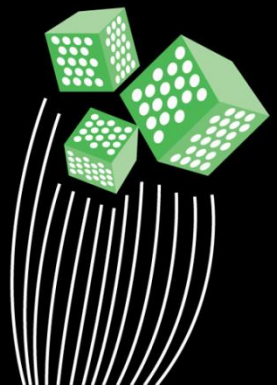
UNIVERSITY OF TWENTE.

REINFORCEMENT LEARNING FOR OPERATIONS MANAGEMENT
VALUE-BASED REINFORCEMENT LEARNING

Martijn Mes
Universiteit Twente



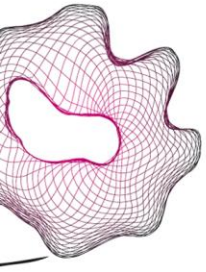
December 3, 2025

CONTENT

1. Positioning Reinforcement Learning
2. Introduction Reinforcement Learning
3. Markov Decision Processes
4. Approximate Dynamic Programming
5. Value Function Approximation
6. Preview of different forms of RL…

Running example: Trucks & Barges

Example to study in your own time: Patient Admission Planning



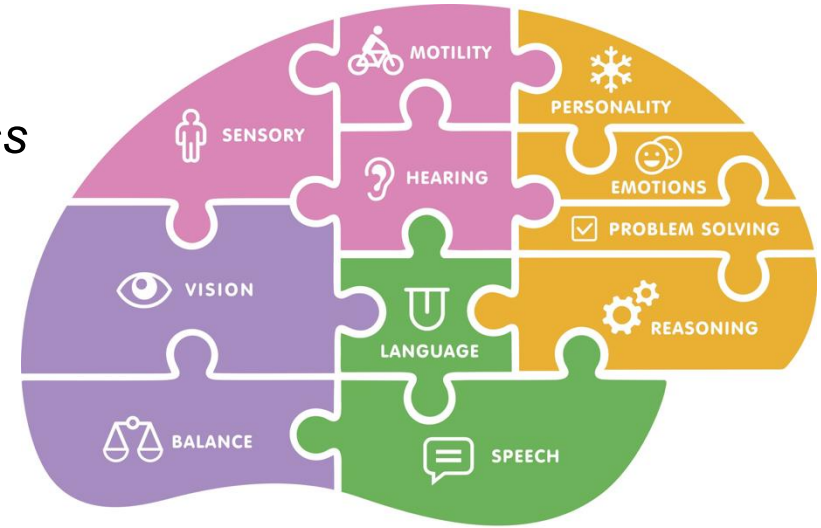
PART 1

Positioning Reinforcement Learning

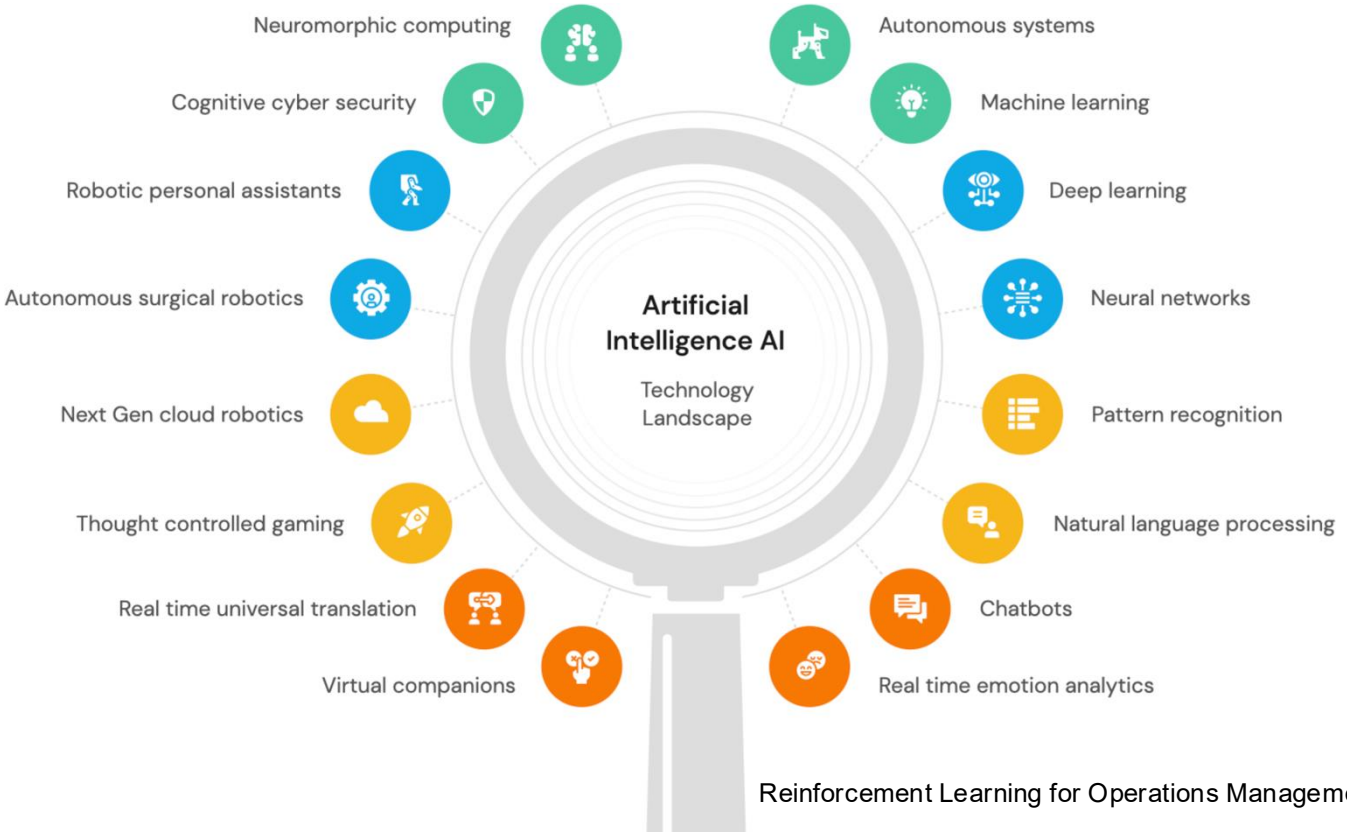
ARTIFICIAL INTELLIGENCE

- "*Artificial intelligence (AI) refers to systems that display intelligent behaviour by analysing their environment and taking actions – with some degree of autonomy – to achieve specific goals.*"
  European Commission

