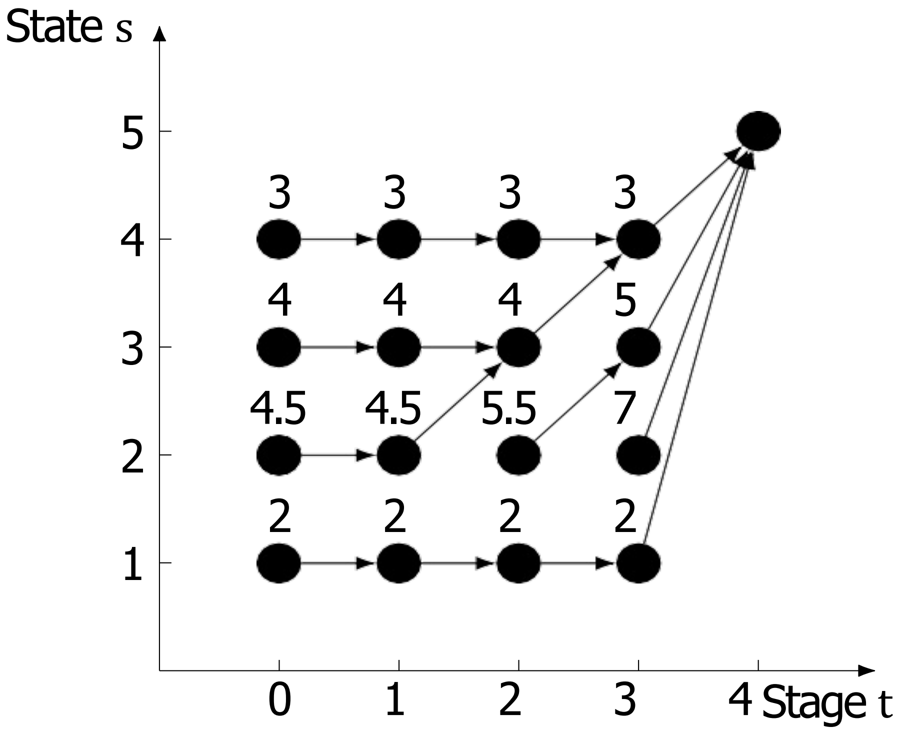
Destination is node 5.



Solving Shortest Path



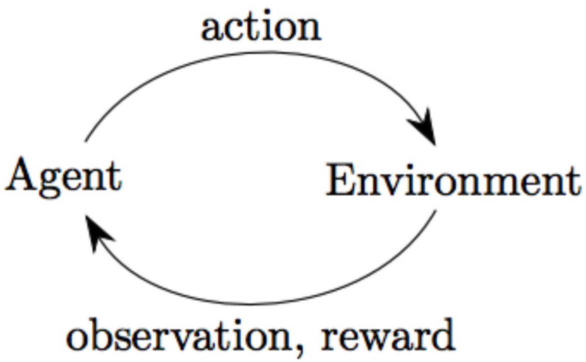
Destination is node 5.



Outline

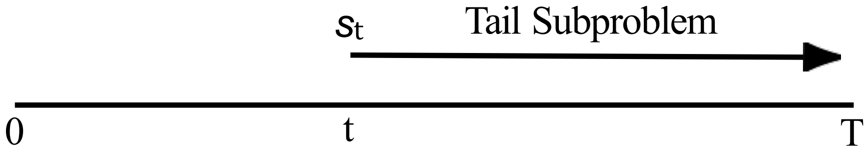
1. **Solving finite-horizon decision problems**
   a. Example: shortest path routing
   b. **Dynamic programming algorithm**
   c. Sequential decision making as shortest path
   d. Forward DP

More generally: stochastic problems



- Stochastic environment:
  - Uncertainty in rewards (e.g. multi-armed bandits, contextual bandits)
  - Uncertainty in dynamics, i.e.
    $$(s_t, a_t) \rightarrow s_{t+1}$$
  - Uncertainty in horizon (called stochastic shortest path)
- Stochastic policies (technical reasons)
  - Trades off exploration and exploitation
  - Enables off-policy learning
  - Compatible with maximum likelihood estimation (MLE)

Dynamic programming in **deterministic setting** is insufficient.



Bellman's Principle of optimality (1957)

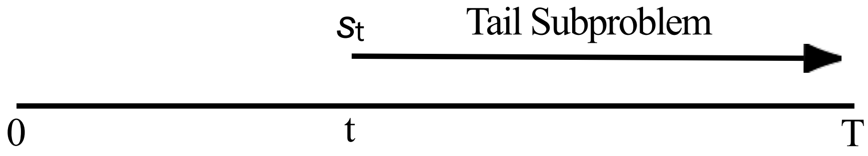
"An optimal policy has the property that, whatever the initial state and the initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

$$V^*(s) = \max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s')$$

Optimal Bellman equation, i.e. the workhorse of reinforcement learning

(Intuition for now, we will show it later)

Principle of optimality (Bellman, 1957)



Principle (Optimality)

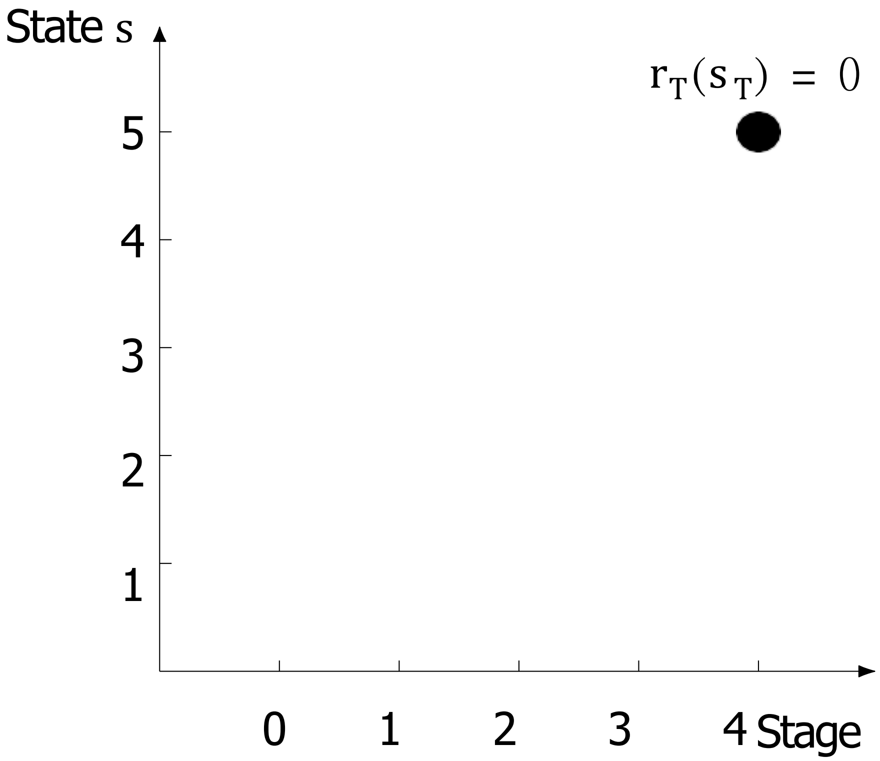
Let $\{a_0^*, \dots, a_{T-1}^*\}$ be an optimal action sequence, which together with s_0 and $\{\epsilon_0, \dots, \epsilon_{T-1}\}$ determines the corresponding state sequence $\{s_1^*, \dots, s_T^*\}$ via the state transition function. Consider the subproblem whereby we start at s_t^* at time t and wish to maximize the value function from time t to time T,

