2022-09-08

Dynamic programming

What makes sequential decision making hard?

Cathy Wu

6.7950: Reinforcement Learning: Foundations and Methods

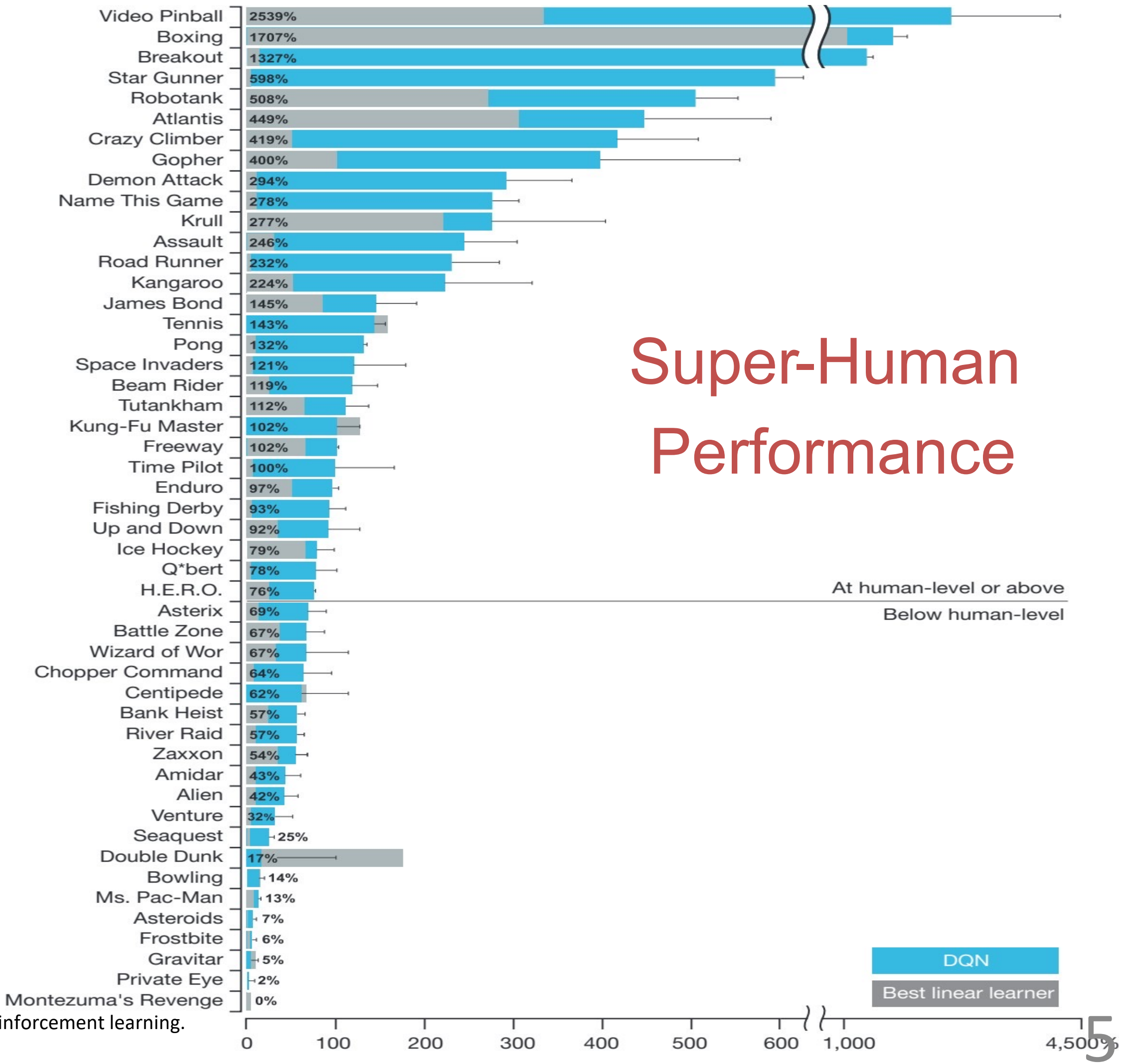
References

1. Some slides adapted from Alessandro Lazaric (FAIR/INRIA)
2. DPOC vol 1, 1.1-1.3, 2.1

Outline

1. Reinforcement learning to solve sequential decision problems
2. Formulation of finite-horizon decision problems
3. Solving finite-horizon decision problems
   a. Example: shortest path routing
   b. Dynamic programming algorithm
   c. Sequential decision making as shortest path
   d. Forward DP
4. Course overview
   a. Administrivia

2015:

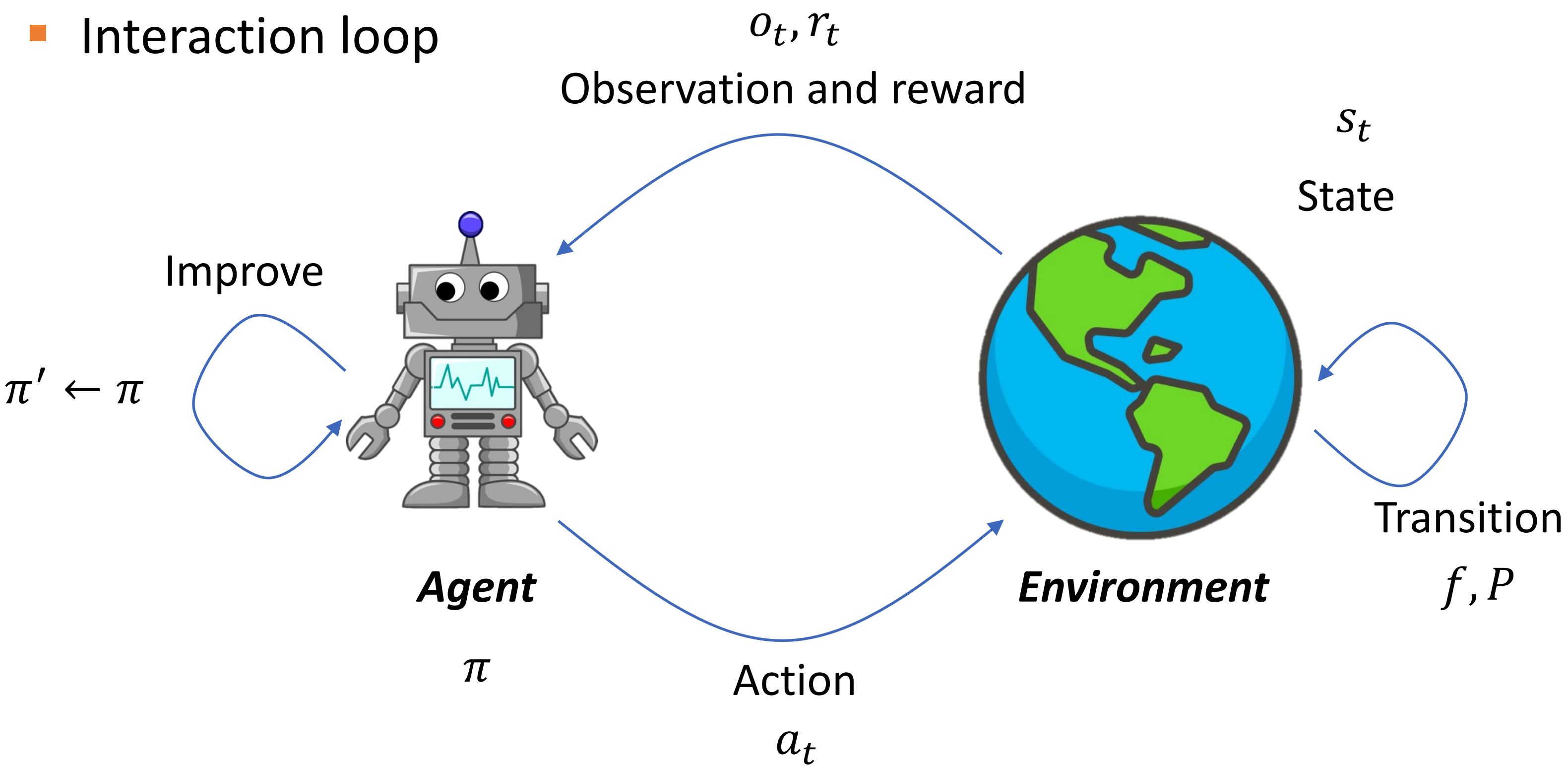


Super-Human Performance

Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

Introduce the characters*

- Interaction loop

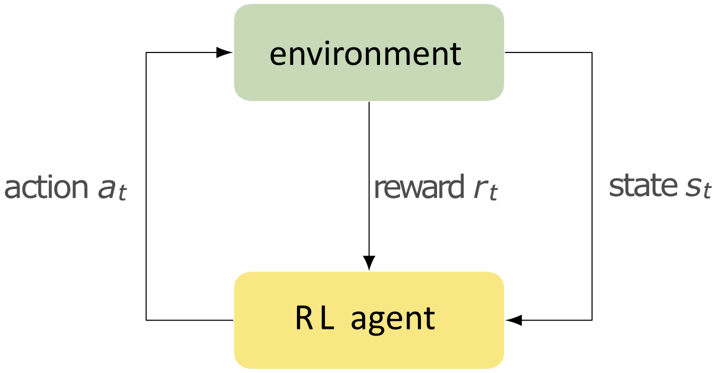


Goal: maximize reward over time (returns, cumulative reward)

* pun intended

What: Reinforcement Learning

Also known as *approximate dynamic programming* (ADP). We will use these terms more-or-less interchangeably.



"**Reinforcement learning** is learning how to map states to actions so as to **maximize** a numerical **reward** signal in an unknown and **uncertain** environment.

In the most interesting and challenging cases, **actions** affect not only the immediate reward but also the **next situation** and all subsequent rewards (**delayed reward**).

The agent is not told which actions to take but it must discover which actions yield the most reward by trying them (**trial-and-error**)."

— Sutton and Barto (1998)

“No simple yet reasonable evaluation function will ever be found for Go.”

-- 2002, Martin Müller
(winner of 2009 Go program competition)

2016:

ARTICLE

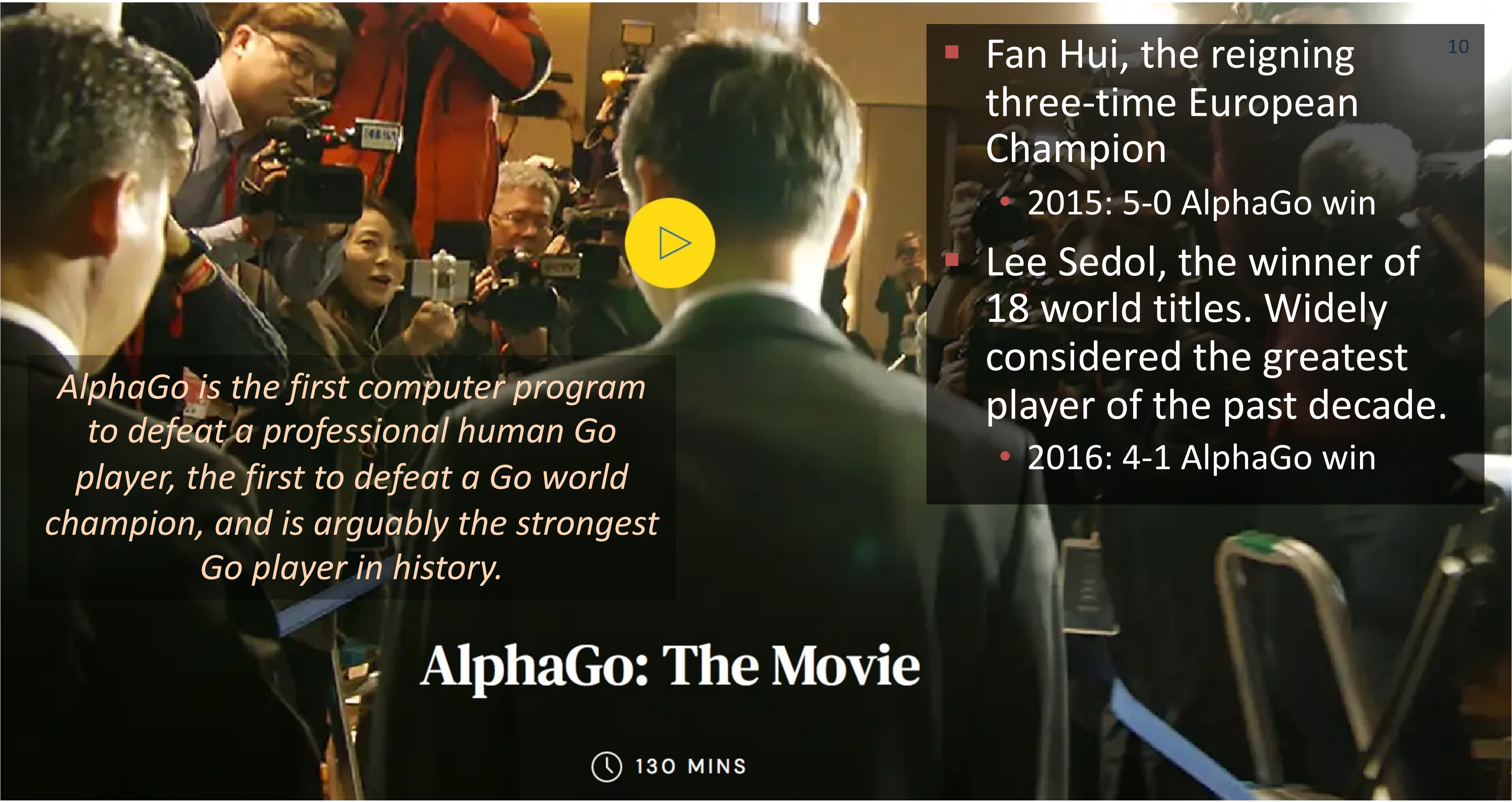
doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

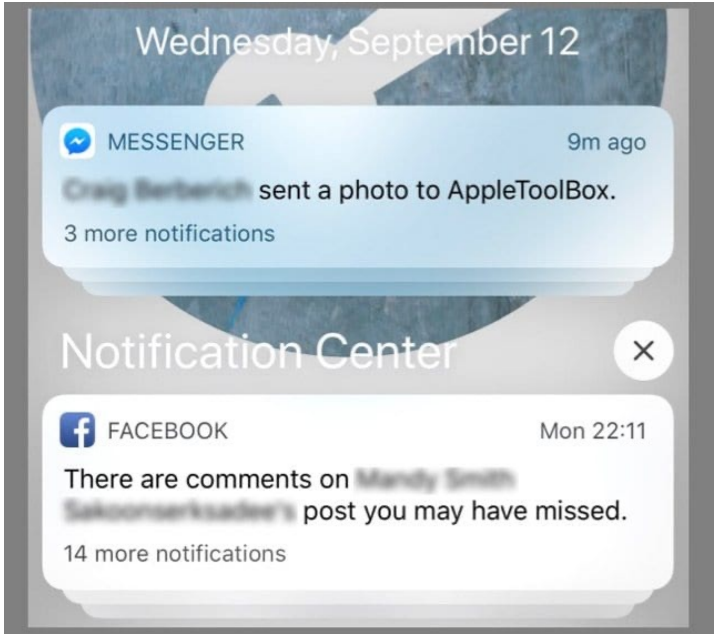
- Fan Hui, the reigning three-time European Champion
  - 2015: 5-0 AlphaGo win
- Lee Sedol, the winner of 18 world titles. Widely considered the greatest player of the past decade.
  - 2016: 4-1 AlphaGo win

AlphaGo is the first computer program to defeat a professional human Go player, the first to defeat a Go world champion, and is arguably the strongest Go player in history.