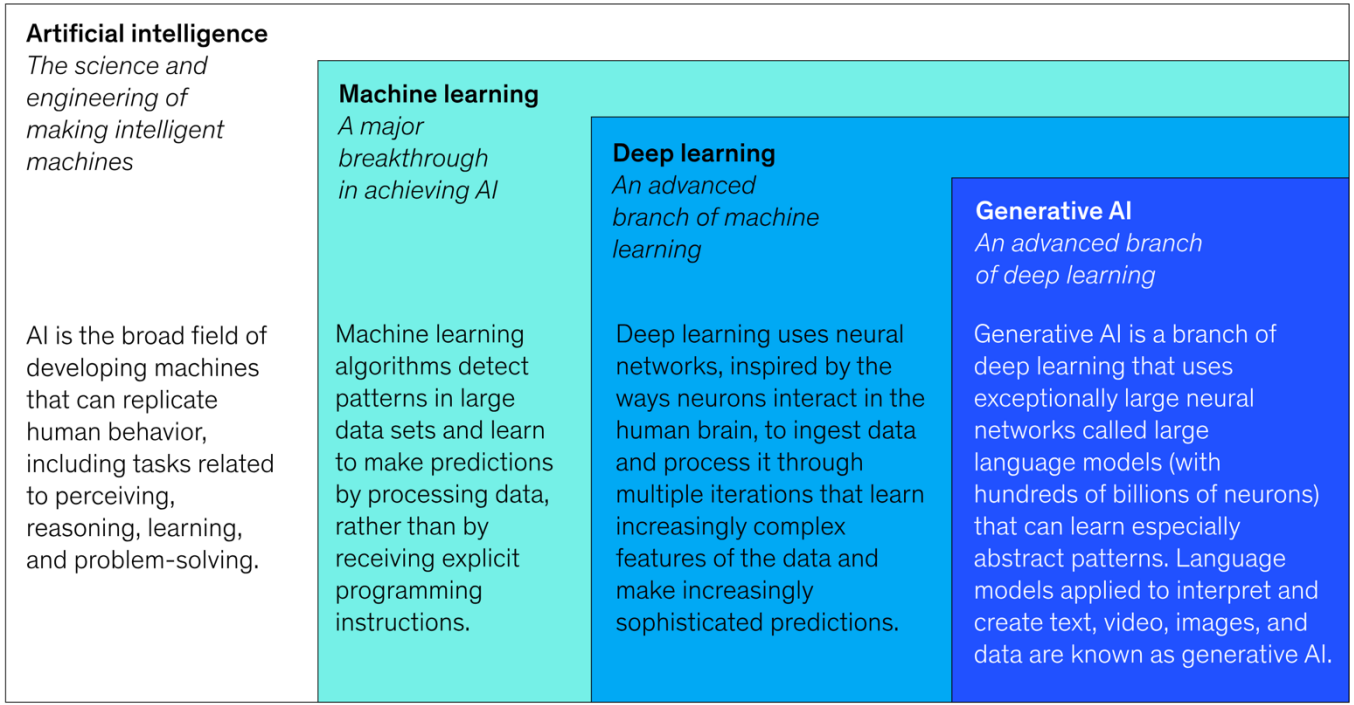
- "*AI […] refers to a broad class of technologies that allow a computer to perform tasks that normally require human cognition, including adaptive decision making.*"
  Tambe et al. (2019)



ARTIFICIAL INTELLIGENCE LANDSCAPE



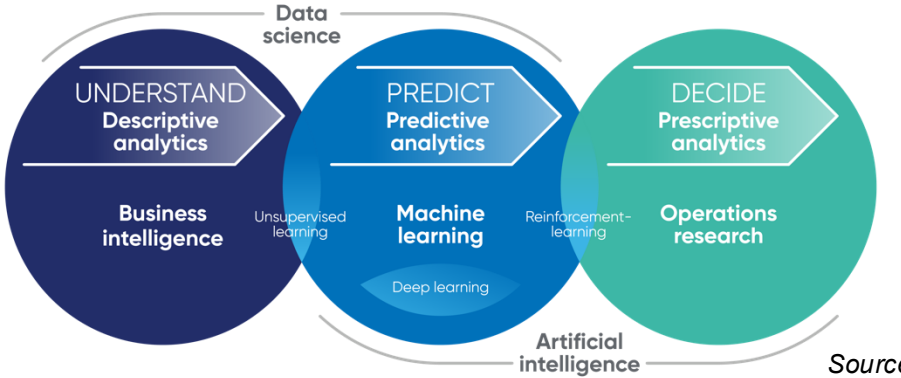
POSITIONING MACHINE LEARNING



Source: McKinsey & Company

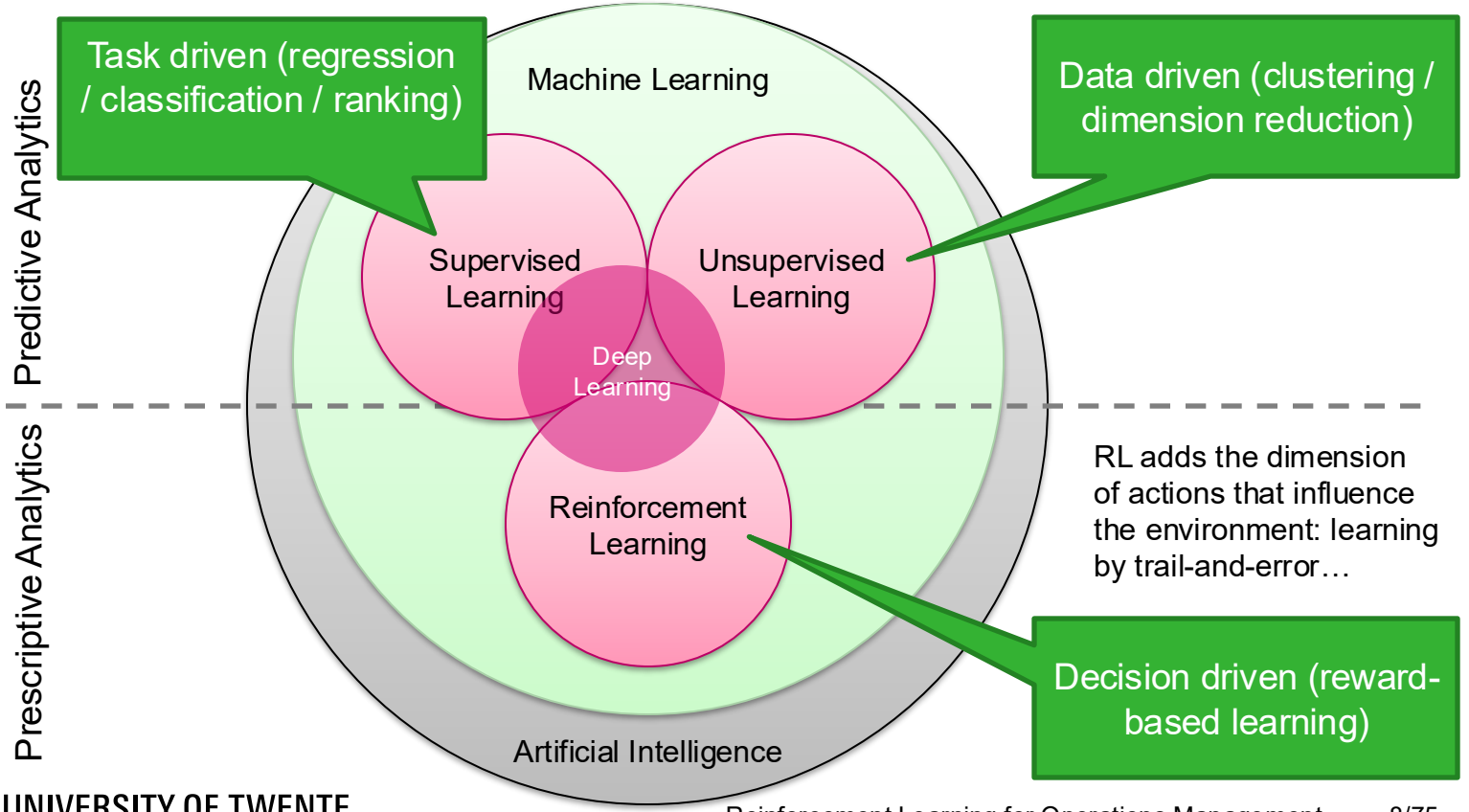
POSITIONING REINFORCEMENT LEARNING [1/2]