$$\mathbb{E}\left[r_t(s_t^*) + \sum_{\tau=t+1}^{T-1} r_\tau(s_\tau, a_\tau) + r_T(s_T)\right] \tag{1}$$

over $\{a_t, \dots, a_{T-1}\}$ with $a_\tau \in A_\tau(s_\tau), \tau = t, \dots, T-1$. Then, the truncated optimal action sequence $\{a_t^*, \dots, a_{T-1}^*\}$ is optimal for this subproblem.

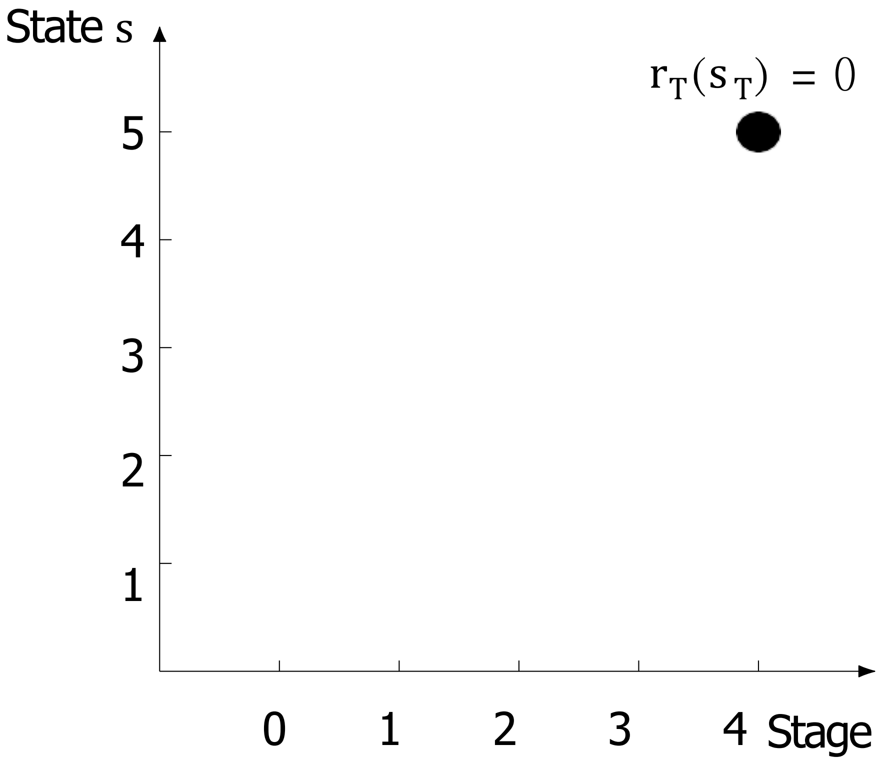
Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$



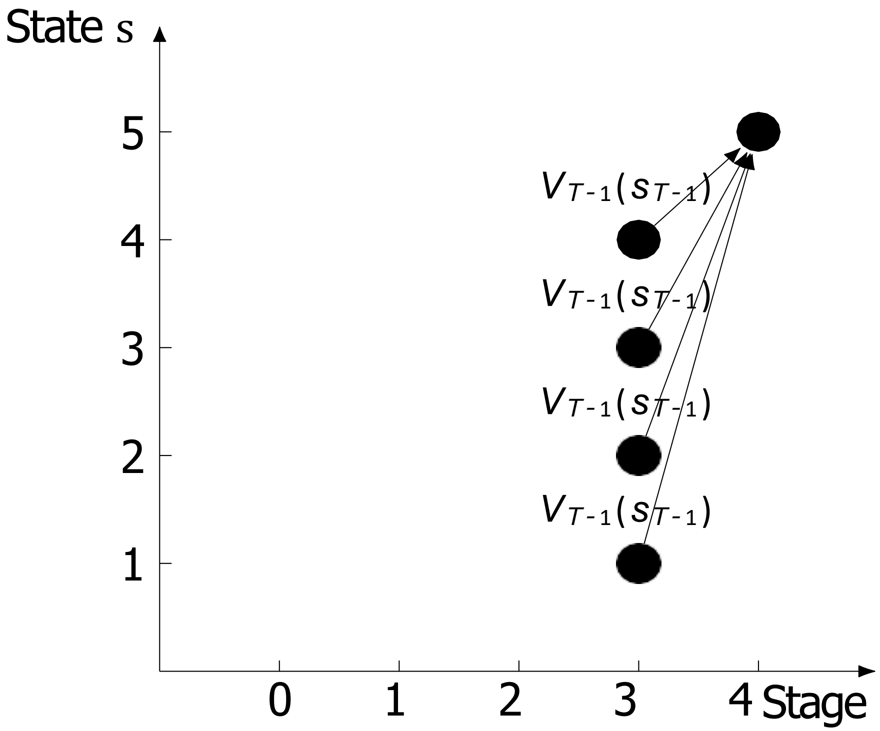
Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \dots, 0$ **do**