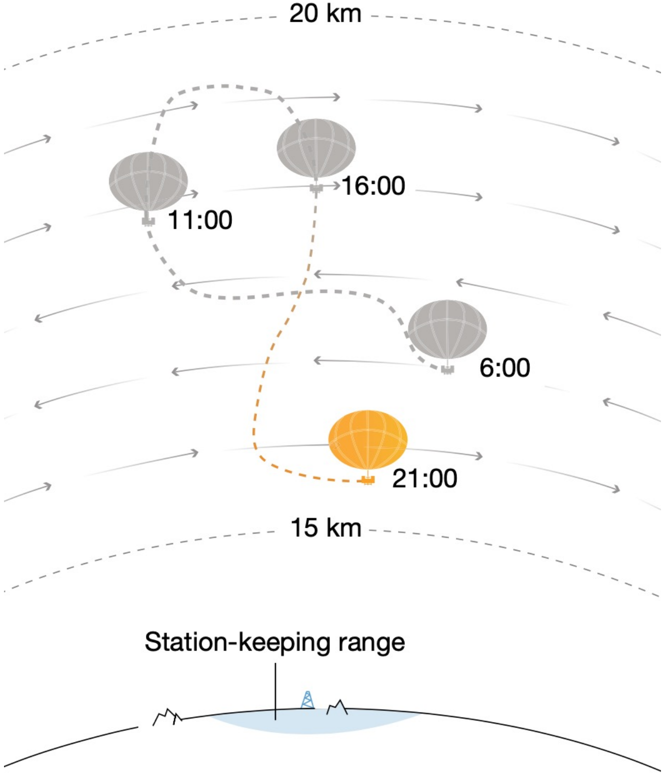
https://www.youtube.com/watch?v=WXuK6gekU1Y

Push notifications (2020)



Gauci, et al., “Horizon: Facebook's Open Source Applied Reinforcement Learning Platform - Facebook Research” (2020)

High-altitude balloons (2020)



Bellemare et al., “Autonomous navigation of stratospheric balloons using reinforcement learning” Nature, 2020.

Traffic flow smoothing (2021)


Sugiyama, et al. 2008

Wu, et al. "Flow: A Modular Learning Framework for Mixed Autonomy Traffic." T-RO, 2021.

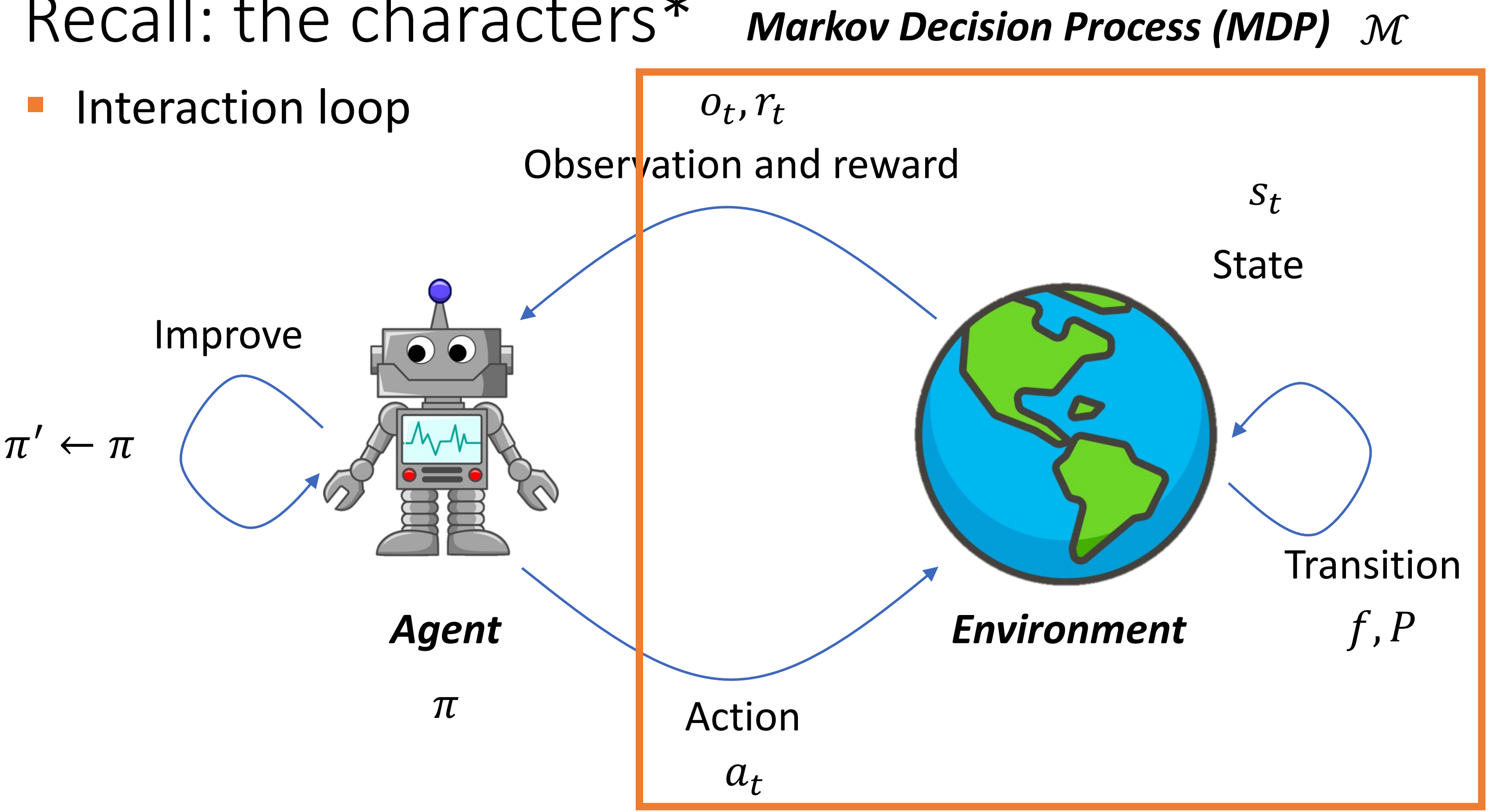
Q: What applications are you excited about?

Outline

1. Reinforcement learning to solve sequential decision problems
2. **Formulation of finite-horizon decision problems**
3. Solving finite-horizon decision problems
   a. Example: shortest path routing
   b. Dynamic programming algorithm
   c. Sequential decision making as shortest path
   d. Forward DP
4. Course overview
   a. Administrivia

Recall: the characters*

- Interaction loop



Goal: maximize reward over time (returns, cumulative reward)

* pun intended

Assume for now: finite horizon problems, i.e. $T < \infty$

Used when: there is an intrinsic deadline to meet.

Later: infinite horizon

The value function

Given a policy π (deterministic to simplify notation)

- Finite time horizon T: deadline at time T, the agent focuses on the sum of the rewards up to T.

$$V^{\pi}(t,s) = \mathbb{E}\left[\sum_{\tau=t}^{T-1} r(s_\tau, \pi \quad (a_\tau)) + R(s_T) | s_t = s; \pi\right]$$

  where R is a value function for the final state.

- Shorthand: $V_t^{\pi}(s)$ or simply V_t^{π} (think: vector of size $|S|$)

Optimization Problem

- Our goal: achieve the best value
  - Max value-to-go (min cost-to-go)

Definition (Optimal policy and optimal value function)

The solution to an MDP is an optimal policy π^* satisfying

$$\pi^* \in \arg\max_{\pi \in \Pi} V_0^{\pi}$$

where Π is some policy set of interest.

The corresponding value function is the optimal value function

$$V^* = V_0^{\pi^*}$$

Expectations

- Technical note: the expectations refer to all possible stochastic trajectories.
- A (possibly non-stationary stochastic) policy π applied from state s_0 returns

$$(s_0, r_0, s_1, r_1, s_2, r_2, \dots)$$

- Where $r_t = r(s_t, a_t)$ and $s_{t+1} \sim p(\cdot \mid s_t, a_t = \pi_t(s_t))$ are random realizations.

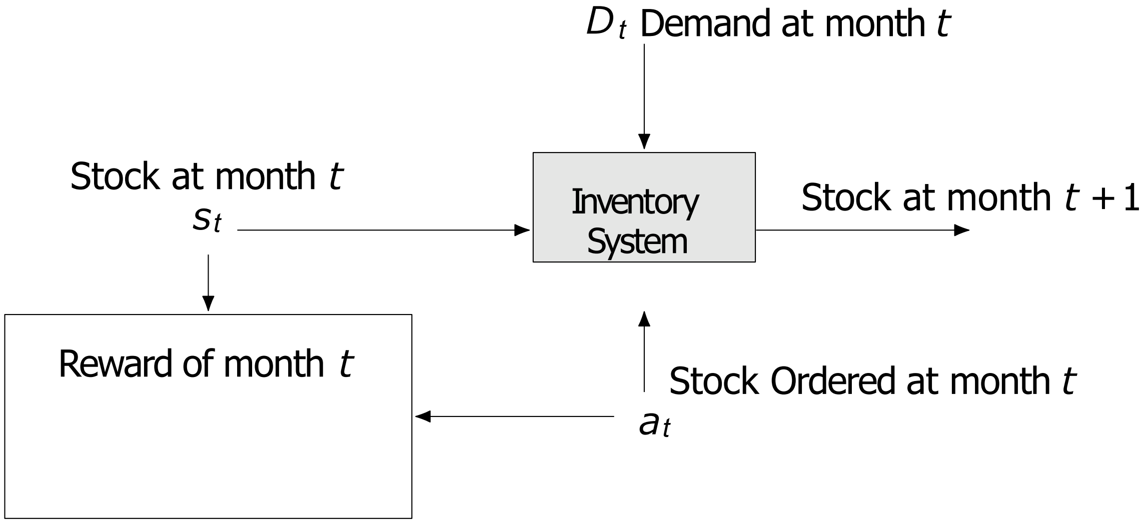
- The value function is

$$V^\pi(t,s) = \mathbb{E}_{(s_1,s_2,\dots)}\left[\sum_{\tau=t}^{T-1} r(s_\tau, \pi \quad (a_\tau)) + R(s_T) | s_t = s; \pi\right]$$

- More generally, for stochastic policies:

$$V^\pi(t,s) = \mathbb{E}_{(a_0,s_1,a_1,s_2,\dots)}\left[\sum_{\tau=t}^{T-1} r(s_\tau, \pi \quad (a_\tau)) + R(s_T) | s_t = s; \pi\right]$$

Example: The Amazing Goods Company Example



Example: The Amazing Goods Company Example

- *Description.* At each month t, a warehouse contains s_t *items* of a specific goods and the demand for that goods is D (stochastic). At the end of each month the manager of the warehouse can *order* a_t more items from the supplier.

- The cost of maintaining an inventory of s is $h(s)$.
- The cost to order a items is $C(a)$.
- The income for selling q items if $f(q)$.
- If the demand $d \sim D$ is bigger than the available inventory s, customers that cannot be served leave.
- The value of the remaining inventory at the end of the year is $g(s)$.
- Constraint: the store has a maximum capacity C.

Recall: Markov Chains

Definition (Markov chain)

Let the *state space S* be a subset of the Euclidean space, the discrete-time dynamic system $(s_t)_{t\in\mathbb{N}} \in S$ is a Markov chain if it satisfies the *Markov property*

$$P(s_{t+1} = s | s_t, s_{t-1}, ..., s_0) = P(s_{t+1} = s | s_t),$$

Given an initial state $s_0 \in S$, a Markov chain is defined by the *transition probability*

$$p \quad p(s'|s) = P(s_{t+1} = s' | s_t = s).$$

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,

Example: The Amazing Goods Company

- State space: $s \in S = \{0, 1, \ldots, C\}$.

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \ or \ f, r, H)$ where

- S is the *state* space,
- A is the *action* space,

Example: The Amazing Goods Company

- Action space: it is not possible to order more items than the capacity of the store, so the action space should depend on the current state. Formally, at state s, $a \in A(s) = \{0, 1, \dots, C - s\}$.

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,
- A is the *action* space,
- $P(s'|s,a)$ is the transition probability with

often simplified to finite

$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$

transition equation

$$s' = f_t(s, w_t)$$

where $w_t \sim W_t$

Example: The Amazing Goods Company

- Dynamics: $s_{t+1} = [s_t + a_t - d_t]^+$.
- The demand d_t is stochastic and time-independent. Formally, $d_t \overset{i.i.d}{\sim} D$.

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,
- A is the *action* space,

often simplified to finite

- $P(s'|s,a)$ is the transition probability with

$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$

- $r(s,a,s')$ is the immediate reward at state s upon taking action a,

sometimes simply $r(s)$

Example: The Amazing Goods Company

- Reward: $r_t = -C(a_t) - h(s_t + a_t) + f([s_t + a_t - s_{t+1}]^+)$. This corresponds to a purchasing cost, a cost for excess stock (storage, maintenance), and a reward for fulfilling orders.

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,
- A is the *action* space,

often simplified to finite

- $P(s'|s,a)$ is the transition probability with
$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$
- $r(s,a,s')$ is the immediate reward at state s upon taking action a,

sometimes simply $r(s)$

- H is the horizon.

Example: The Amazing Goods Company

- The horizon of the problem is 12 (12 months in 1 year).

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,
- A is the *action* space,

often simplified to finite

- $P(s'|s,a)$ is the transition probability with

$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$

- $r(s,a,s')$ is the immediate reward at state s upon taking action a,

sometimes simply $r(s)$

- H is the horizon.

Example: The Amazing Goods Company

- Objective: $V(s_0; a_0, \dots) = \sum_{t=0}^{H-1} r_t + r_H$, where $\mathrm{r}_{12} = \mathrm{g}(s_{12})$. This corresponds to the cumulative reward, including the value of the remaining inventory at "the end."

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,
- A is the *action* space,

often simplified to finite

- $P(s'|s,a)$ is the transition probability with

$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$

- $r(s,a,s')$ is the immediate reward at state s upon taking action a,

sometimes simply $r(s)$

- H is the horizon.

☞ In general, a non-Markovian decision process's transitions could depend on much more information:

$$\mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a, s_{t-1}, a_{t-1}, \dots, s_0, a_0),$$

Markov Decision Process

Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple $M = (S, A, P \text{ or } f, r, H)$ where

- S is the *state* space,
- A is the *action* space,

often simplified to finite

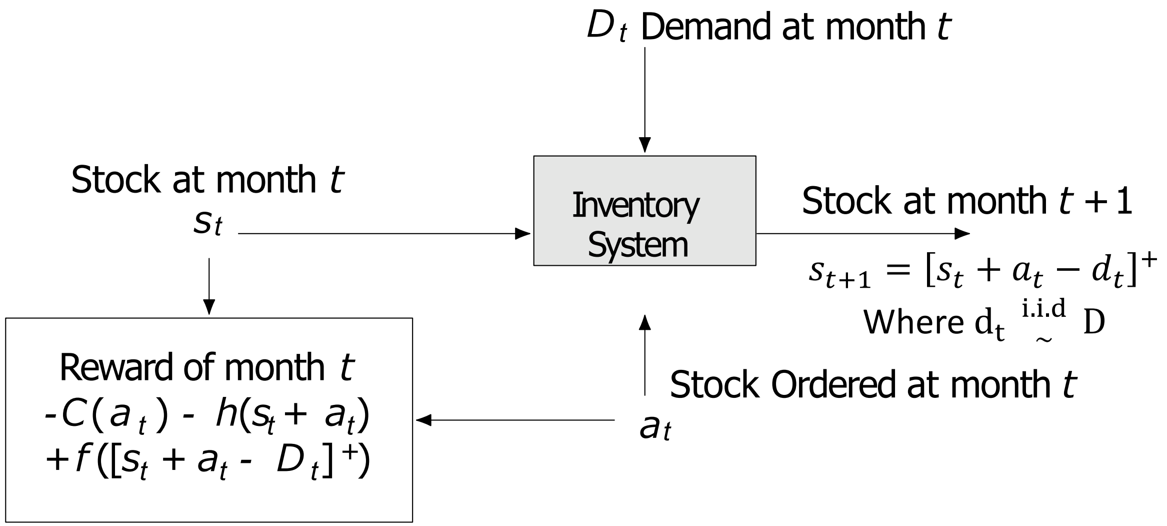
- $P(s'|s,a)$ is the transition probability with
$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a)$$
- $r(s,a,s')$ is the immediate reward at state s upon taking action a,

sometimes simply $r(s)$

- H is the horizon.