- RL is positioned at the interface between (Stochastic) Operations Research (OR) and Machine Learning (ML):
  - OR - "*A discipline that deals with the application of advanced analytical methods to help make better decisions*" [Wikipedia]
  - ML - "*Field of computer science that gives computers the ability to learn without being explicitly programmed*" [Wikipedia]
  - RL - Learn making sequential decision under uncertainty…



Source: Ivado

POSITIONING REINFORCEMENT LEARNING [2/2]



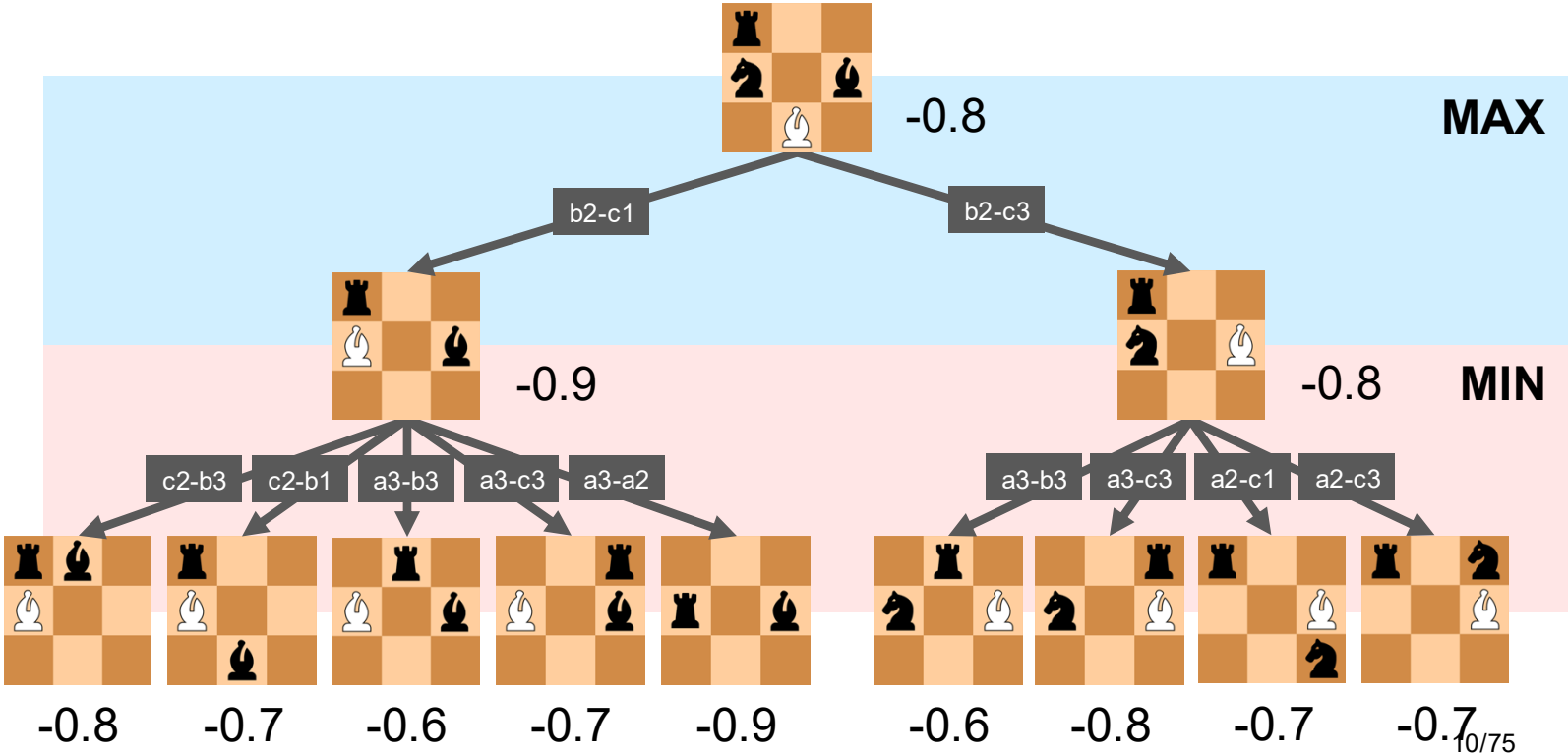
EXAMPLES OF RL FOR GAME AI

- Many nice examples of computers learning to play games, e.g., from Google Deepmind:
  - AlphaGo: https://youtu.be/8tq1C8spV_g (move 37 and 78)
  - AlphaZero: “general purpose” (Shogi, Chess, Go)
  - MuZero: also learning the rules of the game
  - AlphaStar: https://youtu.be/DpRPfidTjDA (StartCraft II)