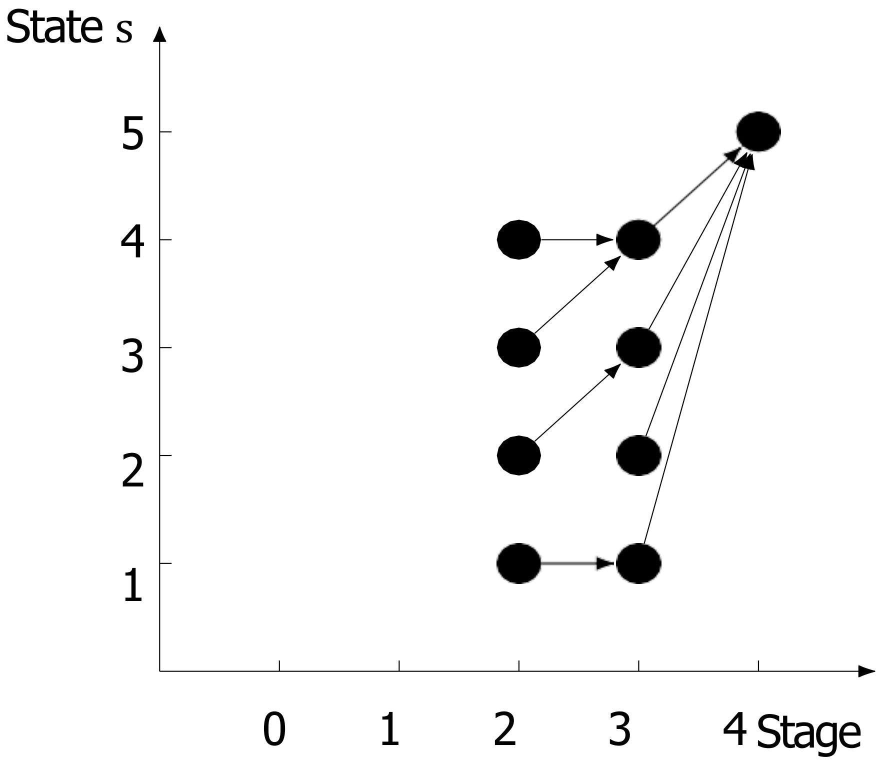
Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T - 1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**



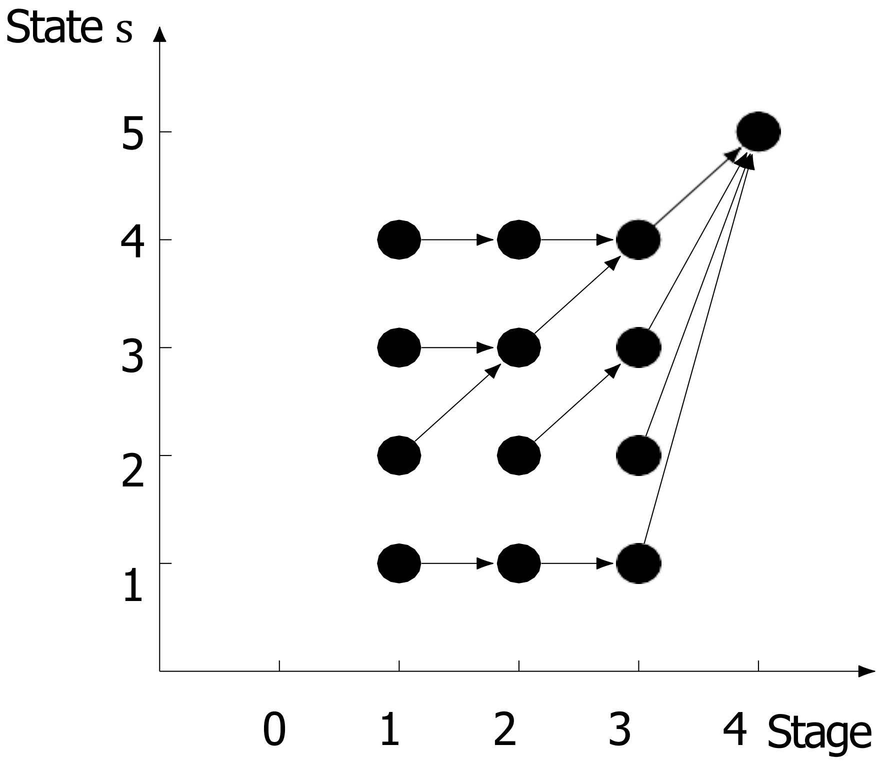
Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{s_{t+1} \sim f(s_t, a_t, \epsilon_t)}[r_t(s_t, a_t) + V_{t+1}(s_{t+1})]$
end for



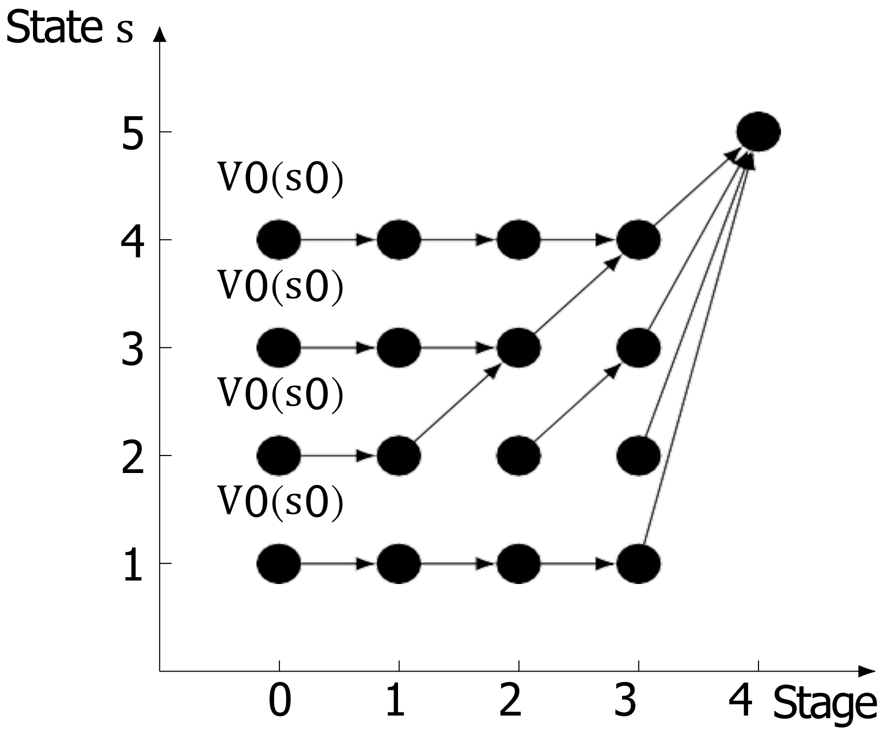
Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{s_{t+1} \sim f(s_t, a_t, \epsilon_t)}[r_t(s_t, a_t) + V_{t+1}(s_{t+1})]$
end for



Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{s_{t+1} \sim f(s_t, a_t, \epsilon_t)}[r_t(s_t, a_t) + V_{t+1}(s_{t+1})]$
end for



Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \ldots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{s_{t+1} \sim f(s_t, a_t, \epsilon_t)}[r_t(s_t, a_t) + V_{t+1}(s_{t+1})]$
end for

Theorem (Dynamic programming)

For every initial state s_0, the optimal value $V^*(s_0)$ is equal to $V_0(s_0)$, given above.
Furthermore, if $a_t^* = \pi_t^*(s_t)$ maximizes the right side of the above for each s_t and t, the policy $\pi^* = (\pi_0^*, \ldots, \pi_{T-1}^*)$ is optimal.

Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \ldots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{s_{t+1} \sim f(s_t, a_t, \epsilon_t)}[r_t(s_t, a_t) + V_{t+1}(s_{t+1})]$
end for

- **Proof**: by induction
- "Efficient": O(|S|^2|A|T)

- For deterministic shortest path routing
  - Equivalent to Bellman-Ford algorithm
  - **Strength**: Generality
  - "Efficient": O(|S| |A|T)
  - Much better than naive approach O(T!)
  - **Weakness**: ALL the tail subproblems are solved
- **Consider**: Do other shortest path algorithms have sequential decision interpretations? Dijkstra's, A*, Floyd–Warshall, Johnson's, Viterbi, etc.

Proof of the induction step

Let $f_t: S \times A \times \mathbb{R} \to S$ denote the transition *function*.

For simplicity, consider deterministic policies $\pi_t: S \to A$.

Denote tail policy from time t onward as $\pi_{t:T-1} = \{\pi_t, \pi_{t+1}, \dots, \pi_{T-1}\}$

Assume that $V_{t+1}(s_{t+1}) = V^*_{t+1}(s_{t+1})$. Then:

$$V_t^*(s_t) = \max_{(\pi_t, \pi_{t+1:T-1})} \mathbb{E}_{\epsilon_{t:T-1}} \left\{ r_t\big(s_t, \pi_t(s_t)\big) + r_T(s_T) + \sum_{i=t+1}^{T-1} r_i\big(s_i, \pi_i(s_i)\big) \right\}$$

$$= \max_{\pi_t} r_t\big(s_t, \pi_t(s_t)\big) + \max_{\pi_{t+1:T-1}} \left[\mathbb{E}_{\epsilon_{t:T-1}} \left\{ r_T(s_T) + \sum_{i=t+1}^{T-1} r_i\big(s_i, \pi_i(s_i)\big) \right\} \right]$$

$$= \max_{\pi_t} r_t\big(s_t, \pi_t(s_t)\big) + \mathbb{E}_{\epsilon_t} \left\{ \max_{\pi_{t+1:T-1}} \left[\mathbb{E}_{\epsilon_{t+1:T-1}} \left\{ r_T(s_T) + \sum_{i=t+1}^{T-1} r_i\big(s_i, \pi_i(s_i)\big) \right\} \right] \right\}$$

$$= \max_{\pi_t} r_t\big(s_t, \pi_t(s_t)\big) + \mathbb{E}_{\epsilon_t} \left\{ V^*_{t+1}\Big(f_t\big(s_t, \pi_t(s_t), \epsilon_t\big)\Big) \right\}$$

$$= \max_{\pi_t} r_t\big(s_t, \pi_t(s_t)\big) + \mathbb{E}_{\epsilon_t} \left\{ V_{t+1}\Big(f_t\big(s_t, \pi_t(s_t), \epsilon_t\big)\Big) \right\}$$

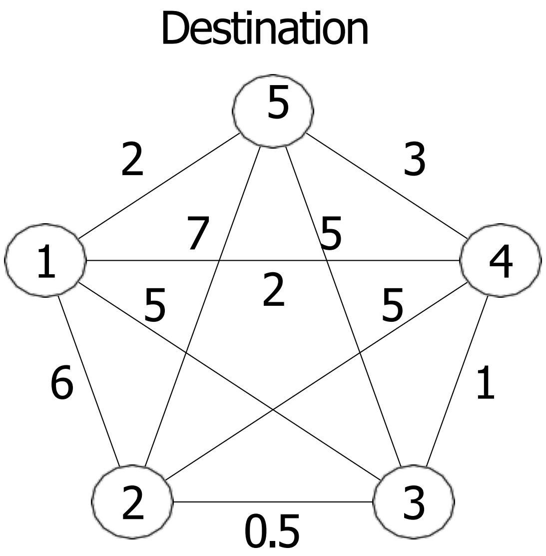
$$= \max_{a_t \in \mathcal{A}_t(s_t)} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim f(s_t, a_t, \epsilon_t)} \left\{ V_{t+1}\big(f_t(s_t, a_t, \epsilon_t)\big) \right\}$$

$$= V_t(s_t)$$

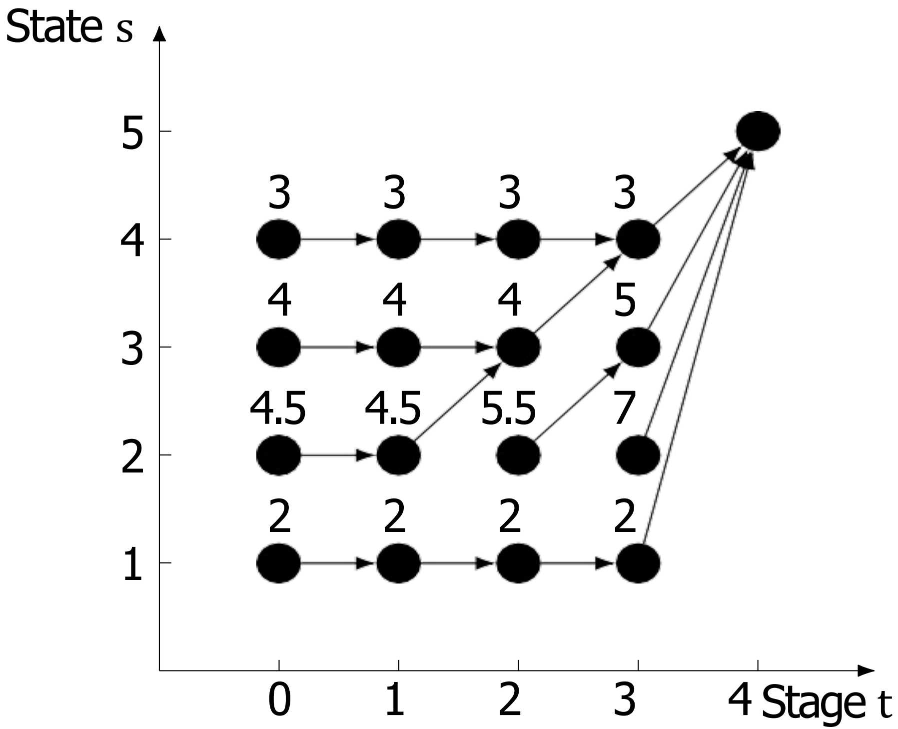
Interpretation as optimal reward-to-go (cost-to-go) function.