☞ The process generates trajectories $\tau_t = (s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)$, with $s_{t+1} \sim P(\cdot\,|s_t, a_t)$

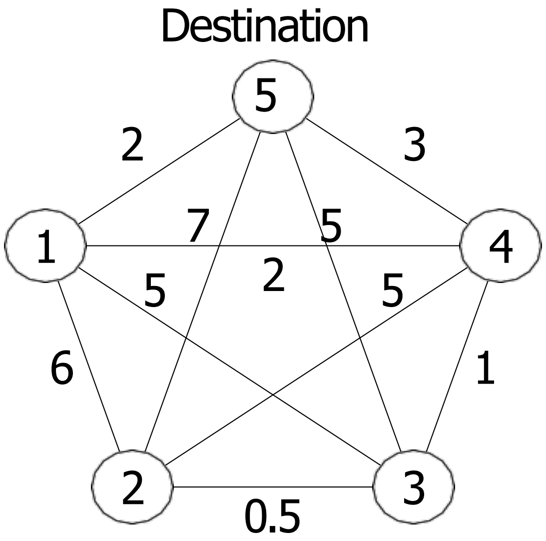
Example: The Amazing Goods Company Example



- State space: $s \in S = \{0, 1, \dots, C\}$.
- Action space: it is not possible to order more items than the capacity of the store, so the action space should depend on the current state. Formally, at state s, $a \in A(s) = \{0, 1, \dots, C - s\}$.
- Objective: $V(s_0; a_0, \dots) = \sum_{t=0}^{H-1} r_t + r_H$, where $H = 12$ and $r_{12} = g(s_{12})$

Outline

Example: Shortest Path Problem



Destination is node 5.

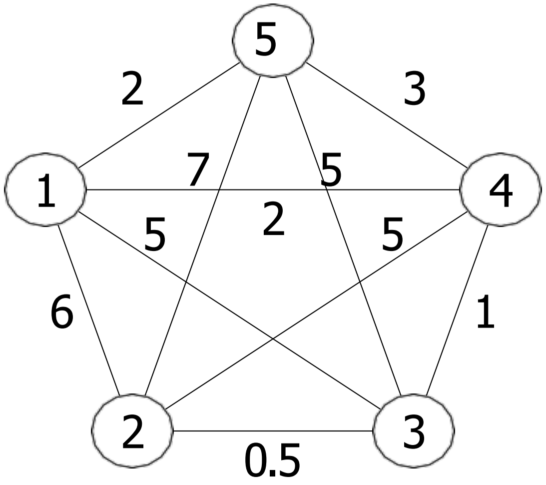
Sequential decision problem

- 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

Assumption: all cycles have non-negative length.

Destination

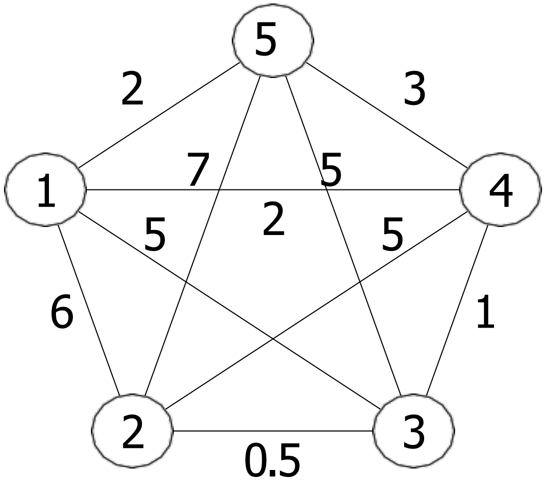


Destination is node 5.

- **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

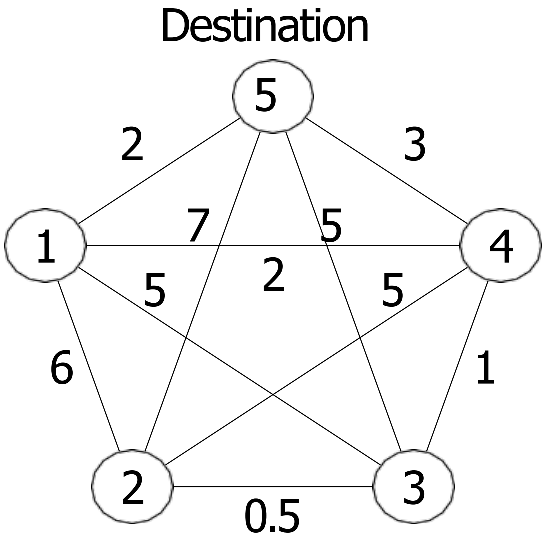
Destination



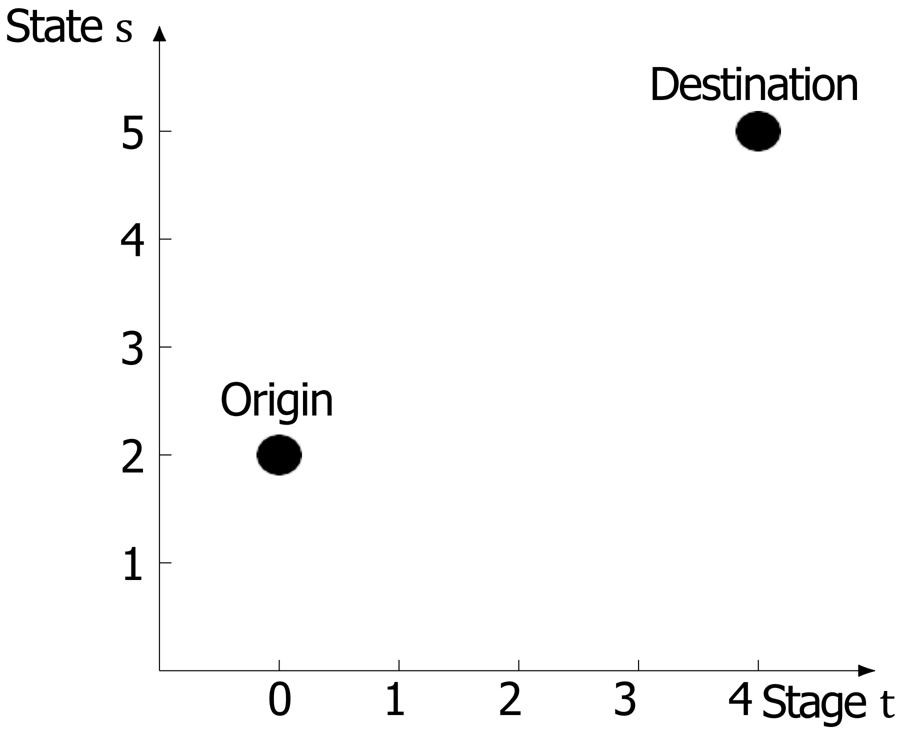
Destination is node 5.

- **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.

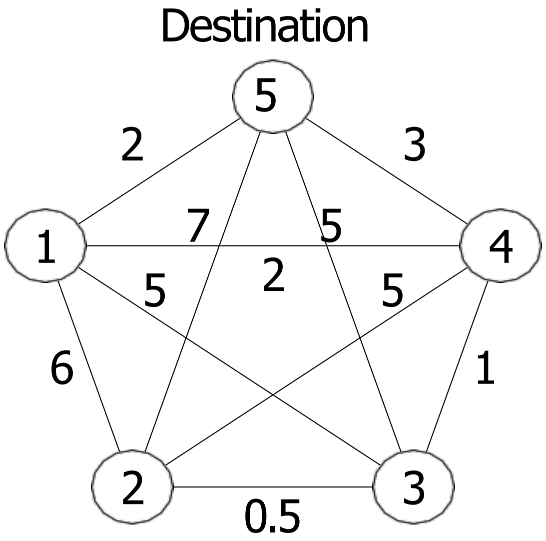
Solving Shortest Path



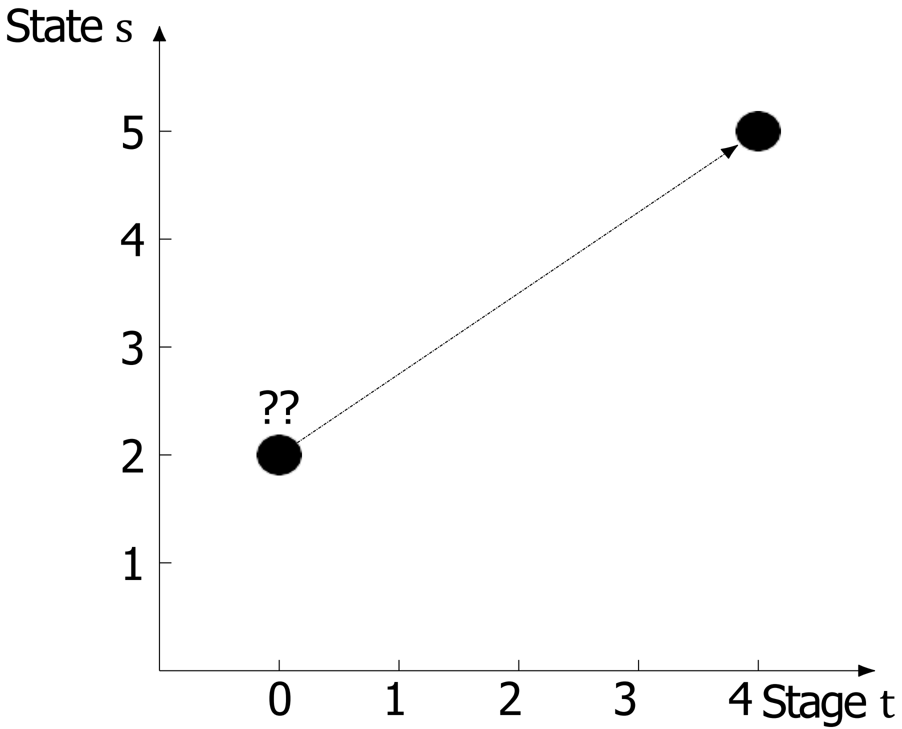
Destination is node 5.



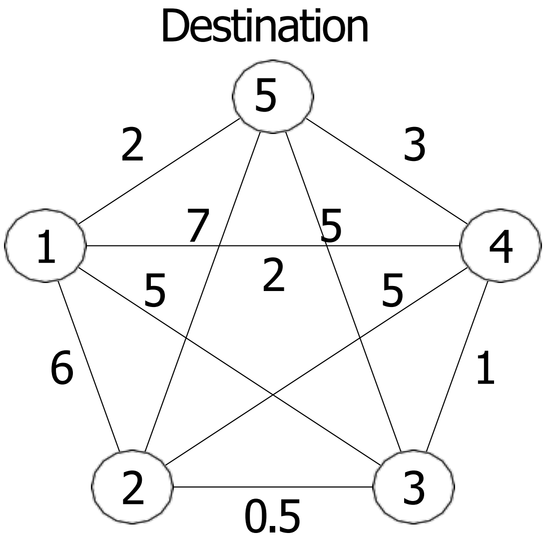
Solving Shortest Path

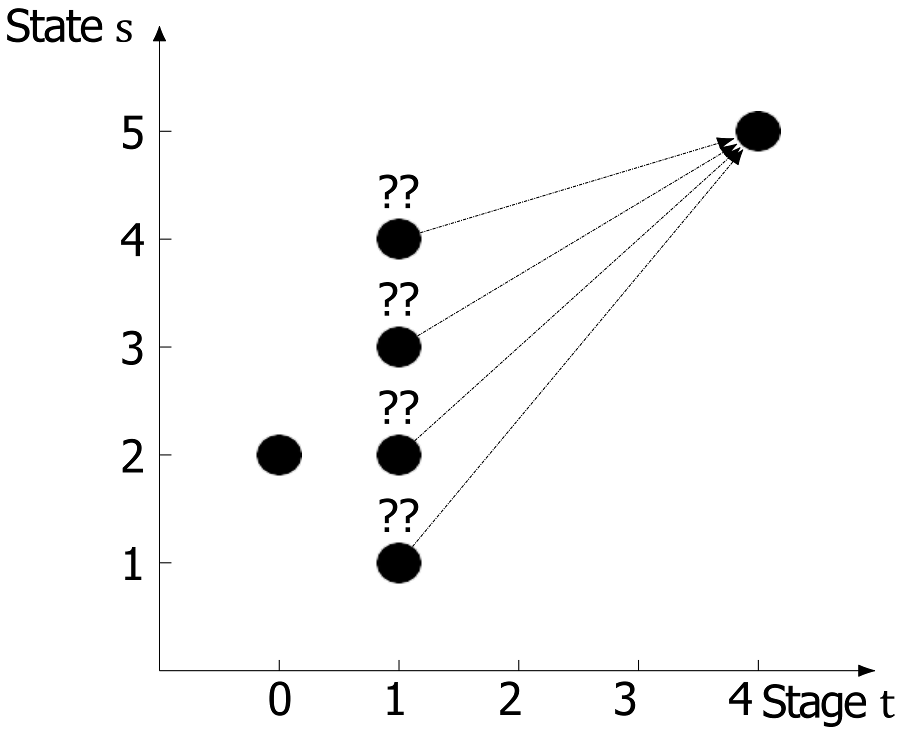


Destination is node 5.



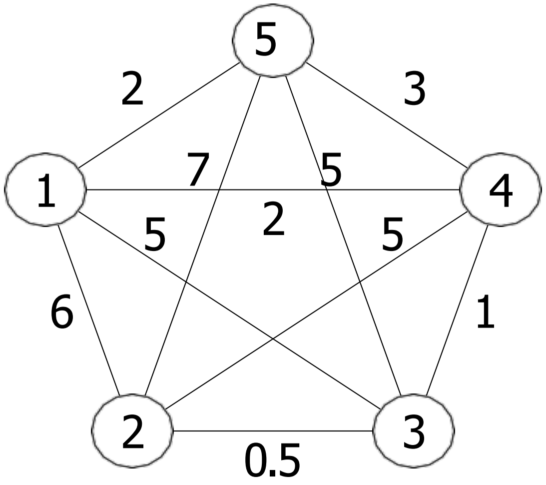
Solving Shortest Path



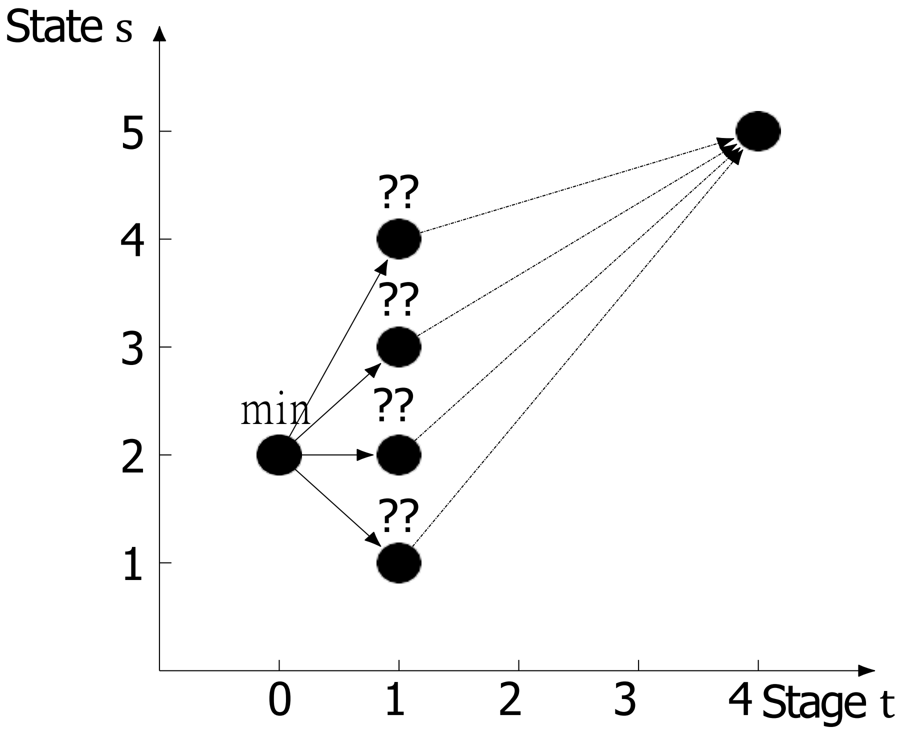


Destination is node 5.