- Let's start with a simple example: Chess

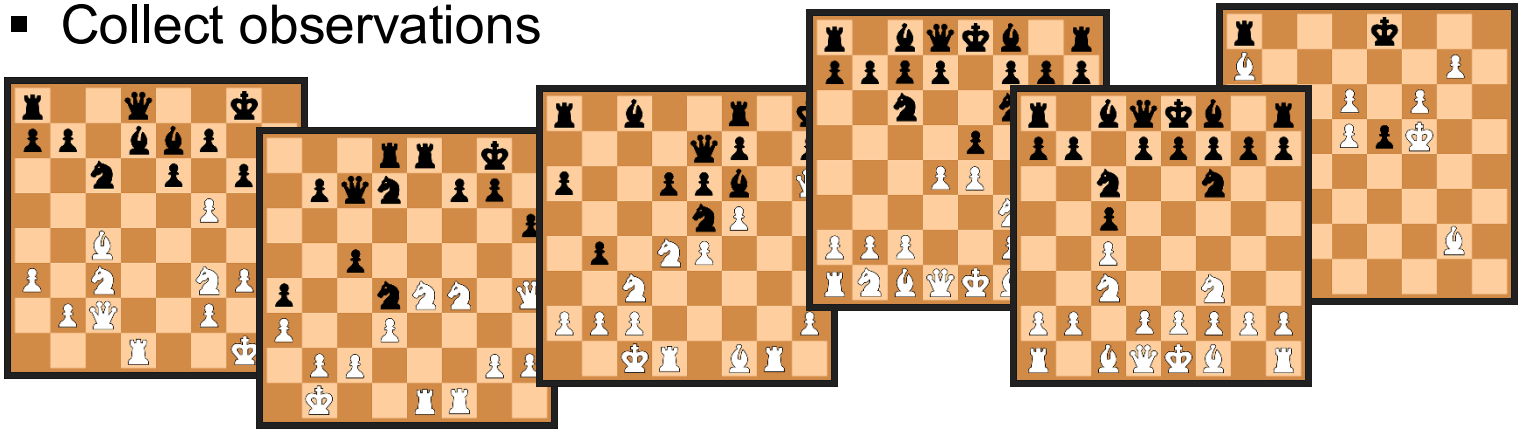
CHESS: OPTIMIZATION/ENUMERATION

- Game tree complexity: 10^{123} (versus 10^{360} for Go)

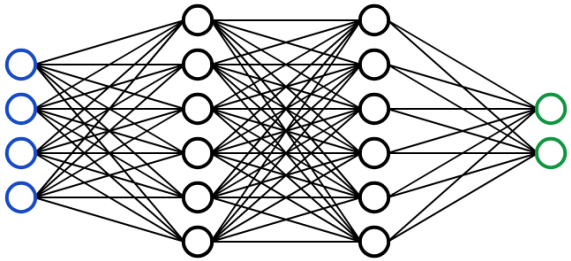


CHESS: SUPERVISED LEARNING [1/2]

- Collect observations



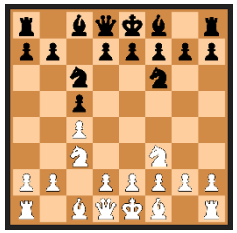
Input x:
Composition of pieces on the board derived from many game plays



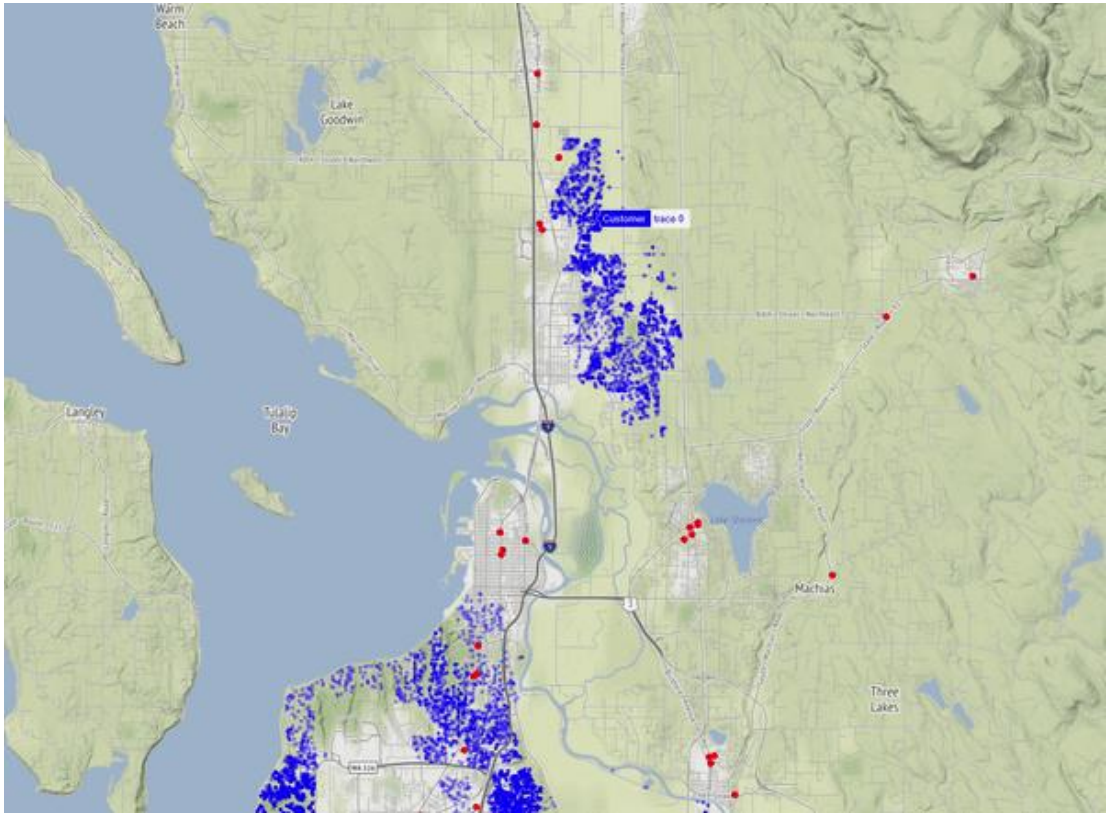
Output y:
Won or lost

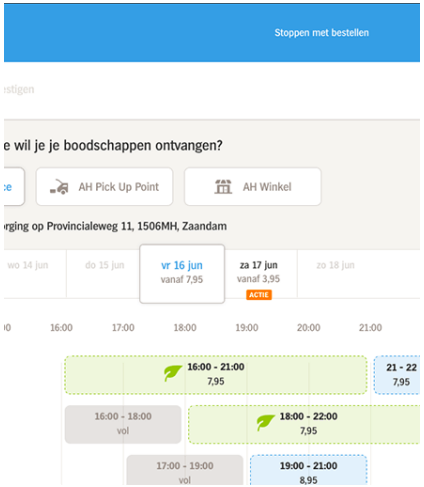
$$y = f(x)$$

CHESS: SUPERVISED LEARNING [2/2]

$$f\left(\text{  }\right) = [\text{d2}-\text{d3}]$$

REAL WORLD PROBLEMS: E-COMMERCE



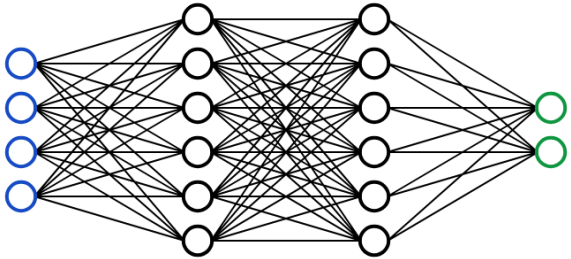


E-COMMERCE: SUPERVISED LEARNING [1/2]

Historical optimized routing plans:

Input x:

Sets of customers with their characteristics (location, size, time-window)



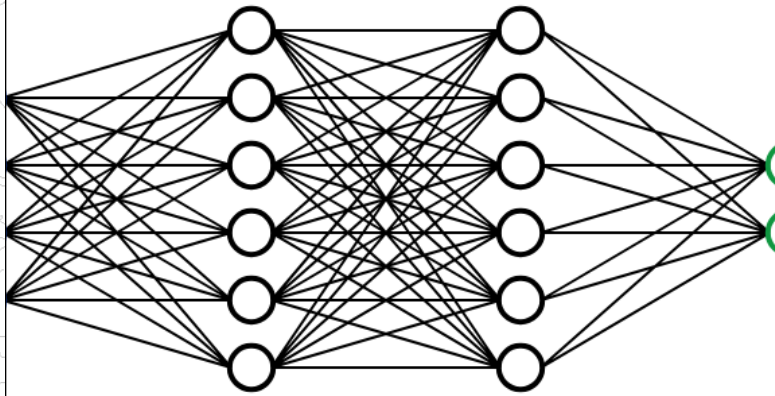
$y = f(x)$

Output x:

For each set, aspects like total distance, total costs, number of vehicles, costs per customer, lateness, etc.