ϵ_t denotes the randomness in transitions from s_t to s_{t+1}

Solving Shortest Path



Destination is node 5.

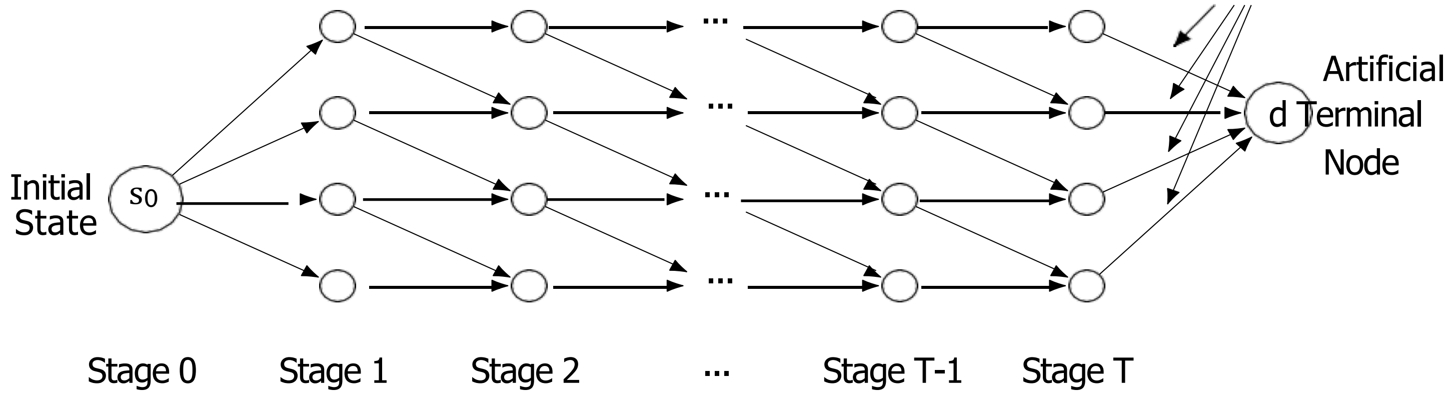


Outline

1. **Solving finite-horizon decision problems**
   a. Example: shortest path routing
   b. Dynamic programming algorithm
   c. **Sequential decision making as shortest path**
   d. Forward DP

Sequential decision making as shortest path

For Deterministic Finite-State Problems



Example: Thermostats (linear-quadratic control)

64F (18C)

64F

65F

... ...

74F

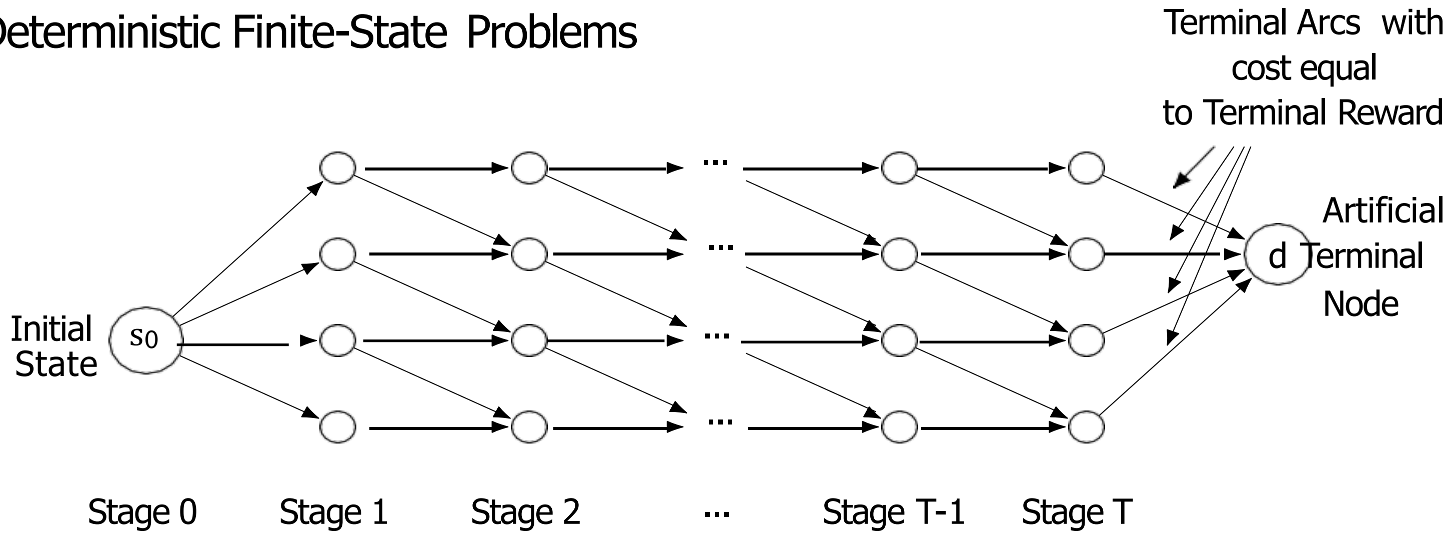
Applications:
control systems,
industrial manufacturing

Too cold! 🥶 ...