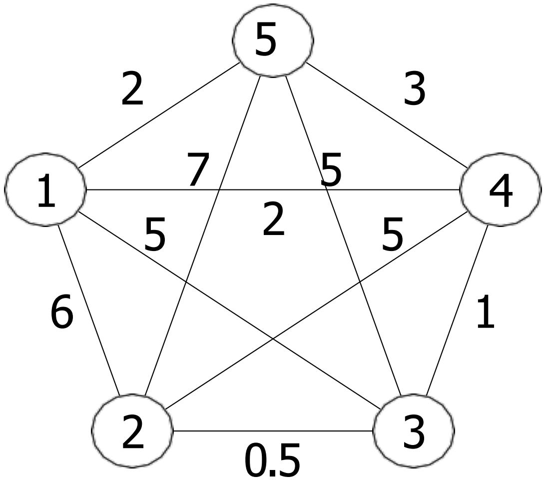
Solving Shortest Path



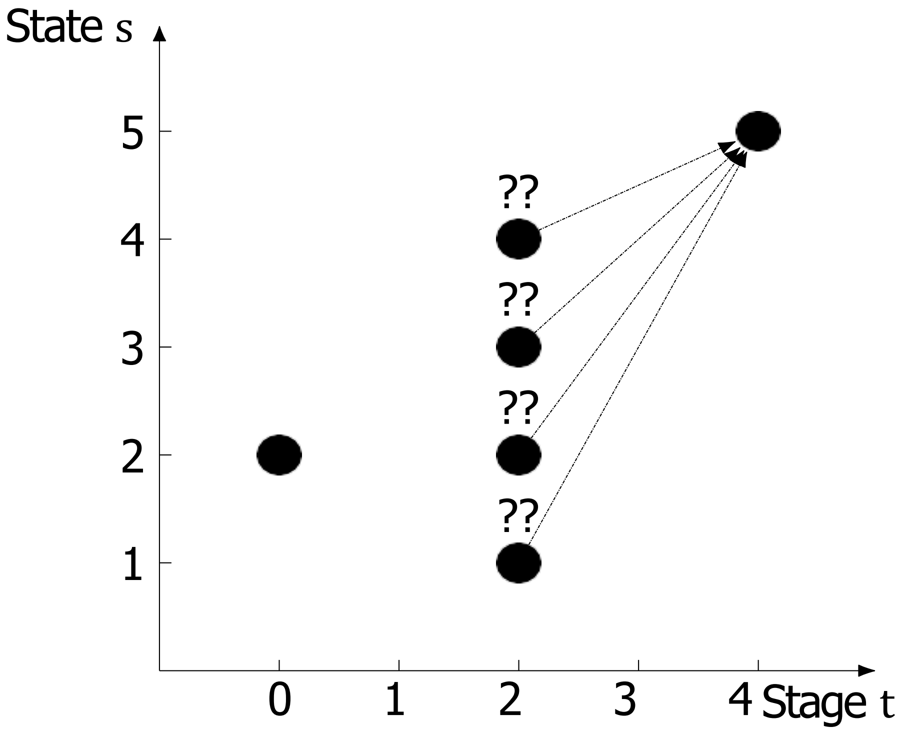
Destination is node 5.



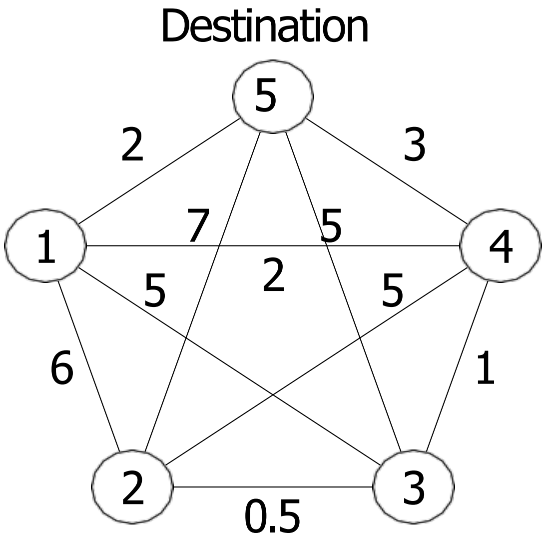
Solving Shortest Path

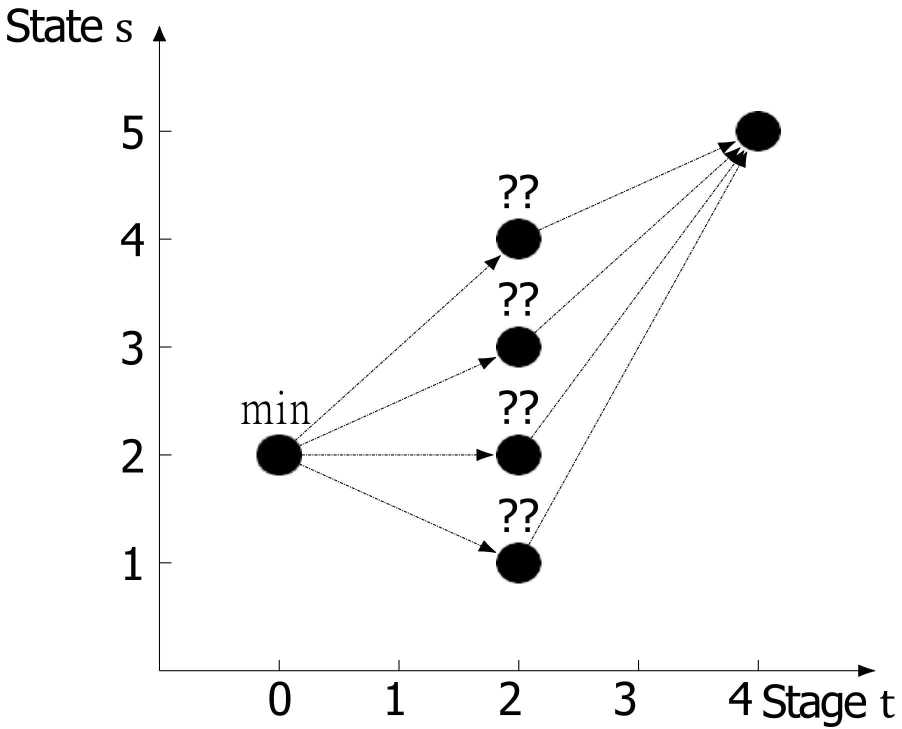


Destination is node 5.



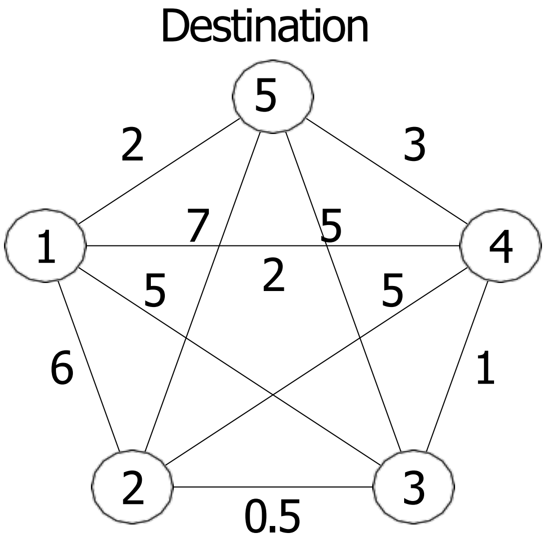
Solving Shortest Path



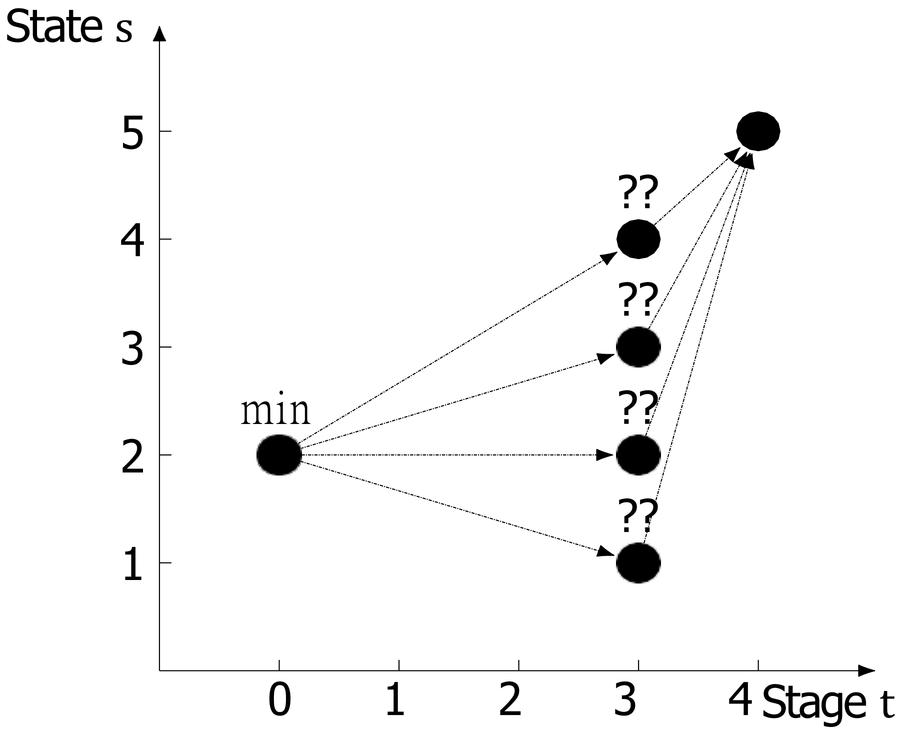


Destination is node 5.

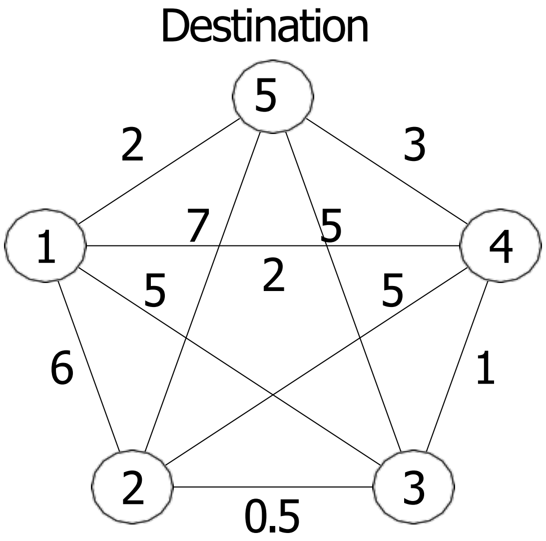
Solving Shortest Path



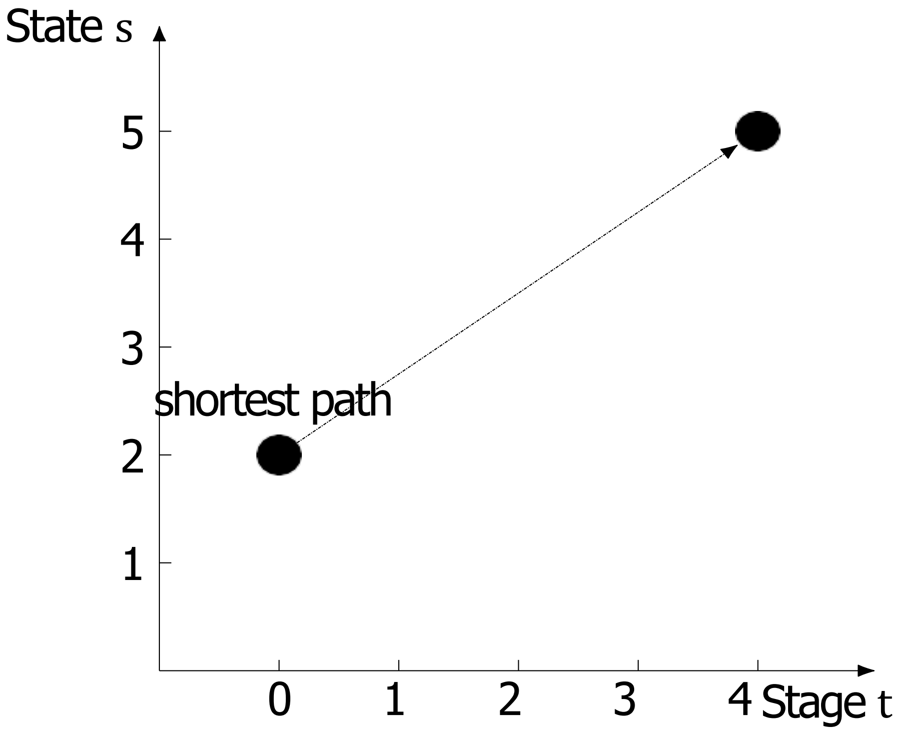
Destination is node 5.



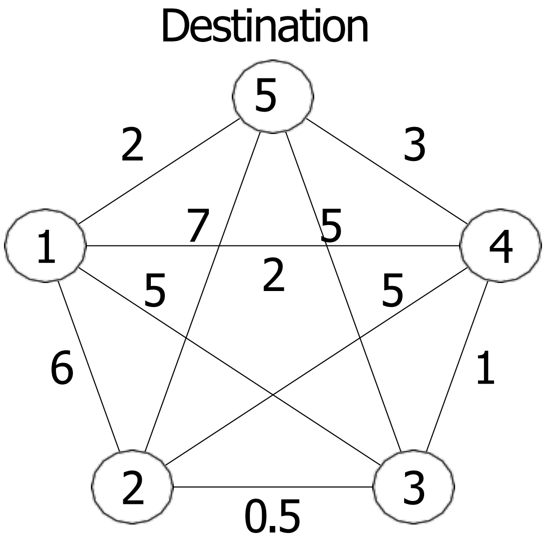
Solving Shortest Path

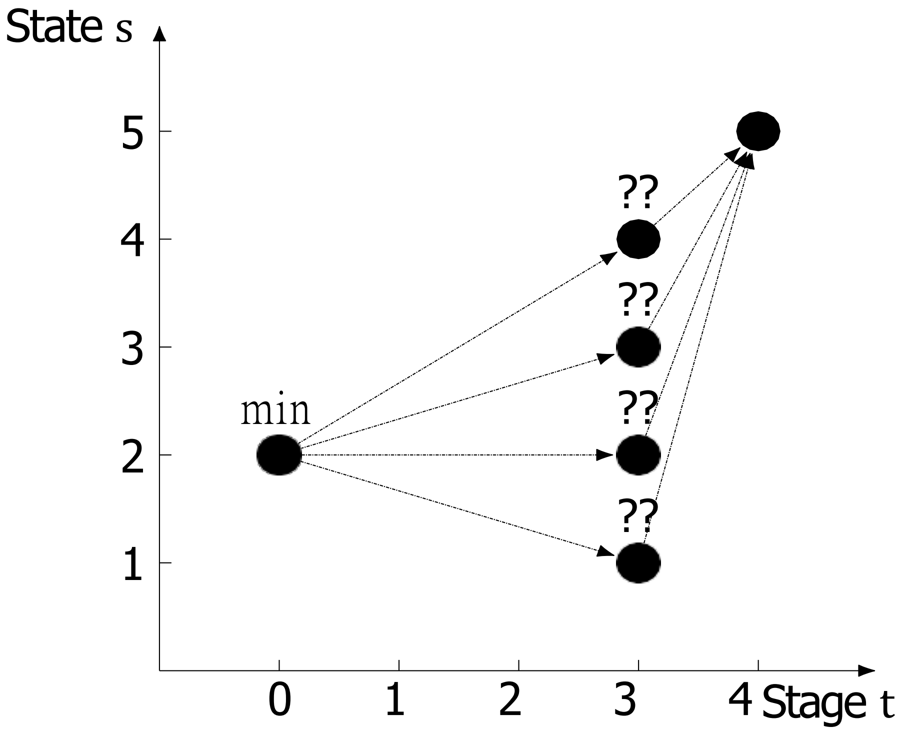


Destination is node 5.