E-COMMERCE: SUPERVISED LEARNING [2/2]

Input to trained neural network: set of specific customers

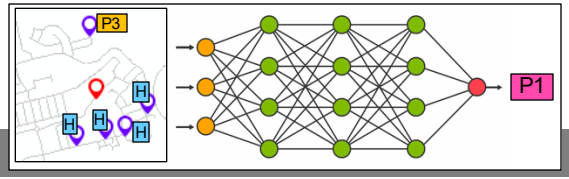


Expected in yet unknown final routing plan

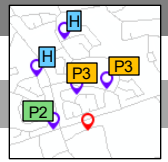
We can use this NN to decide whether to accept a customer, what time window to offer, or perhaps whether to nudge a customer to be delivered at a parcel locker. Possible weakness of this approach?

PREVIEW DEEP REINFORCEMENT LEARNING (DRL)

Initialize policy



Perform iterations

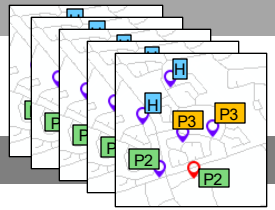


Collect batches of sampled states

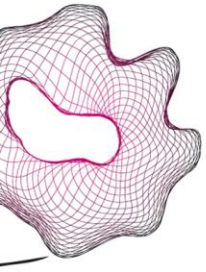
For all possible actions H P1 P2 P3

Perform simulations till the end of the day using the current policy

Determine best action P2



Update policy

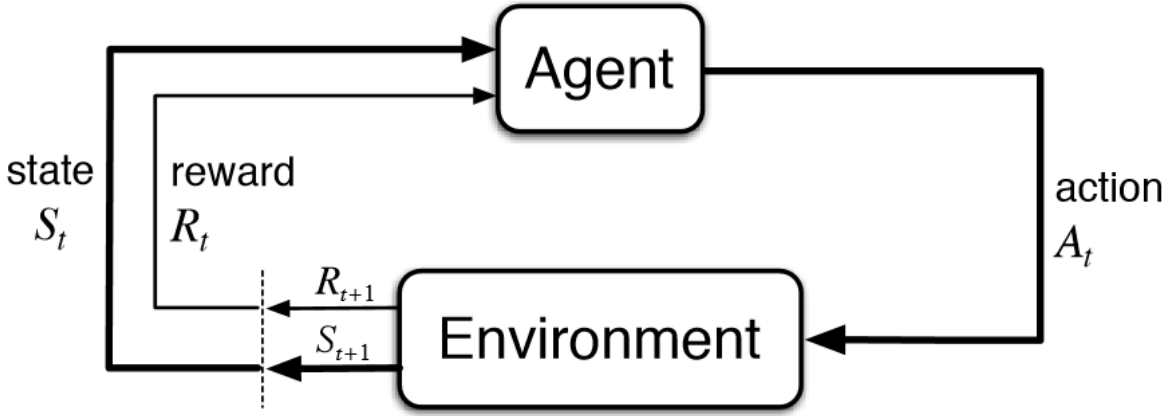


PART 2

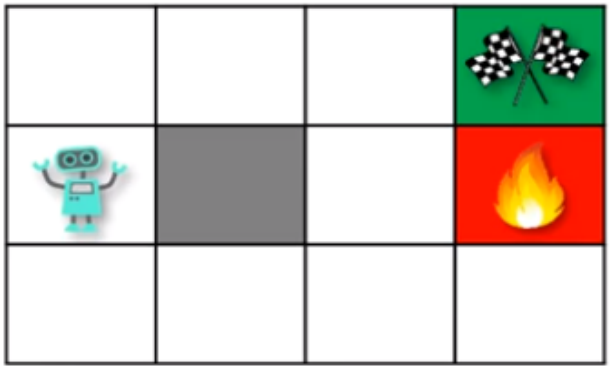
Introduction Reinforcement Learning

REINFORCEMENT LEARNING

With RL, we study the interaction with the environment:

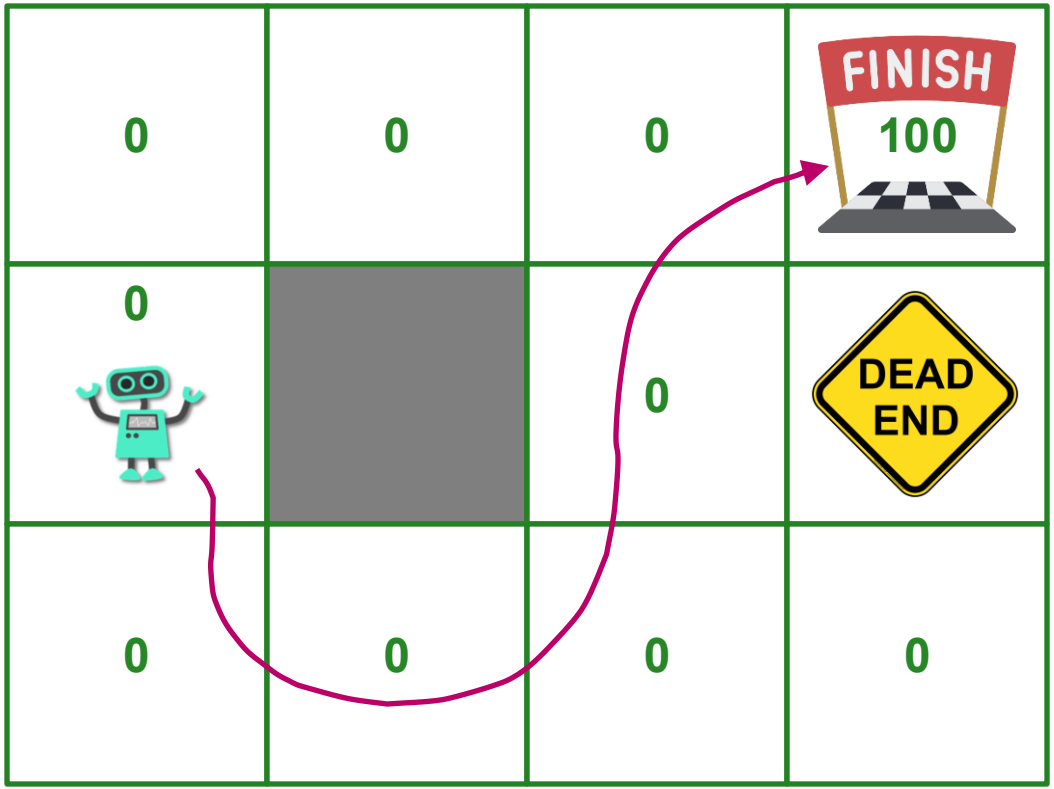


THE IDEA OF VALUE-BASED RL



- We have to learn the consequences of our actions (rewards + reachable states):
  - *V(S)* = (discounted) (in)finite future rewards from state *S* onwards given an optimal policy
    Q(S,a) = previous + direct reward of *a* in *S*
- Ways of learning:
  - Look-ahead one step, take action, update *V(S)/Q(S,a)* → *TD(0)*
  - Play out entire episode with a "given policy" (probably using Monte Carlo simulation) and propagate values to update the Q-values for observed *(S,a)* combinations or *V(S)* for observed *S* → *TD(1)*

LEARNING THE VALUE OF STATES

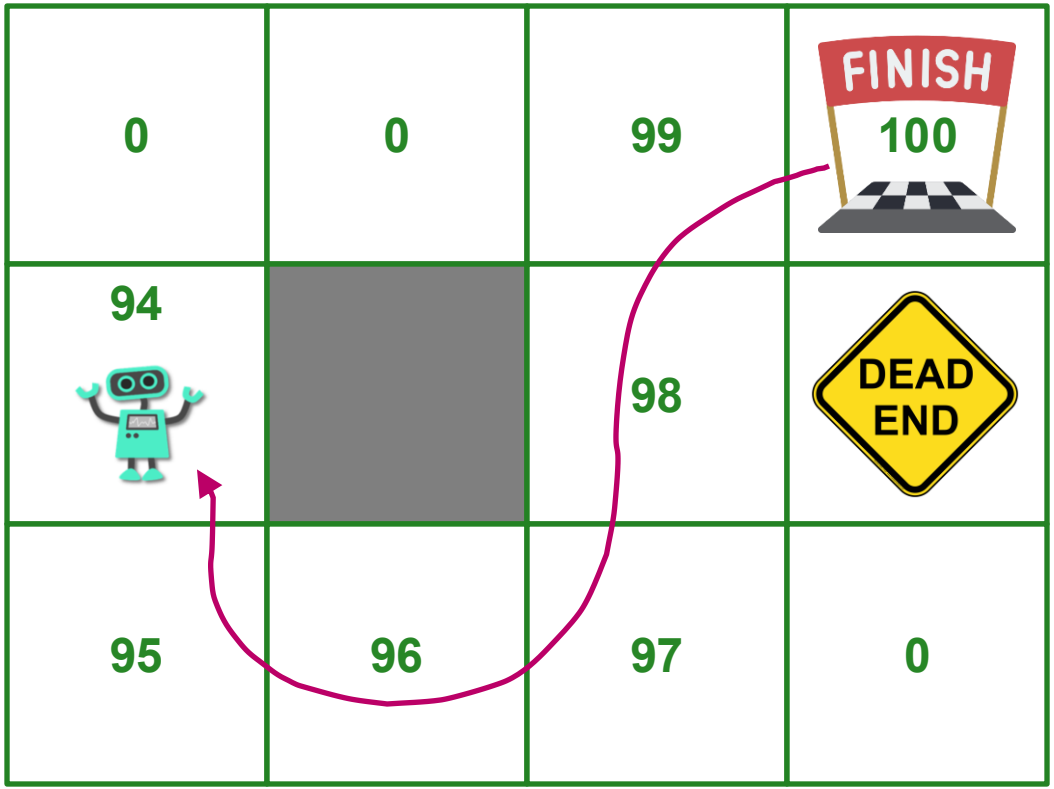


V(S) = value of *S*

Ways of learning:

- Look-ahead one step, take action, update *V(S)*
- Play out entire episode with a "given policy" and propagate values to update *V(S)* for all encountered *S*

LEARNING THE VALUE OF STATES

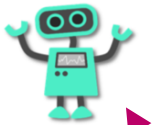


V(S) = value of *S*

Ways of learning:

- Look-ahead one step, take action, update *V(S)*
- Play out entire episode and propagate values to update *V(S)* for all encountered *S*

LEARNING THE VALUE OF STATE-ACTION PAIRS

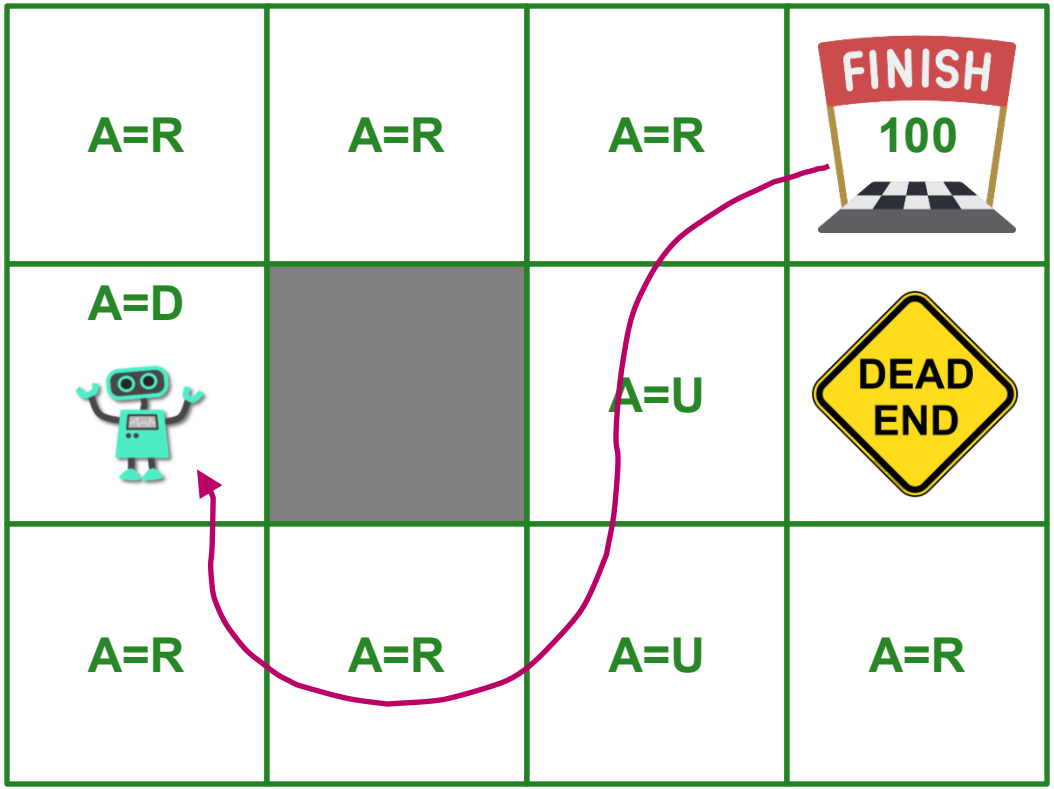
R=0 L=0 U=0 D=0	R=0 L=0 U=0 D=0	R=99 L=0 U=0 D=0	
U=0; D=94 		R=0 L=0 U=98 D=0	
R=95 L=0 U=0 D=0	R=96 L=0 U=0 D=0	R=0 L=0 U=97 D=0	R=0 L=0 U=0 D=0