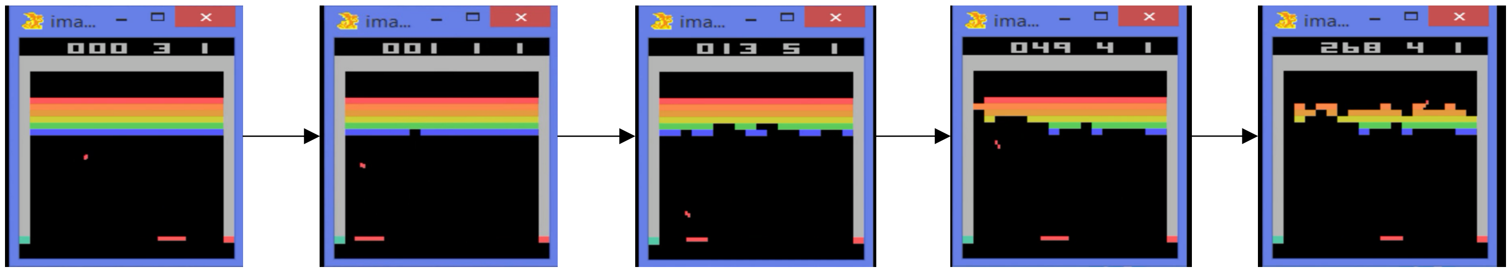
Great temperature ☺️

Sequential decision making as shortest path

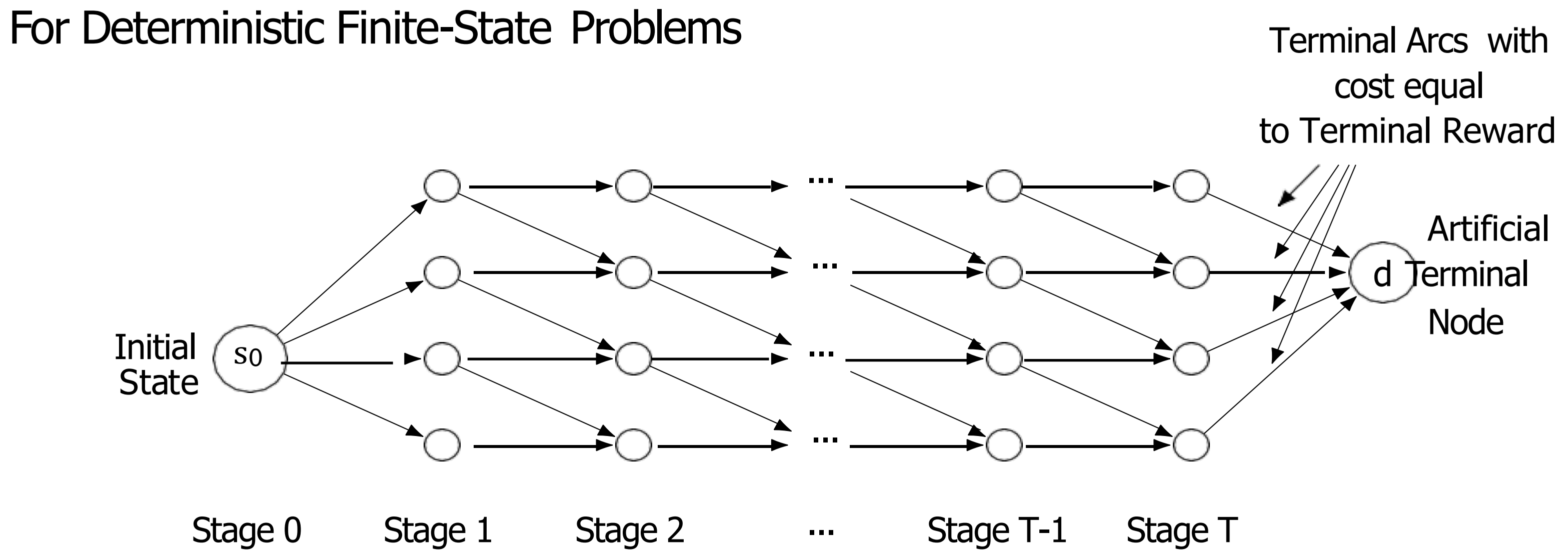
For Deterministic Finite-State Problems



Example: Breakout



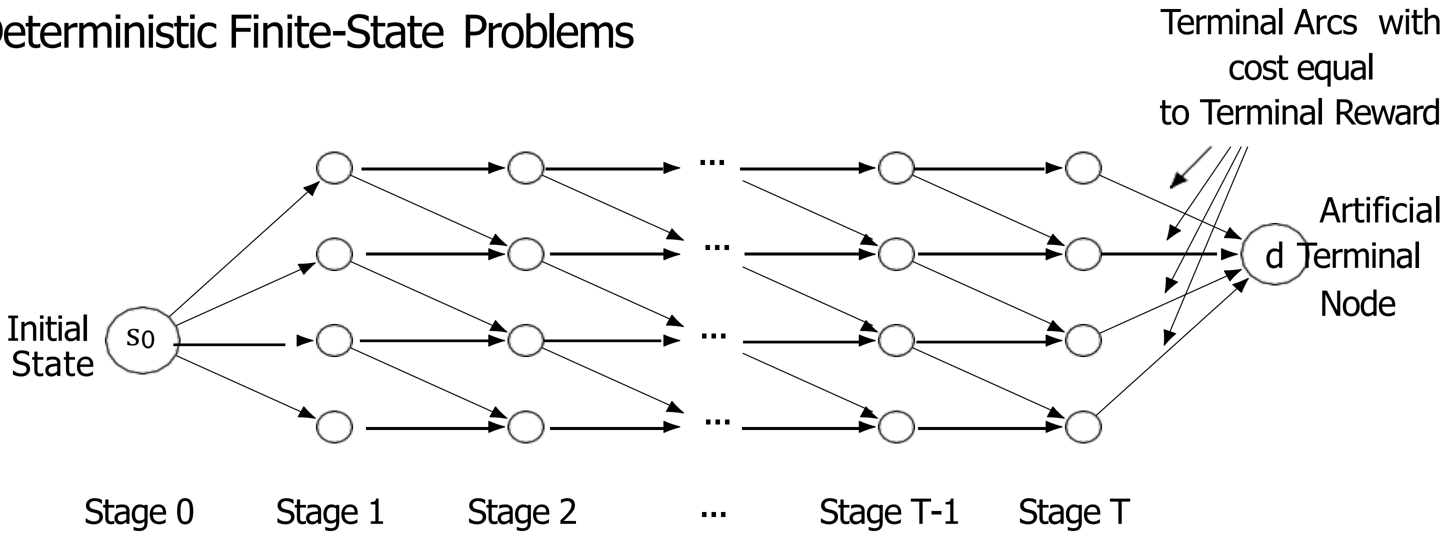
Sequential decision making as shortest path



Discuss: If shortest path isn't hard, why are DP problems still challenging?