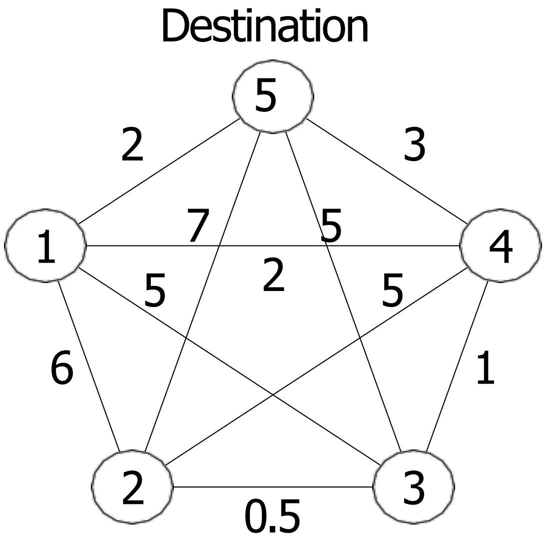
Solving Shortest Path



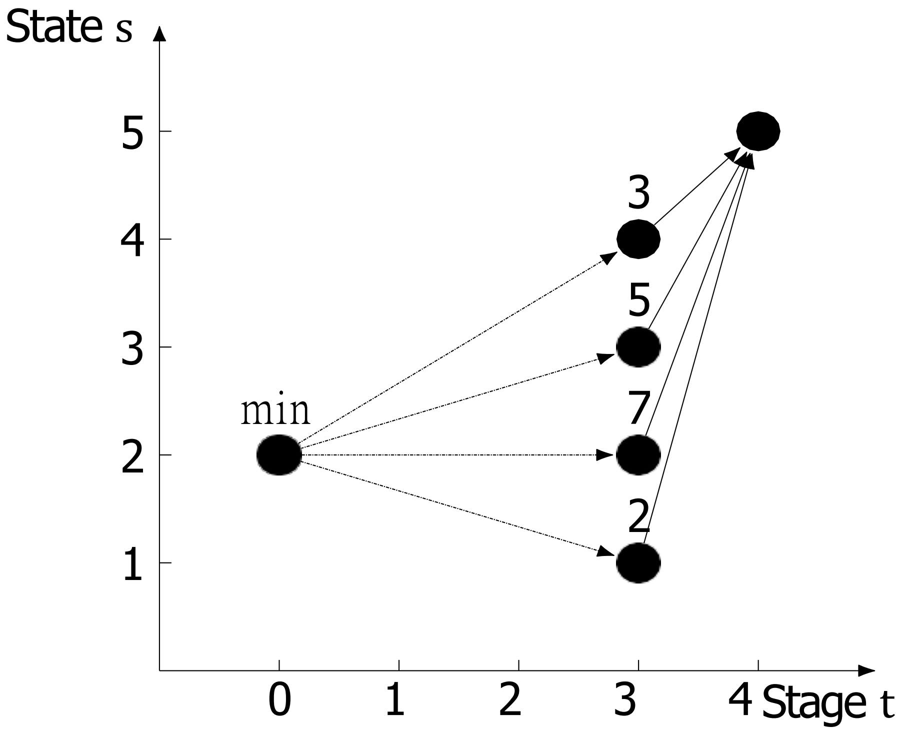


Destination is node 5.

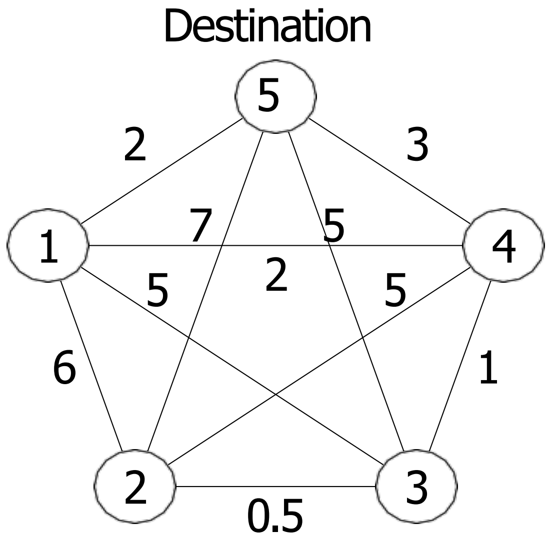
Solving Shortest Path



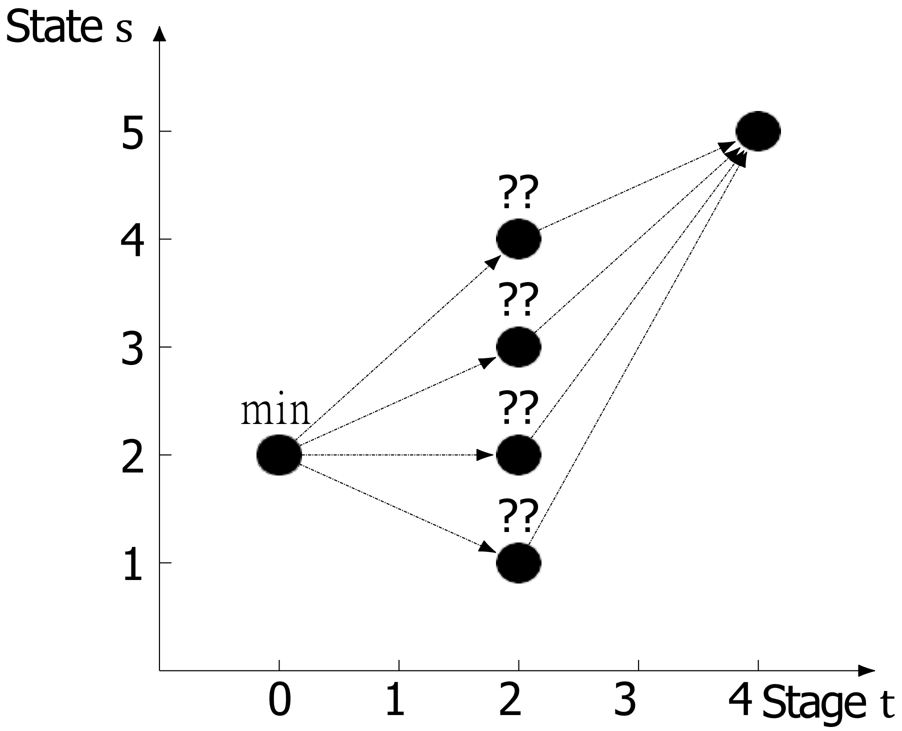
Destination is node 5.



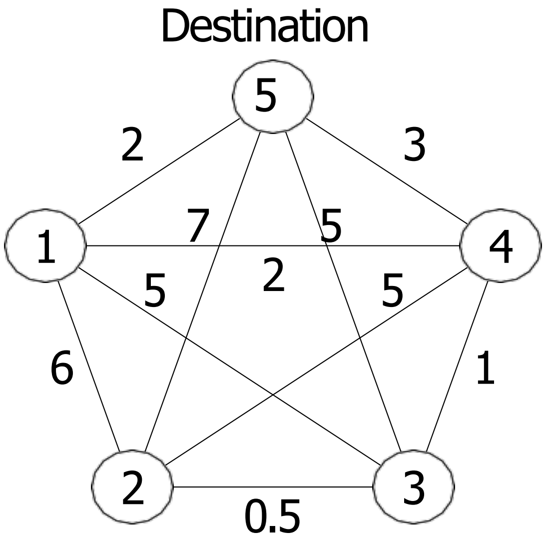
Solving Shortest Path



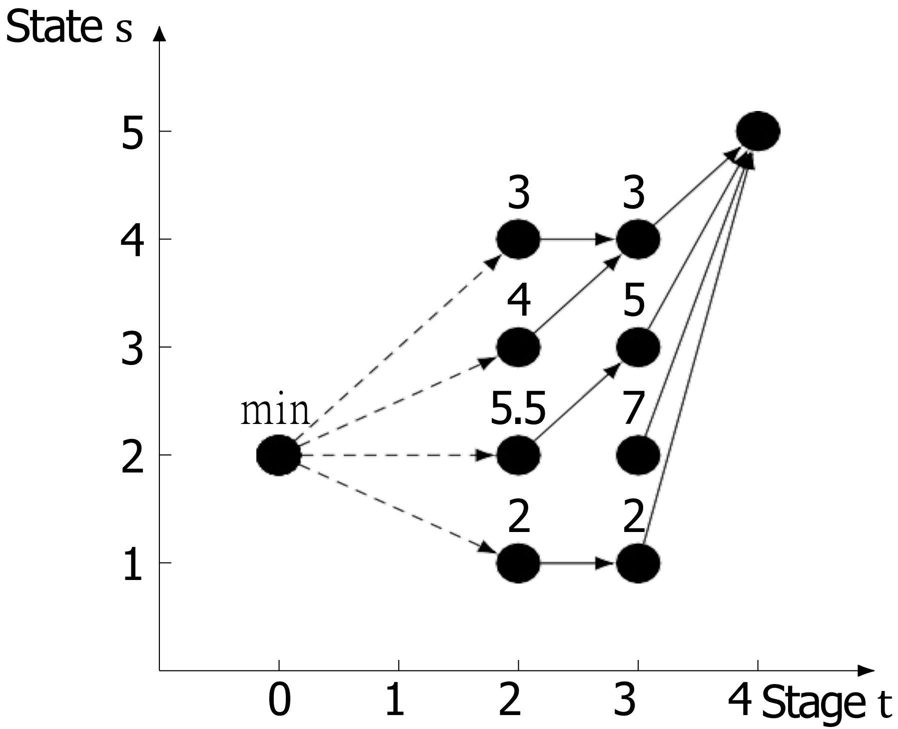
Destination is node 5.



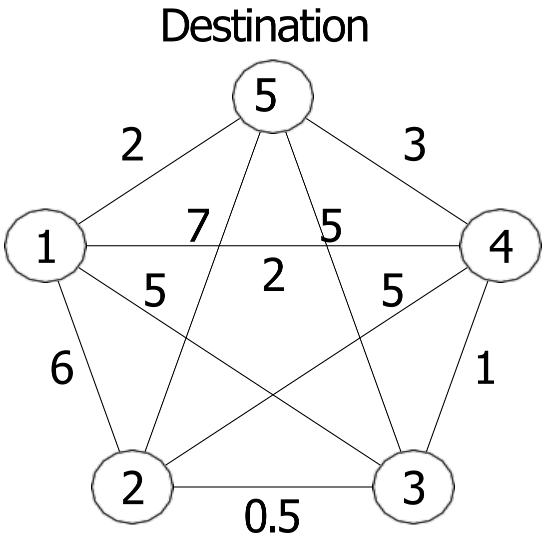
Solving Shortest Path



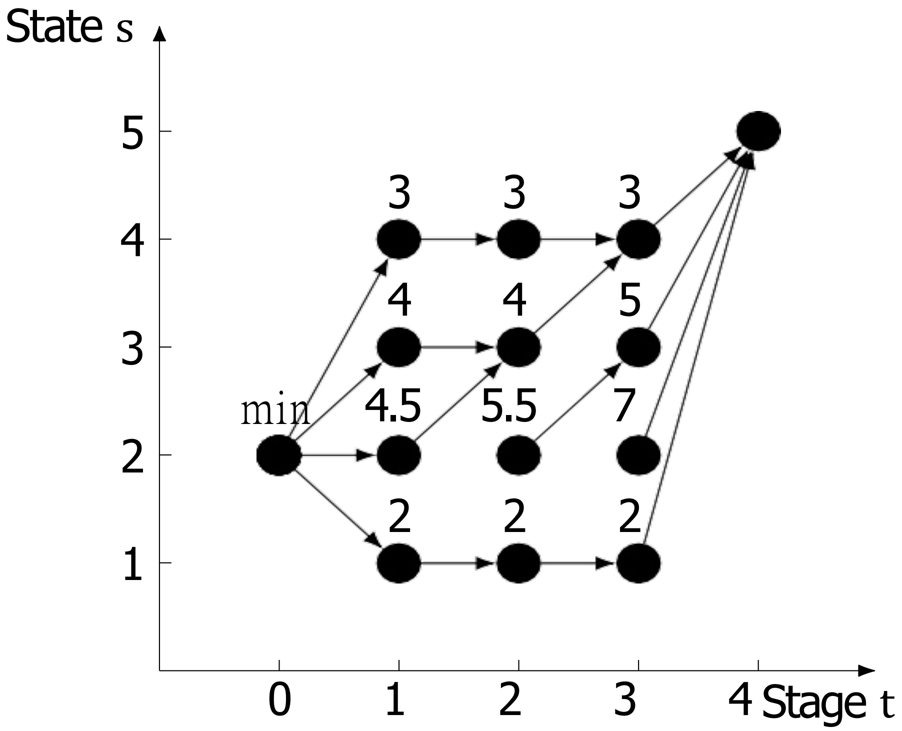
Destination is node 5.



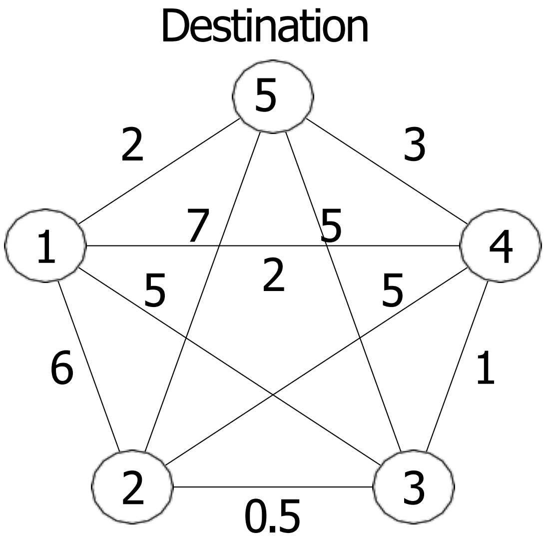
Solving Shortest Path



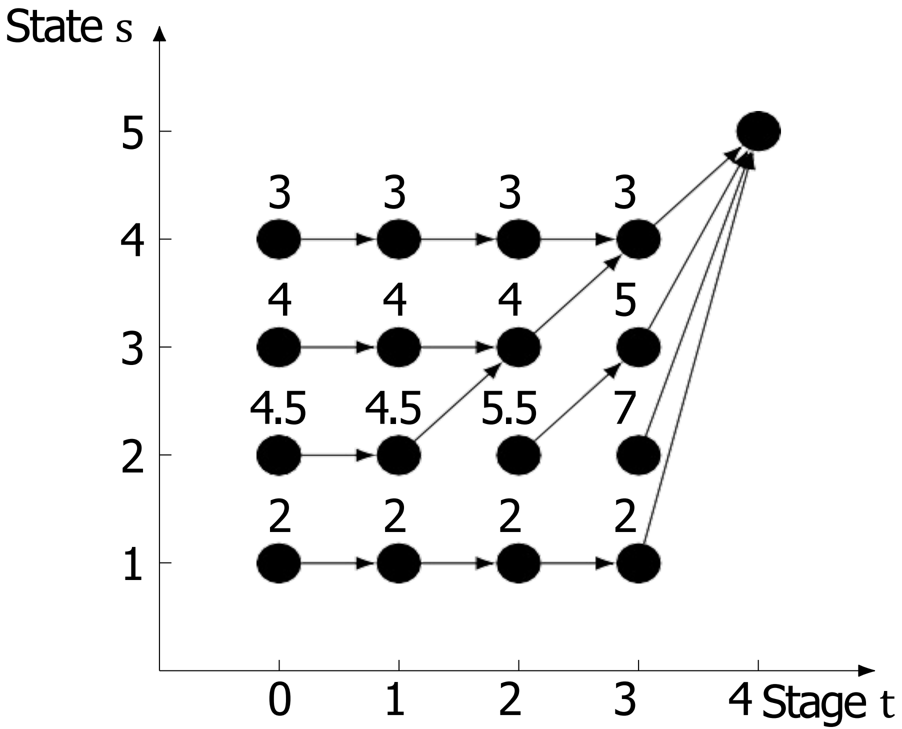
Destination is node 5.



Solving Shortest Path



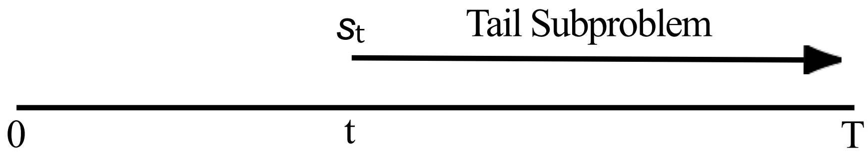
Destination is node 5.