Q(S,a) = value of action *a* in *S*

Ways of learning:

- Look-ahead one step, take action, update *Q(S,a)*
- Play out entire episode with a "given policy" and propagate values to update *Q(S,a)* for all encountered *(S,a)* combinations

LEARNING THE POLICY DIRECTLY



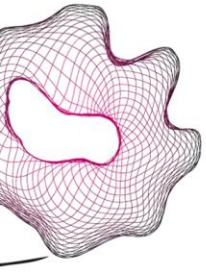
f(S) = *a*

Ways of learning:

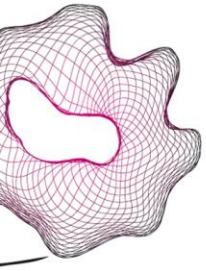
- Evaluate long term impact of all decisions in a state
- Update policy function approximation *f(S)*

LEARNING DIMENSIONS

- Model-free or model-based: do we have a model of the world, i.e., of the rewards and transition probabilities?
- Real-world (online) or simulator (offline): can we train offline in a simulator before implementing our decision/policy in the real world?
- Active or passive learning: do we simultaneously need to learn the value functions and the policy (active) or is the policy already given (passive)?
- On-policy or off-policy: do we learn the optimal policy independently on the agent's actions (off) or does the agent learn the value of the policy followed including the exploration steps (on)? The latter constrains our learning process, as we need an exploration strategy that is built into the policy itself.



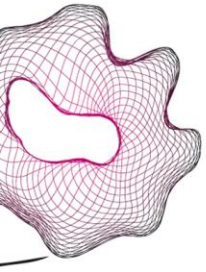
Markov Decision Processes (MDPs)



MODELING THE DYNAMICS

- MDPs are used to model and solve sequential decision problems under uncertainty
- Characterized by:
  - Decision epochs t (not necessarily time)
  - States $S_t \in \mathcal{S} = \{1,2, \ldots\}$
  - Decisions/actions $x_t \in \mathcal{X}_t(S_t)$: follow from a decision function X_t^{π}, where $\pi \in \Pi$ is a family of policies
  - Immediate contributions/rewards (payment or costs): $C(S_t, x_t)$
  - Exogenous information W_{t+1}: information arriving between t and $t+1$ (transition probabilities)

➢ Choosing decision x_t in the current state S_t with exogenous information W_{t+1}, results in a transition $S_{t+1} = S^M(S_t, x_t, W_{t+1})$ with contribution $C(S_t, x_t)$



OBJECTIVE

- Objective is to find the policy π that maximizes the expected sum of discounted contributions over all time periods (where T could be infinite):

$$\sup_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{T} \gamma^t C(S_t, X_t^\pi) \middle| S_0 \right\}$$

MDP FORMULATION TRUCKS & BARGES

- Decision epochs
- States
- Decisions
- Rewards
- Transition

Trucks And Barges

trucksandbarges.nl/leaderboard/kb66S2uFrsaCuzuxa

room name
DRL4OML2025

room number
MAR716

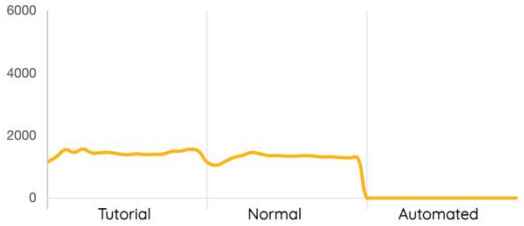
players
10

CLOSE ROOM

1 Asal

Best round
1281

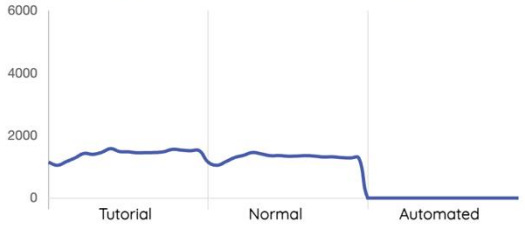
Average
1396



2 MirMohammad

Best round
1281

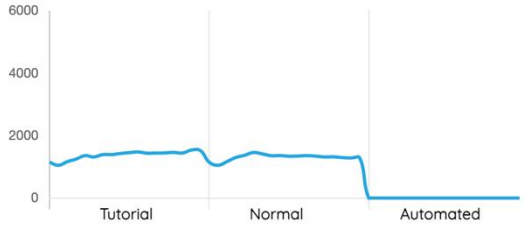
Average
1401



3 Latte

Best round
1281

Average
1403



Place	Player	Progress	Best round	Average
4th	Tijmen		1281	1448
5th	Tobias		1381	1493
6th	Mir		1281	1501
7th	Janne		1281	1533

Race for the lowest average score

1 Asal 1396

2 MirMohammad 1401

3 Latte 1403

MDP FORMULATION TRUCKS & BARGES [1/3]

- Decision epochs: days $t \in \mathcal{T} = \{0, \ldots, T\}$
- States: $S_t = [S_{t,d,r,k}]_{\forall [d,r,k] \in \mathcal{D} \times \mathcal{R} \times \mathcal{K}}$
- Decisions:

$$x_t = [x_{t,m,d,k}]_{\forall [m,d,k] \in \mathcal{M} \times \mathcal{D} \times \mathcal{K}},$$

s.t.

m = modality
d = destination
r = release day
k = due day

$$x_{t,m,d,k} \leq S_{t,d,0,k} \qquad \forall [m,d,k] \in \mathcal{M} \times \mathcal{D} \times \mathcal{K},$$

$$\sum_{m \in \mathcal{M}} x_{t,m,d,0} = S_{t,d,0,0} \qquad \forall d \in \mathcal{D},$$

$$\sum_{[d,k] \in \mathcal{D} \times \mathcal{K}} x_{t,m,d,k} \leq Q^m \qquad \forall m \in \mathcal{M},$$

$$x_{t,m,d,k} \in \mathbb{N} \qquad \forall [m,d,k] \in \mathcal{M} \times \mathcal{D} \times \mathcal{K}.$$

MDP FORMULATION TRUCKS & BARGES [2/3]

- Rewards:

$$C(S_t, x_t) = \sum_{m \in \mathcal{M}} \left(\sum_{\mathcal{D}' \subseteq \mathcal{D}} I_{m,\mathcal{D}'} \cdot c^f_{m,\mathcal{D}'} \right) + \sum_{[m,d,k] \in \mathcal{M} \times \mathcal{D} \times \mathcal{K}} c^v_{m,d} \cdot x_{t,m,d,k} + I_t C^T(S_t, x_t),$$

s.t.