Sequential decision making as shortest path

For Deterministic Finite-State Problems



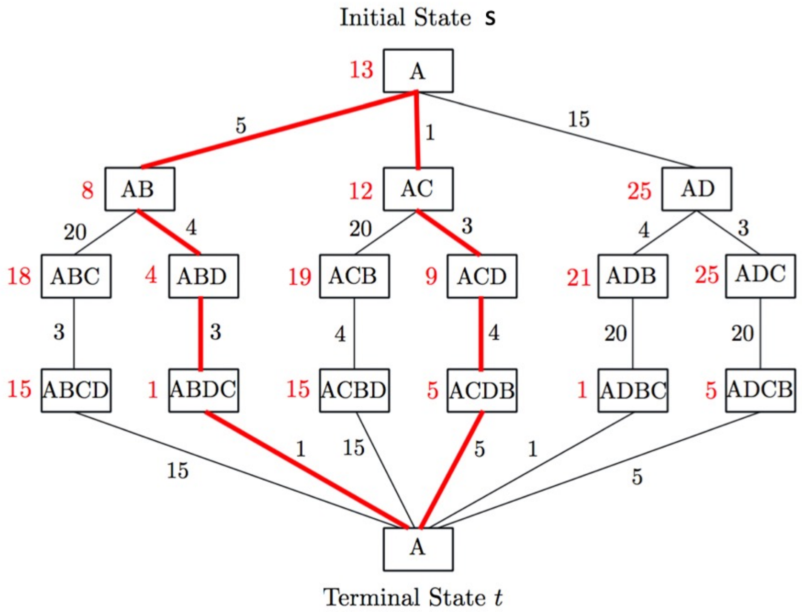
Example: Integer programming (combinatorial optimization)

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \in \{0,1\}^T \end{aligned}$$

Sequential decision making can get hairy

Example: traveling salesman problem (TSP)

- N cities.
- **Goal**: Find the shortest tour (visit every city exactly once and return home).
- In this case, can't get around exponential. (why?)
- $|S| = O(N!)$, $|A| = N$, $T = N$, so $O(|S||A|T) = O(N!)$.
- (Actually, DP *is* slightly better: $|S| = O(2^N N^2)$.)
- This is called the **curse of dimensionality**.



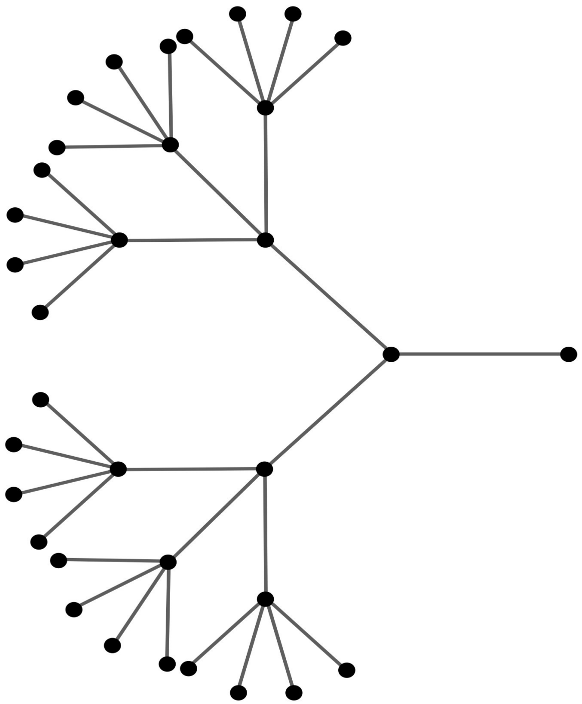
Matrix of Intercity Travel Costs

	5	1	15
5		20	4
1	20		3
15	4	3	

Sequential decision making can get hairy

Example: traveling salesman problem (TSP)

- N cities.
- **Goal**: Find the shortest tour (visit every city exactly once and return home).
- In this case, can't get around exponential. (why?)
- $|S| = O(N!)$, $|A| = N$, $T = N$, so $O(|S||A|T) = O(N!)$.
- (Actually, DP *is* slightly better: $|S| = O(2^N N^2)$.)
- This is called the **curse of dimensionality**.



Key challenge: huge decision spaces

- Arcade Learning Environment (ALE): framework that allows researchers and hobbyists to develop AI agents for Atari 2600 games
- Suppose the state is discretized at 10 x 20 and each cell takes one of 4 values: {ball, paddle, brick, empty}
- Possible game states: $4^{200} \approx 10^{120}$

$a_t = \text{left}$

t

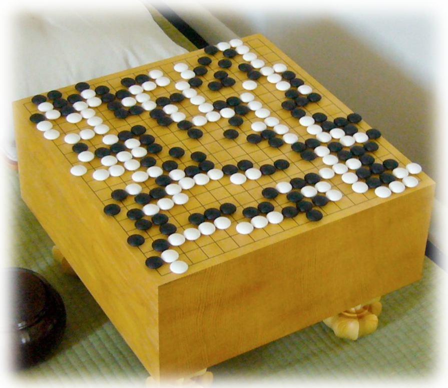
For reference:
There are between 10^{78} to 10^{82} atoms in the observable universe.

Cannot only explore. Cannot only exploit.
Must trade off exploration and exploitation.