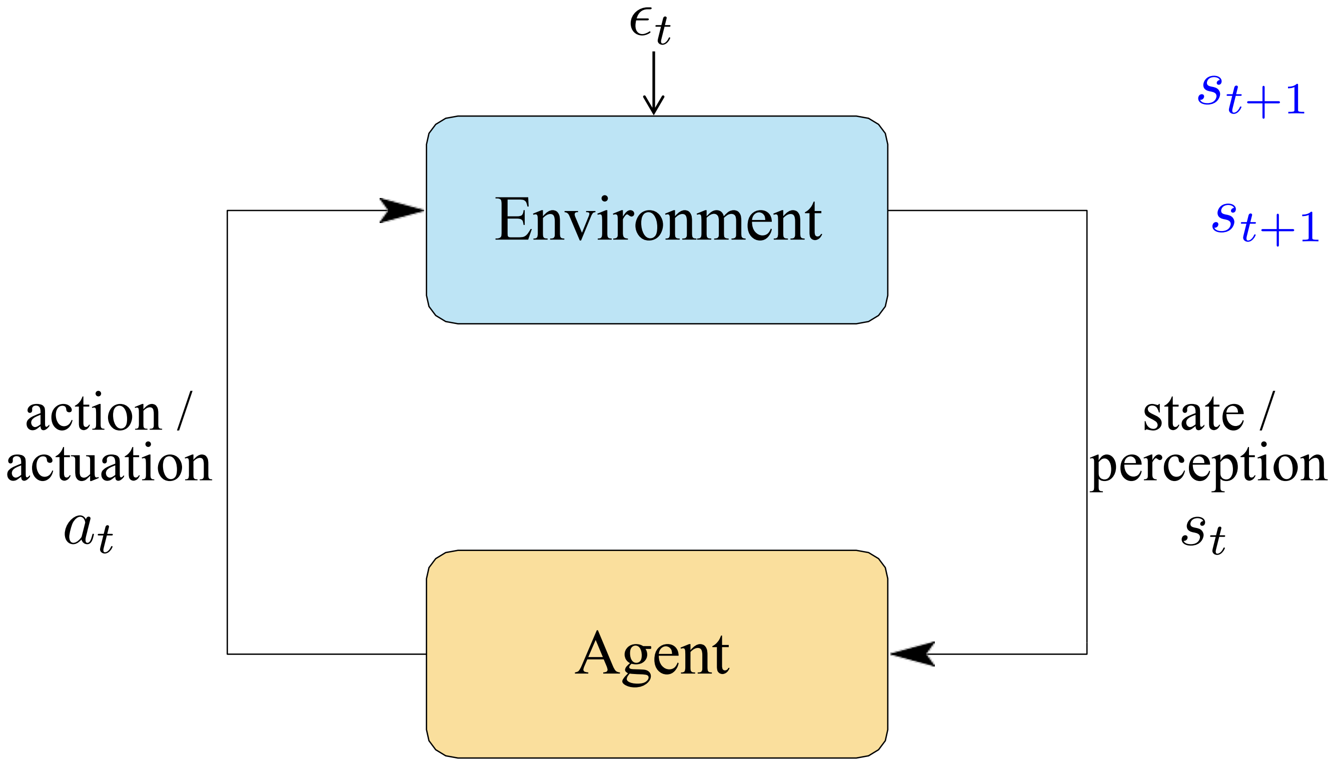
Outline

1. Reinforcement learning to solve sequential decision problems
2. Formulation of finite-horizon decision problems
3. **Solving finite-horizon decision problems**
   a. Example: shortest path routing
   b. **Dynamic programming algorithm**
   c. Sequential decision making as shortest path
   d. Forward DP
4. Course overview
   a. Administrivia

Principle of optimality (Bellman, 1957)



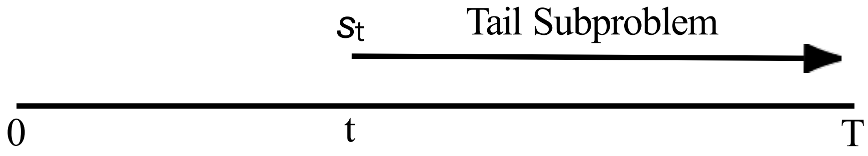
The Agent-Environment Interaction Protocol



$$s_{t+1} = f(s_t, a_t, \epsilon_t)$$

$$s_{t+1} \sim P(\cdot | s_t, a_t)$$

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,

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

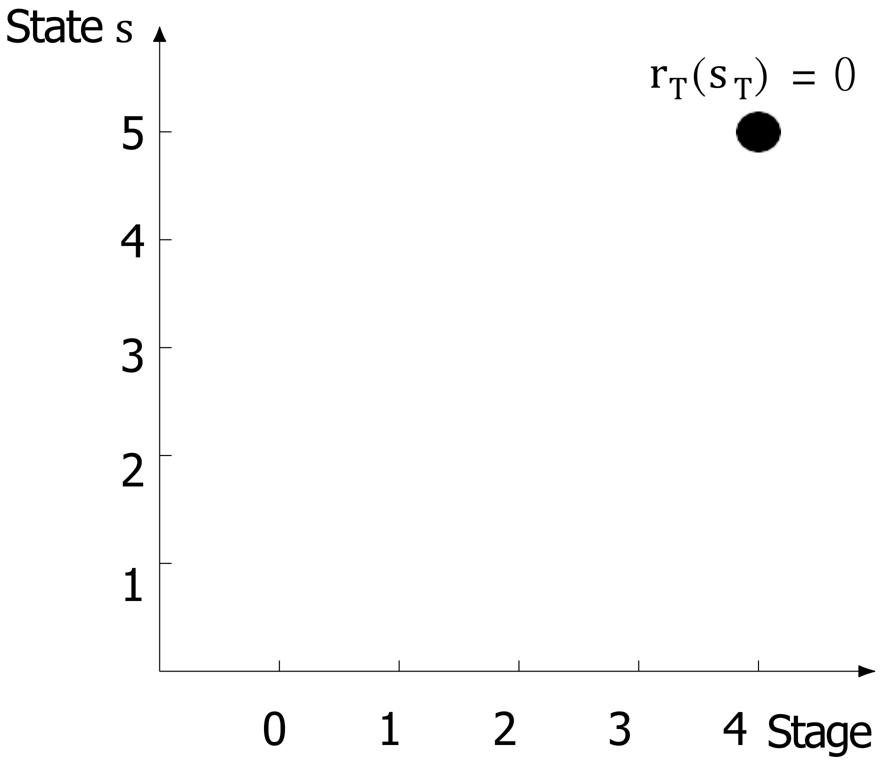
over $\{a_t, \dots, a_{T-1}\}$ with $a_\tau \in A_\tau(s_\tau)$, $\tau = t, \dots, T-1$. Then, the truncated optimal action sequence $\{a_t^*, \dots, a_{T-1}^*\}$ is optimal for this subproblem.

Dynamic programming algorithm

$$\boxed{V_T(s_T) = r_T(s_T)}$$

Dynamic programming algorithm

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

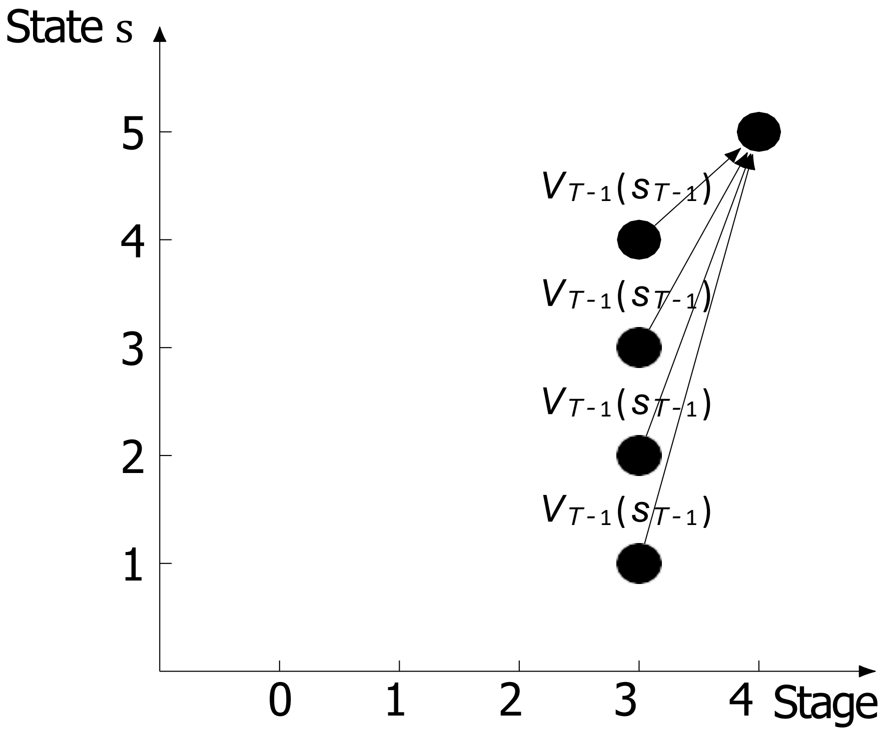


Dynamic programming algorithm

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

$$V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{\epsilon_t}\left[r_t(s_t, a_t) + V_{t+1}(s_{t+1})\right]$$

end for

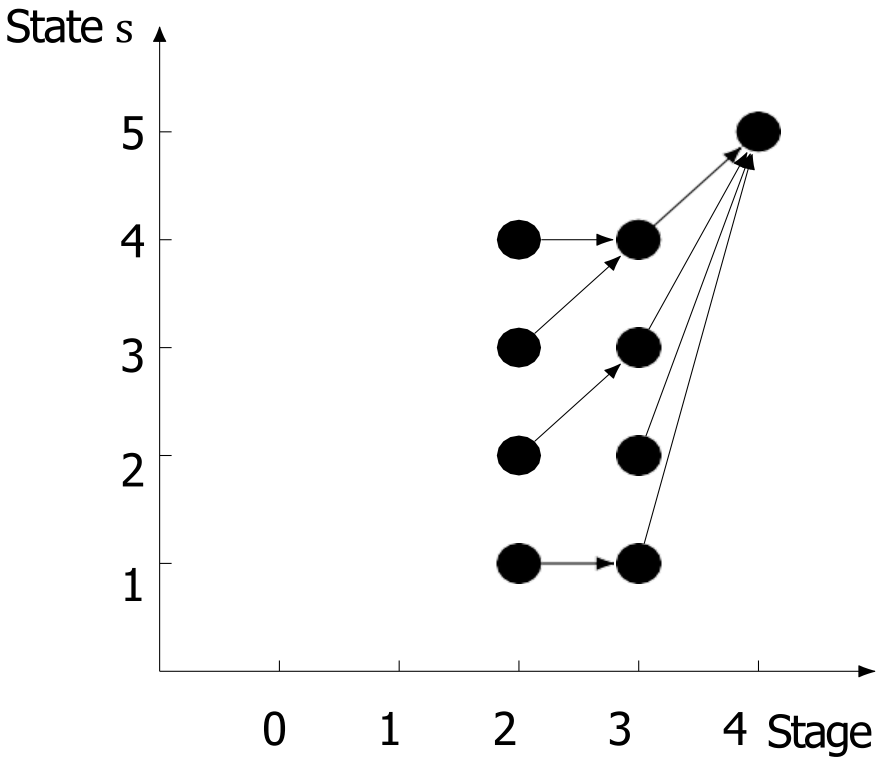


Dynamic programming algorithm

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

$V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{\epsilon_t} \left[r_t(s_t, a_t) + V_{t+1}(s_{t+1}) \right]$

end for

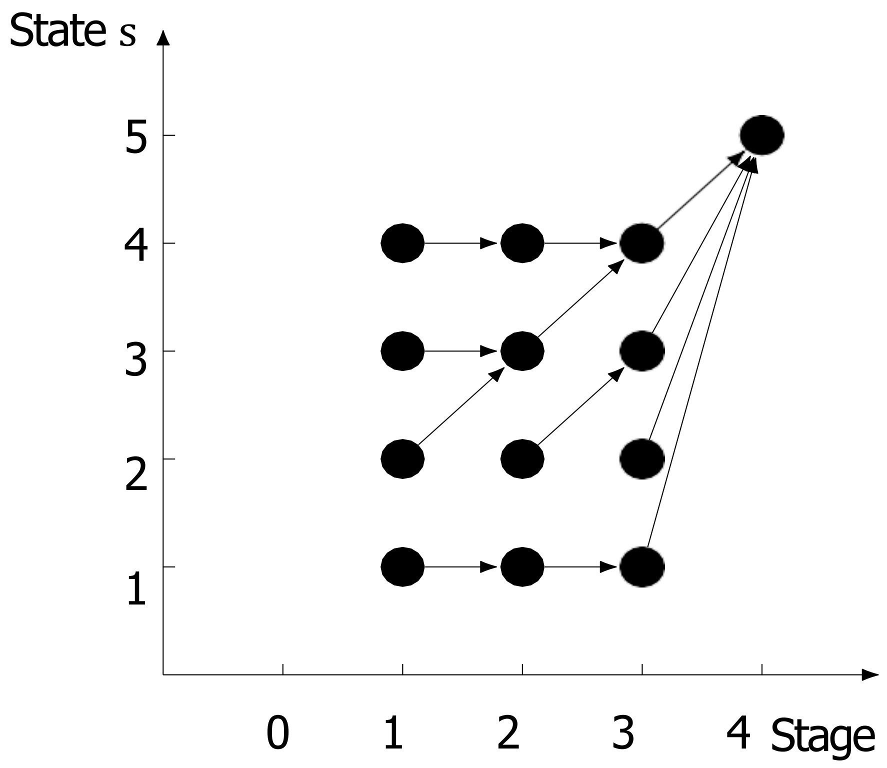


Dynamic programming algorithm

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

$$V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{\epsilon_t} \left[r_t(s_t, a_t) + V_{t+1}(s_{t+1}) \right]$$

end for

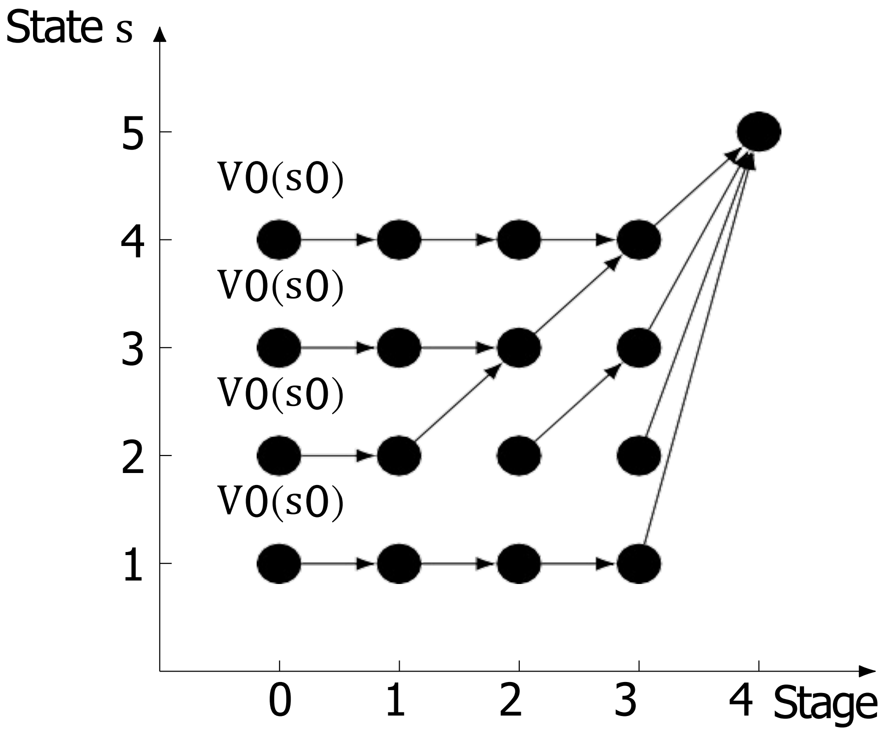


Dynamic programming algorithm

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

$V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E}_{\epsilon_t}\left[r_t(s_t, a_t) + V_{t+1}(s_{t+1})\right]$

end for



Note (simplification): we drop γ.

Dynamic programming algorithm

$V_T(s_T) = r_T(s_T)$
for $t = T-1, \dots, 0$ **do**
$\quad V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \mathbb{E} \, [r_t(s_t, a_t) + V_{t+1}(s_{t+1})]$
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 $\pi^* = (\pi_0^*, \dots, \pi_{T-1}^*)$ is optimal.

Dynamic programming algorithm

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

- **Proof**: by induction
- Equivalent to Bellman-Ford algorithm
- **Strength**: Generality
- **Weakness**: Computationally expensive O(|S||A|T)
- Much better than naive approach O(T!)
- ALL the tail subproblems are solved (in addition to the original problem)

Consider: Do other shortest path algorithms have sequential decision interpretations? Dijkstra’s, A*, Floyd–Warshall, Johnson’s, Viterbi, etc.

Proof of the induction step

Assume w.l.o.g. that $\gamma = 1$. Let $f_t: S \times A \to S$ denote the transition *function*.

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

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

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

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

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

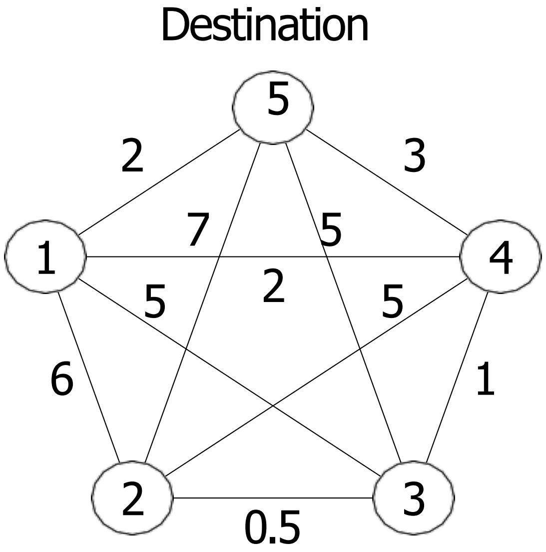
$$= \max_{\pi_t} \mathbb{E}\left\{r_t\big(s_t, \pi_t(s_t)\big) + V_{t+1}\left(f_t\big(s_t, \pi_t(s_t)\big)\right)\right\}$$

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

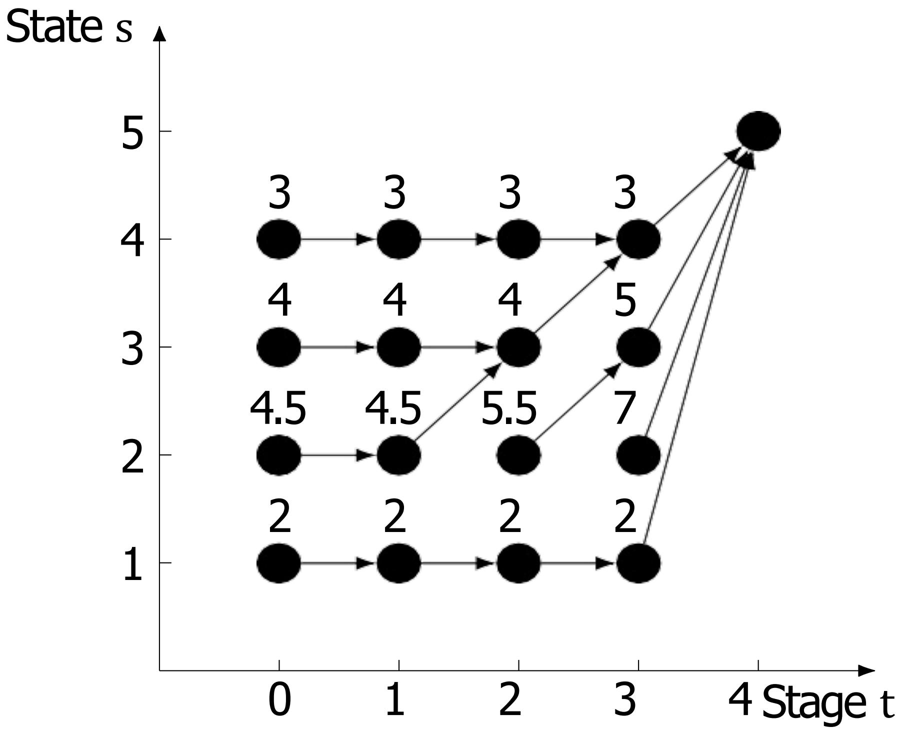
$$= V_t(s_t)$$

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

Solving Shortest Path



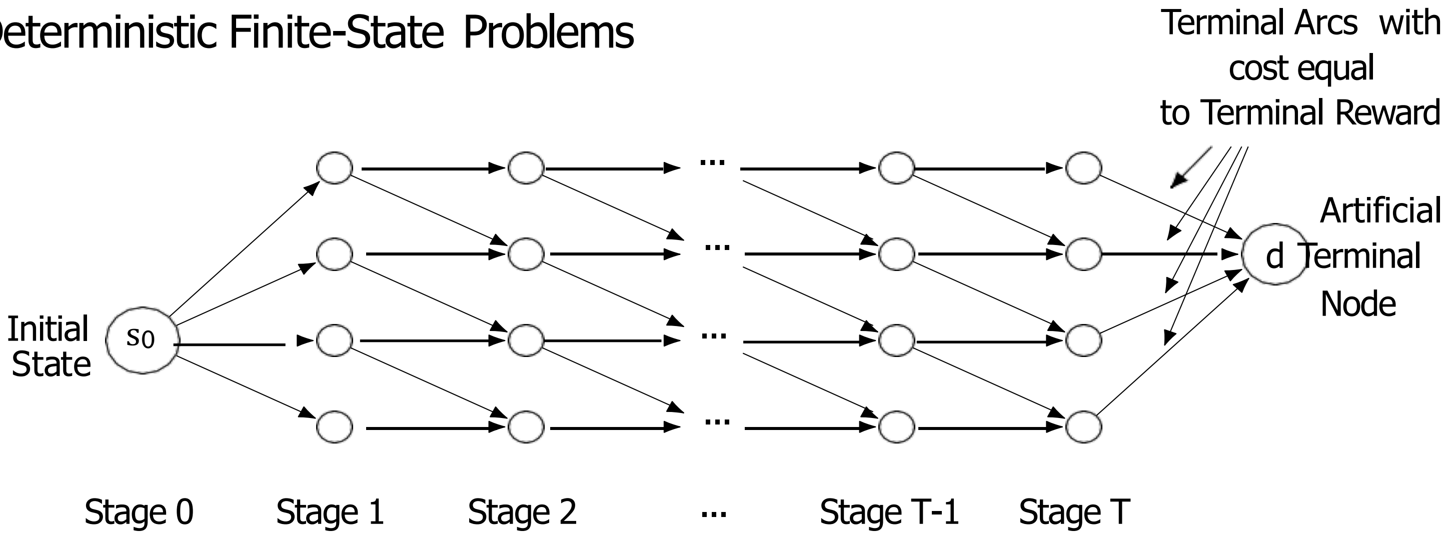
Destination is node 5.



Outline

Sequential decision making as shortest path

For Deterministic Finite-State Problems



Example: Thermostats (linear-quadratic control)

64F
(18C)

64F
65F

... ...

74F

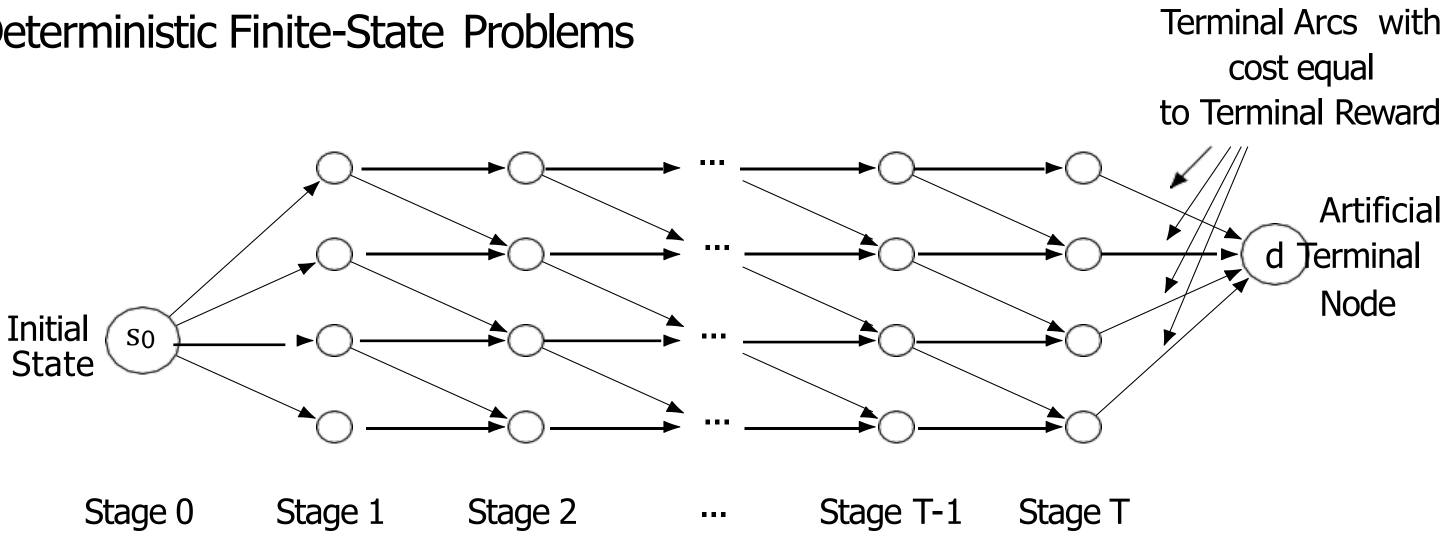
Applications:
control systems,
industrial manufacturing

Too cold! 🥶 ...

Great temperature ☺️

Sequential decision making as shortest path

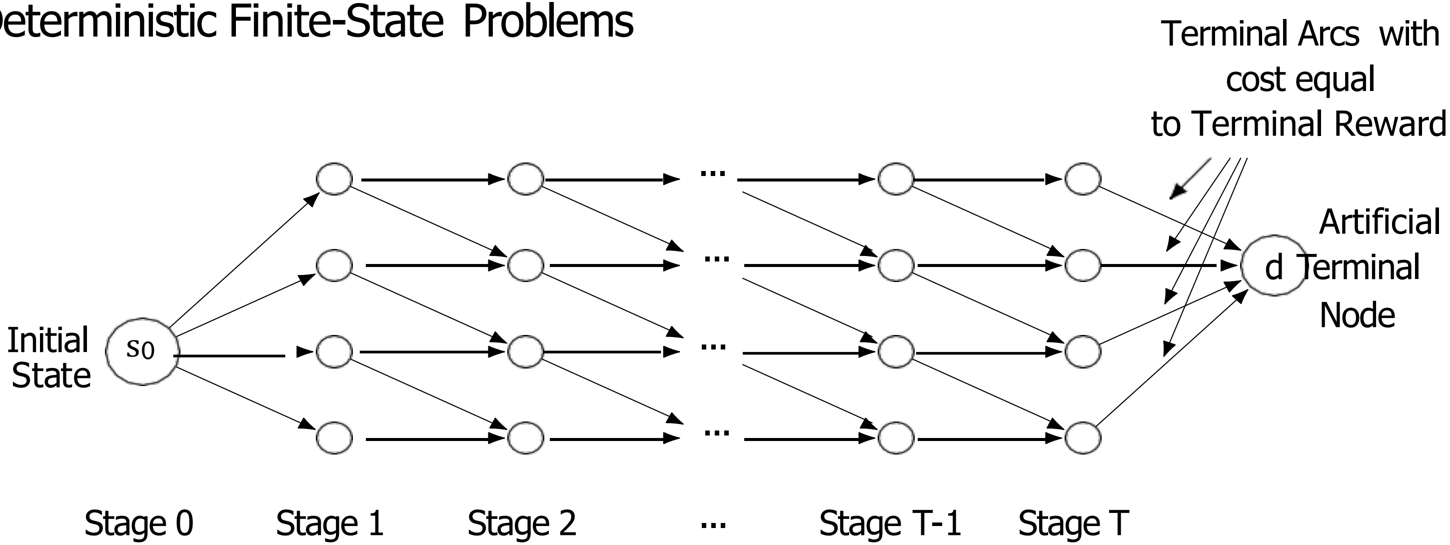
For Deterministic Finite-State Problems



Example: Breakout

Sequential decision making as shortest path

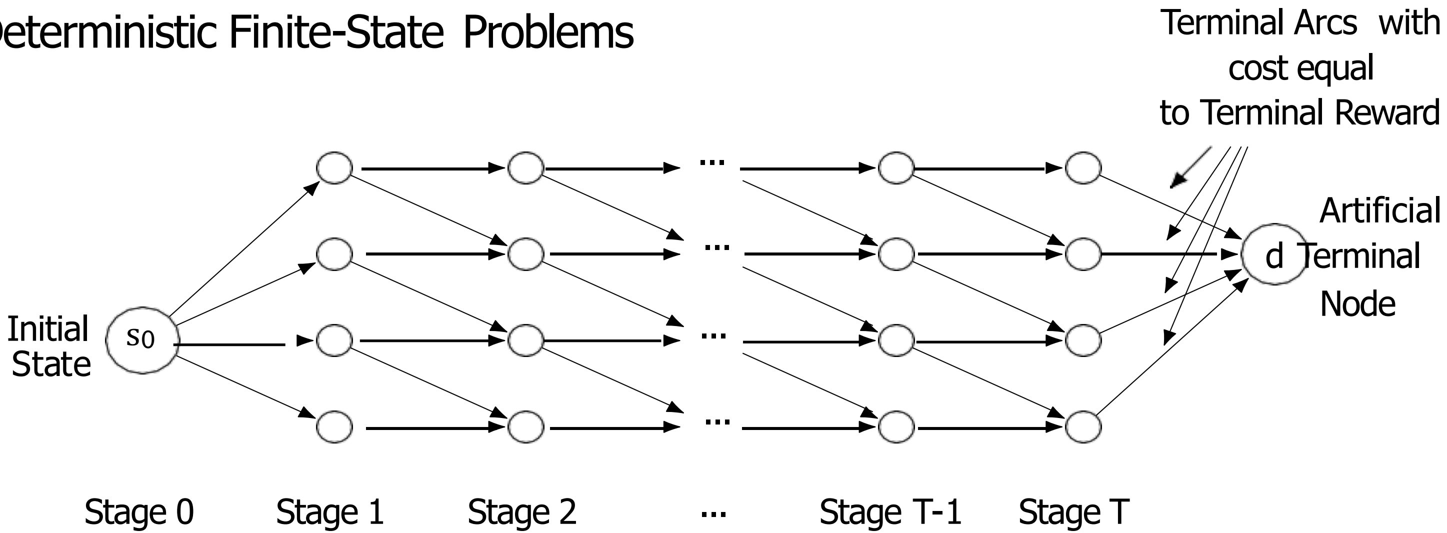
For Deterministic Finite-State Problems



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

Sequential decision making as shortest path

For Deterministic Finite-State Problems



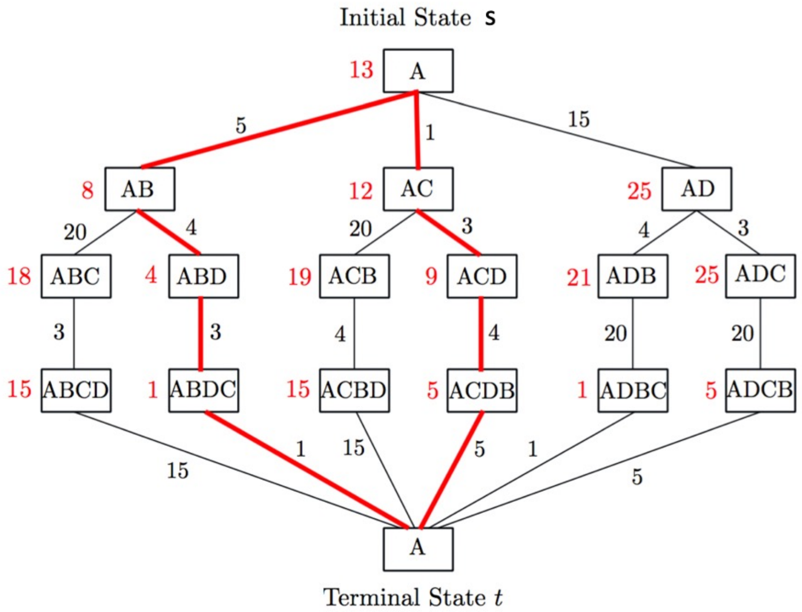
Example: Integer programming (combinatorial optimization)

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

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**.