Key challenge: huge decision spaces

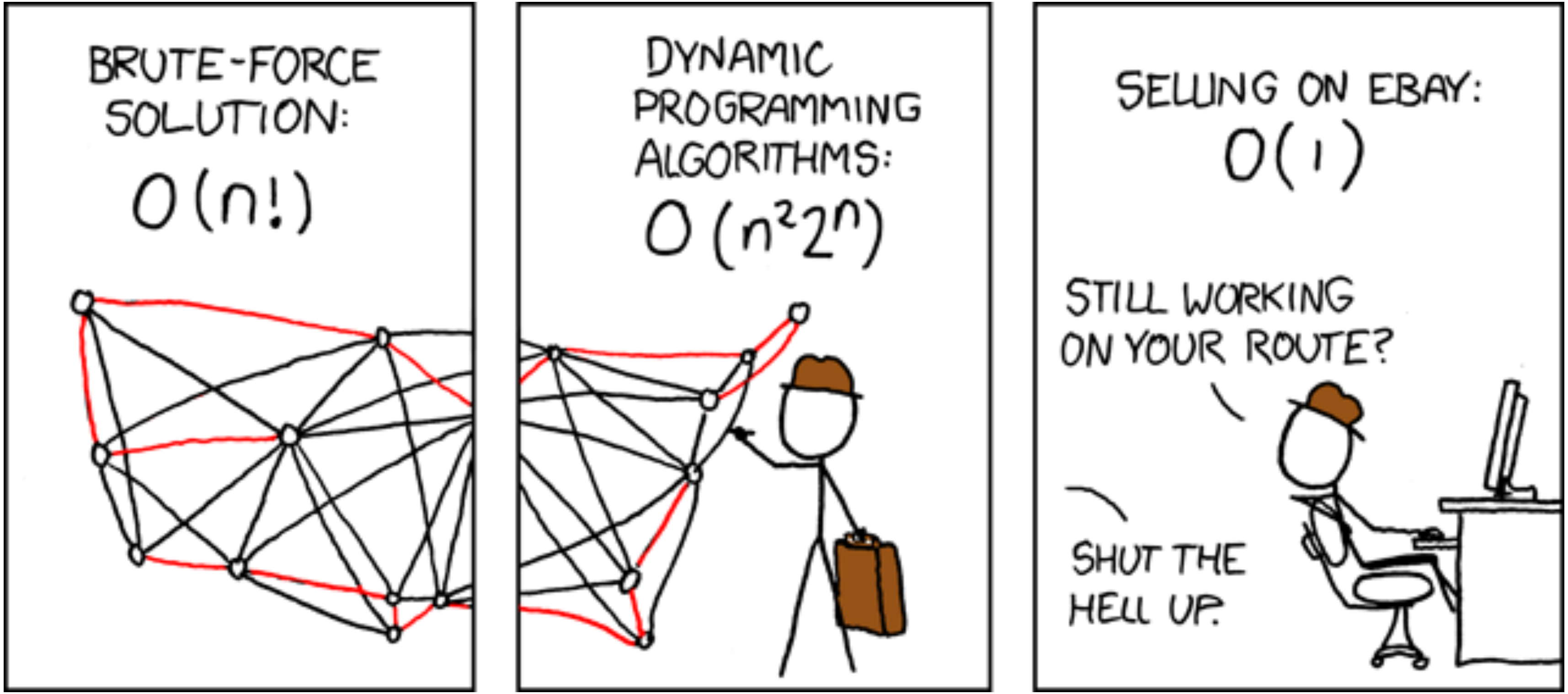
Go: 3^{19x19}

$\approx 10^{90}$ x (# atoms in universe)



For reference:
There are between 10^{78} to 10^{82} atoms in the observable universe.

Cannot only explore. Cannot only exploit.
Must trade off exploration and exploitation.



Travelling Salesman Problem (https://xkcd.com/399/)

Summary & takeaways

- The principle of optimality relates solving a sequential decision problem to smaller ”future” subproblems (called tail subproblems).
- Dynamic programming solves sequential decision problems by leveraging the principle of optimality. It applies in both deterministic and stochastic settings.
- The curse of dimensionality refers to the exponential growth in state spaces. This renders “efficient” dynamic programming algorithms insufficient for many problems of interest.

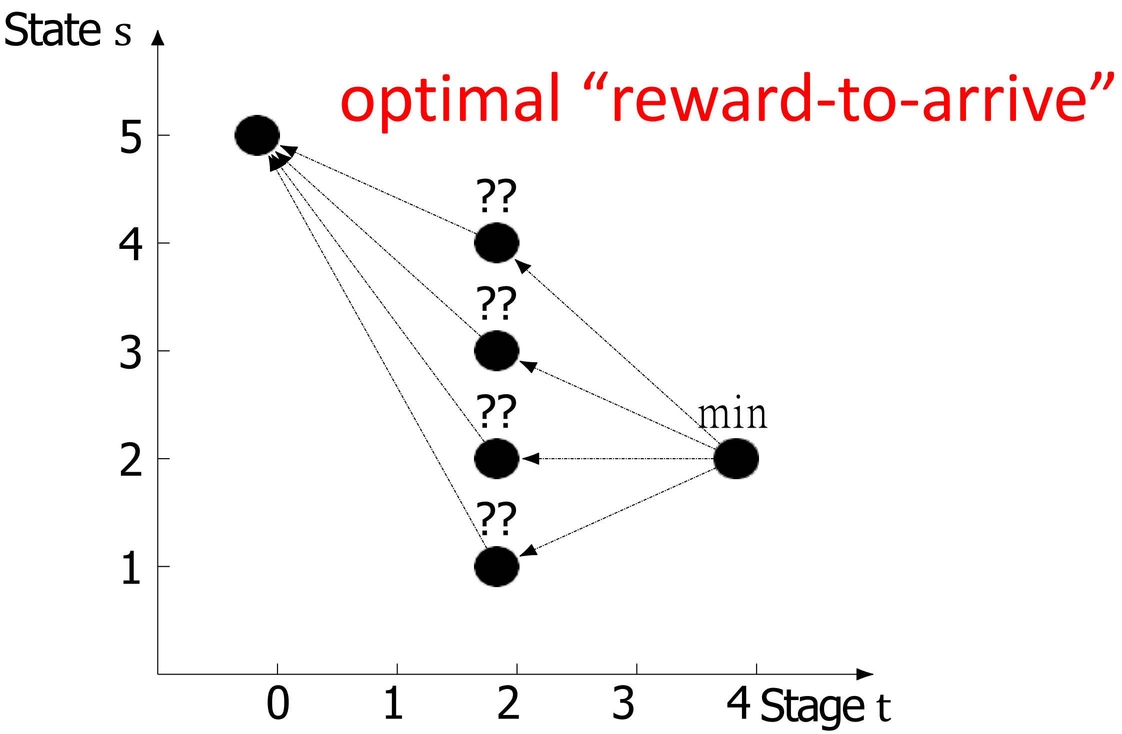
Outline

1. **Solving finite-horizon decision problems**
   a. Example: shortest path routing
   b. Dynamic programming algorithm
   c. Sequential decision making as shortest path
   d. **Forward DP**

Bonus: Forward dynamic programming algorithm?

Consider: ***stochastic*** shortest path routing

- Travel to intended city with probability $1 - \epsilon$.
- Travel to any city with probability ϵ.



Forward Dynamic Programming Algorithm?

$V_0(s_0) = r_0(s_0)$
for $t = 1, \dots, T$ **do**

$$V_t(s_t) = \max_{a_{t-1} \in \mathcal{A}_{t-1}(s_{t-1})} \mathbb{E}_{\epsilon_{t-1}}[r_t(s_t) + V_{t-1}(s_{t-1}) | s_t]$$
$$\text{s.t. } s_t = f_{t-1}(s_{t-1}, a_{t-1}, \epsilon_{t-1})$$

end for

Discuss: Does forward DP work? Why/why not? When/when not?

Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \ldots, 0$ **do**
$$V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}\left[r_t(s_t, a_t) + V_{t+1}(s_{t+1})\right]$$
end for

References

1. Some slides adapted from Alessandro Lazaric (FAIR/INRIA)
2. DPOC 3.3-3.4