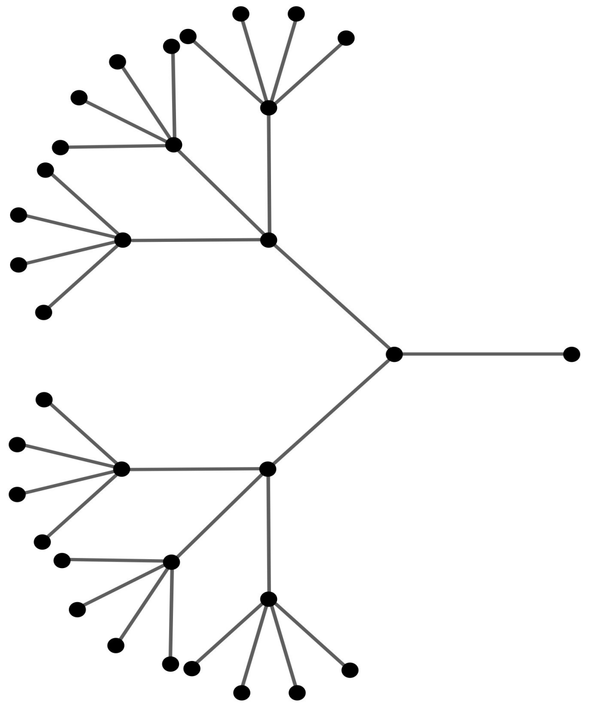
Matrix of Intercity Travel Costs

	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

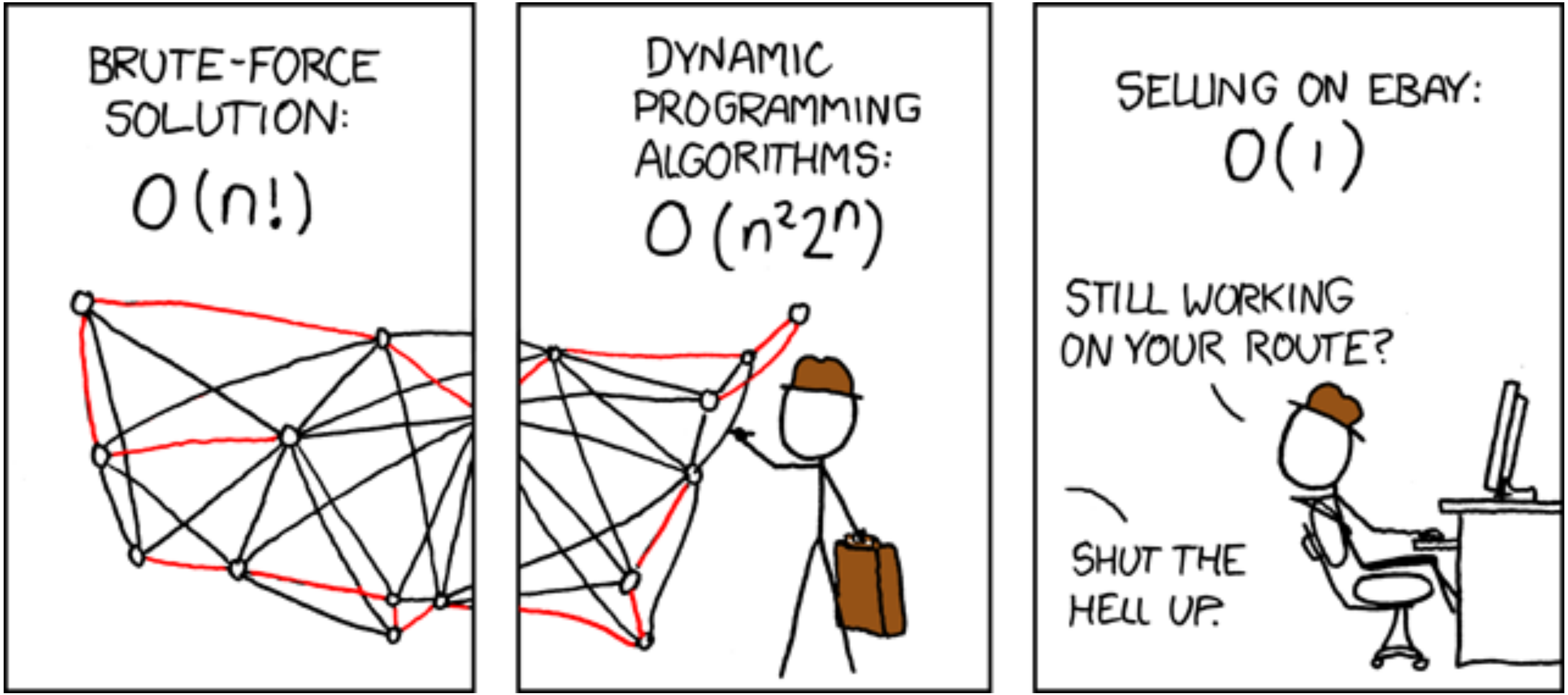
- Arcade Learning Environment (ALE): framework that allows researchers and hobbyists to develop AI agents for Atari 2600 games
- ALE parameters
  - 60 frames per sec
- Suppose a game is 2 minutes long
- Horizon is 2 * 60 * 60 = 7200 steps long
- Given 3 actions, the decision space is $3^{7200} \approx 10^{3435}$

$a_t = \text{left}$

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.



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

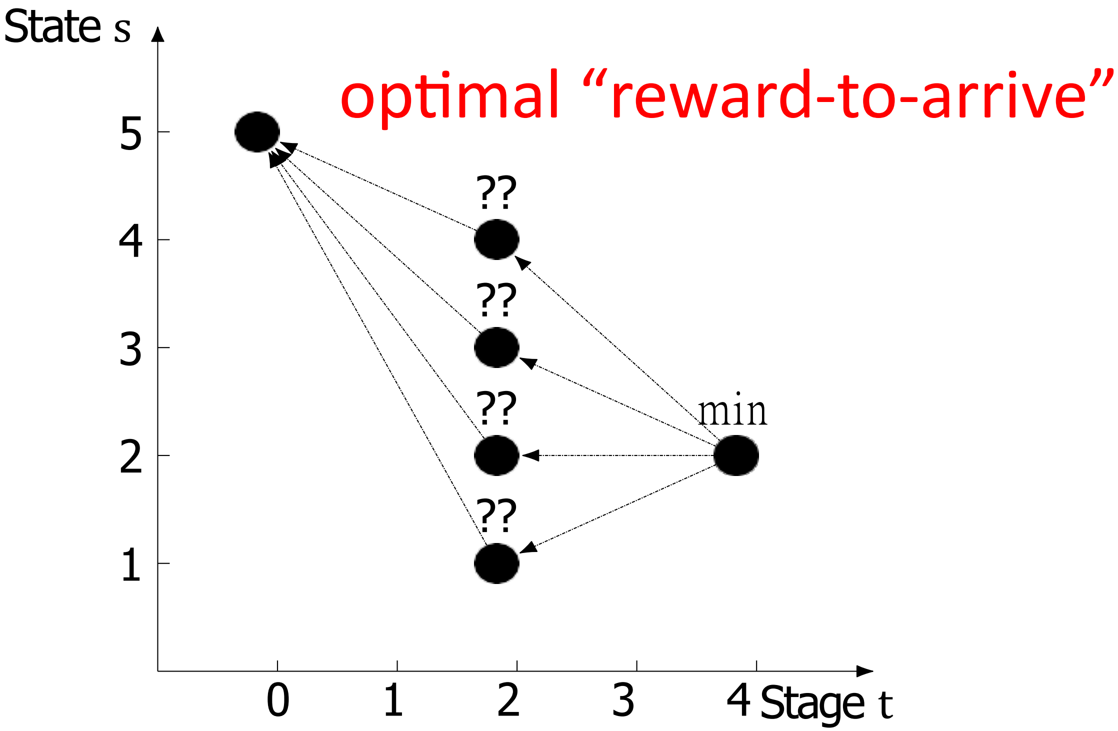
Outline

1. Reinforcement learning to solve sequential decision problems
2. Formulation of finite-horizon decision problems
3. **Solving finite-horizon decision problems**
    a. Example: shortest path routing
    b. Dynamic programming algorithm
    c. Sequential decision making as shortest path
    d. **Forward DP**
4. Course overview
    a. Administrivia

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 $t = 1, \ldots, T$ **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 $t = T - 1, \dots, 0$ **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

Outline

1. Reinforcement learning to solve sequential decision problems
2. Formulation of finite-horizon decision problems
3. Solving finite-horizon decision problems
   a. Example: shortest path routing
   b. Dynamic programming algorithm
   c. Sequential decision making as shortest path
   d. Forward DP
4. **Course overview**
   a. Administrivia

Philosophy + aims of the course

- What is an appropriate **foundational** course to **advance research and practice** in sequential decision making?
- Context

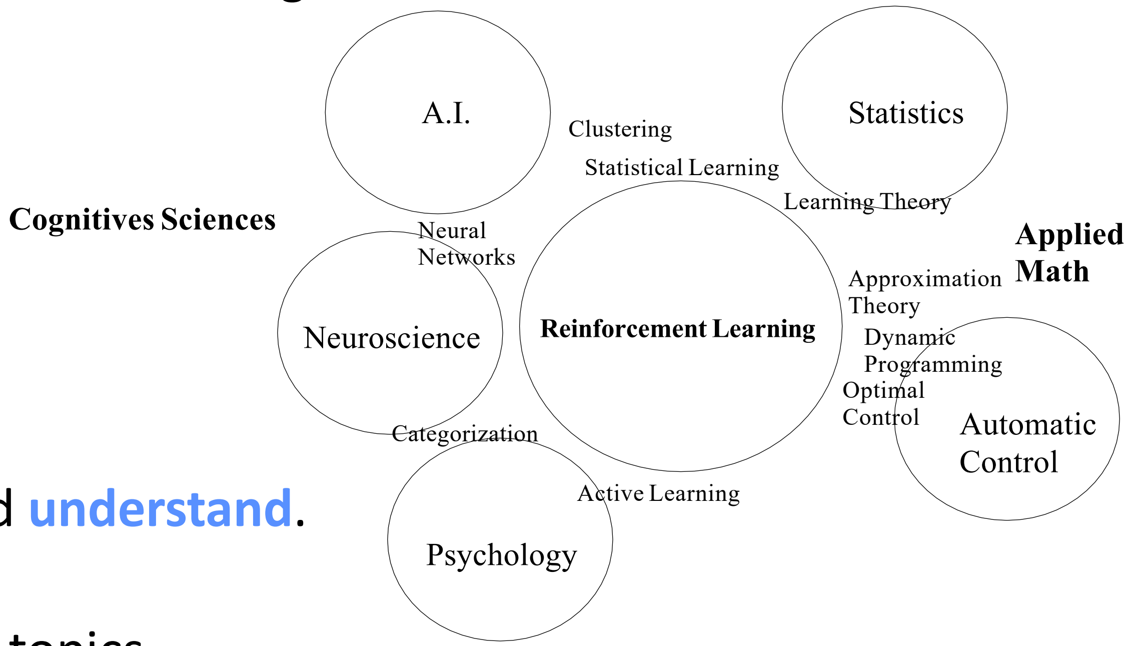


Figure: Note: circles may not be to scale.
Credit: Alessandro Lazaric

Design

- (2/3 **Exploit**)
  Teach what we **know** and **understand**.
- (1/3 **Explore**)
  Selected **up-and-coming** topics.

What: the Highlights of the Course

How to *model* DP & RL problems

- *What:* problem space, deterministic vs Markov decision process, imperfect information
- *Tools:* probability, processes, Markov chain

What: the Highlights of the Course

How to *model* DP & RL problems

How to solve *exactly* DP & RL problems

- *What:* Bellman equations, dynamic programming algorithms
- *Tools:* induction, optimality principle, fixed point operators

What: the Highlights of the Course

How to *model* DP & RL problems

How to solve *exactly* DP & RL problems

How to solve *incrementally* DP & RL problems

- *What:* Monte Carlo, temporal difference (TD), Q-learning
- *Tools:* stochastic approximation, max norm contraction analysis

What: the Highlights of the Course

How to *model* DP & RL problems

How to solve *exactly* DP & RL problems

How to solve *incrementally* DP & RL problems

How to solve *approximately* DP & RL problems

- *What:* approximate RL (TD-based methods, policy space methods, deep RL)
- *Tools:* function approximation, Lyapunov function analysis, deep learning, variance reduction

What: the Highlights of the Course

How to *model* DP & RL problems

How to solve *exactly* DP & RL problems

How to solve *incrementally* DP & RL problems How to

solve *approximately* DP & RL problems

With examples from *resource optimization*, *control systems*, *computer games*, *and beyond*.

Special topics (tentative)

- Empirical rigor in RL
- Scale & diversity of problems
  - Offline RL
  - Learning for Combinatorial Optimization
  - Multi-agent RL
  - Bayesian RL
  - Generalization in RL
- Applications
  - Case studies
  - Healthcare
  - Robotics
- Recent theoretical results

Outline

How: Textbooks and readings

Useful references (recommended but not required)

(a) **Dynamic Programming and Optimal Control** (2007), Vol. I, 4th Edition, ISBN-13: 978-1-886529-43-4 by Dimitri P. Bertsekas. [**DPOC**]

(b) The second volume of the text is a useful and comprehensive reference. [**DPOC2**]

(c) **Neuro Dynamic Programming** (1996) by Dimitri P. Bertsekas and John N. Tsitsiklis. [**NDP**]

Readings: We will give pointers to these references. Some additional readings / notes may be posted.

A note on notation. We will be using contemporary notation (e.g. s, a, V), which differs from notation from these texts (e.g. x, u, J). We will be maximizing instead of minimizing, etc.

How: Pre-requisites

(a) Solid knowledge of undergraduate probability (6.041A & 6.041B)

(b) Mathematical maturity and the ability to write down precise and rigorous arguments

(c) Python programming

We will issue a HW0 (not graded) to help you gauge your level of familiarity with the pre-requisite material and useful concepts (hints for HW).

When/What/Where

- Lecture: TR 4-4:30pm (4-237)
- Instructor
  - Cathy Wu <cathywu@mit.edu>
  - Office Hours: TR 4-4:30pm (4-237, TBD)
- Teaching assistant
  - Guilherme Venturelli Cavalheiro <guivenca@mit.edu>
  - Office hours: TBD (check website)
- Recitations: TBD (check website)
  - First recitation: 1pm tomorrow
- Staff list: <6-7950-staff@mit.edu>
  - Please include "[6.7950]" in your email subject line

Course pointers

- **web.mit.edu/6.7950/www**
- Website: lecture materials & general info
- Piazza: announcements, collab, HW, solutions, readings
- Gradescope: submit HW
- Psetpartners: find pset partners

Grading

- 7 homework assignments (30%)
  - More at the beginning, sparser later
- 1 in-class quiz (25%)
  - Coverage: first 14 lectures
- Class project (35%)
  - Research-level project of your choice.
  - Form groups of 1-3 students, you're welcome to start early!
  - Class presentation + final report
- Class participation (10%)
  - Participation during lecture; answering questions on Piazza; attending office hours and recitation

Homeworks

- 4 late days across all homeworks. Solutions for homework will be released shortly after the deadline (late submitters must abide by honor code).