$$I_{m,\mathcal{D}'} = \begin{cases} 1 & \text{if } \left(\prod_{d \in \mathcal{D}'} \left(\sum_{k \in \mathcal{K}} x_{t,m,d,k} \right) > 0 \right) \wedge \left(\sum_{[d,k] \in \mathcal{D} \setminus \mathcal{D}' \times \mathcal{K}} x_{t,m,d,k} = 0 \right), \\ 0 & \text{otherwise} \end{cases}$$

$$I_t = \begin{cases} 1 & \text{if } t = T \\ 0 & \text{otherwise} \end{cases}.$$

- Exogenous information: $W_t = [\tilde{S}_{t,d,r,k}]_{\forall [d,r,k] \in \mathcal{D} \times \mathcal{R} \times \mathcal{K}}$

$$\mathbb{P}^{\Omega}(W_t = \omega_t) = \beta \mathbb{P}^F(f = F) \prod_{[d,r,k] \in \mathcal{D} \times \mathcal{R} \times \mathcal{K}} \left(\mathbb{P}^J([d,r,k] = [D,R,K]) \right)^{\tilde{S}_{d,r,k}},$$

s.t.

$$f = \sum_{[d,r,k] \in \mathcal{D} \times \mathcal{R} \times \mathcal{K}} \tilde{S}_{d,r,k},$$

$$\beta = \frac{s!}{\prod_{[d,r,k] \in \mathcal{D} \times \mathcal{R} \times \mathcal{K}} \tilde{S}_{d,r,k}!}.$$

MDP FORMULATION TRUCKS & BARGES [3/3]

- Transition: $S_t = S^M(S_{t-1}, x_{t-1}, W_t)$,

s.t.

$$S_{t,d,0,k} = S_{t-1,d,0,k+1} - \sum_{m \in \mathcal{M}} x_{t-1,m,d,k+1} + S_{t-1,d,1,k} + \tilde{S}_{t,d,0,k}$$

$$\forall [d,k] \in \mathcal{D} \times \mathcal{K} \setminus \{K^{max}\} \ ,$$

$$S_{t,d,0,K^{max}} = S_{t-1,d,1,K^{max}} + \tilde{S}_{t,d,0,K^{max}}$$

$$\forall d \in \mathcal{D} \ ,$$

$$S_{t,d,r,k} = S_{t-1,d,r+1,k} + \tilde{S}_{t,d,r,k}$$

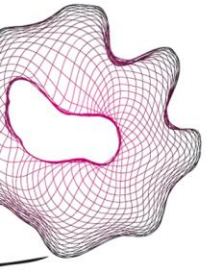
$$\forall [d,r,k] \in \mathcal{D} \times \mathcal{R} \setminus \{0, R^{max}\} \times \mathcal{K} \ ,$$

$$S_{t,d,R^{max},k} = \tilde{S}_{t,d,R^{max},k}$$

$$\forall [d,k] \in \mathcal{D} \times \mathcal{K} \ .$$

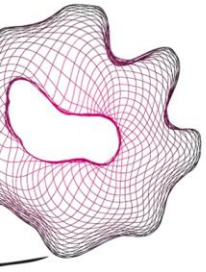
- Bellman optimality equation:

$$V_t(S_t) = \min_{x_t \in \mathcal{X}(S_t)} \left(C(S_t, x_t) + \sum_{\omega \in \Omega} \mathbb{P}^{\Omega}(W_{t+1} = \omega) V_{t+1}\left(S^M(S_t, x_t, \omega)\right) \right) \forall S_t \in \mathcal{S}_t$$



SOLVING AN MDP

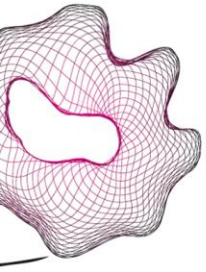
- Value iteration ➔ Iterations of 1-step value function updates (or backwards induction for finite horizon problems)
- Policy iteration ➔ Evaluate value function for given policy and greedily update policy
- Linear programming ➔ Solve the Bellman optimality equation as used in value iteration for all decision epochs at once



THREE CURSUS OF DIMENSIONALITY

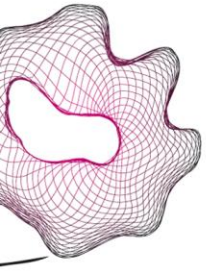
- Three possible cursus of dimensionality:
  - State space S_t might be too large to evaluate $V_t(S_t)$ for all states:
  - Decision space $\mathcal{X}_t(S_t)$ might be too large to evaluate the impact of every decision.
  - Outcome space (possible states for the next time period) might be too large to compute the expectation of cost-to-go.
- Value iteration might become intractable
- Similar arguments apply to the other two methods (policy iteration and linear programming)

➡ Solve approximately…



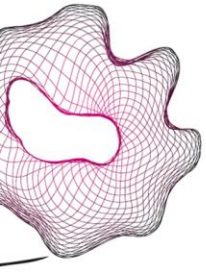
SOLVE MDPs APPROXIMATELY

- Reinforcement Learning: class of methods to solve MDPs approximately… (and more)
- Solve approximately:
  - Approximate value iteration ➔ Typical approach used in Approximate Dynamic Programming (ADP)
  - Approximate policy iteration ➔ The basis of, e.g., AlphaZero
  - Linear programming based ADP



PART 4

Approximate Dynamic Programming (ADP)



APPROXIMATE DYNAMIC PROGRAMMING

- We replace the original optimality equation

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t)\, V_{t+1}(S_{t+1})$$

where

$$S_{t+1} = S^M(S_t, x_t, W_{t+1})$$

- With the following

$$\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t)\, \bar{V}_{t+1}^{n-1}(S_{t+1})$$

where

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}(\omega^n))$$

APPROXIMATE DYNAMIC PROGRAMMING

- We replace the original optimality equation

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t) V_{t+1}(S_{t+1})$$

where

1

Using a value function approximation
This allows us to step forward in time

- With the following

$$\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t) \bar{V}_{t+1}^{n-1}(S_{t+1})$$

where

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}(\omega^n))$$

APPROXIMATE DYNAMIC PROGRAMMING

- We replace the original optimality equation

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t)\, V_{t+1}(S_{t+1})$$

where

S_{t+1}

2

Generating sample paths

$$\omega^n = (W_1, W_2, \ldots, W_T),\ W_t^n = W_t(\omega^n)$$

- With the following

$$\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t)\, \bar{V}_{t+1}^{n-1}(S_{t+1})$$

where

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}(\omega^n))$$

APPROXIMATE DYNAMIC PROGRAMMING

- We replace the original optimality equation

$$V_t(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t)\, V_{t+1}(S_{t+1})$$

where

$$S_{t+1} = S^M(S_t, x_t, W_{t+1})$$

3

Learning through iterations

- With the following

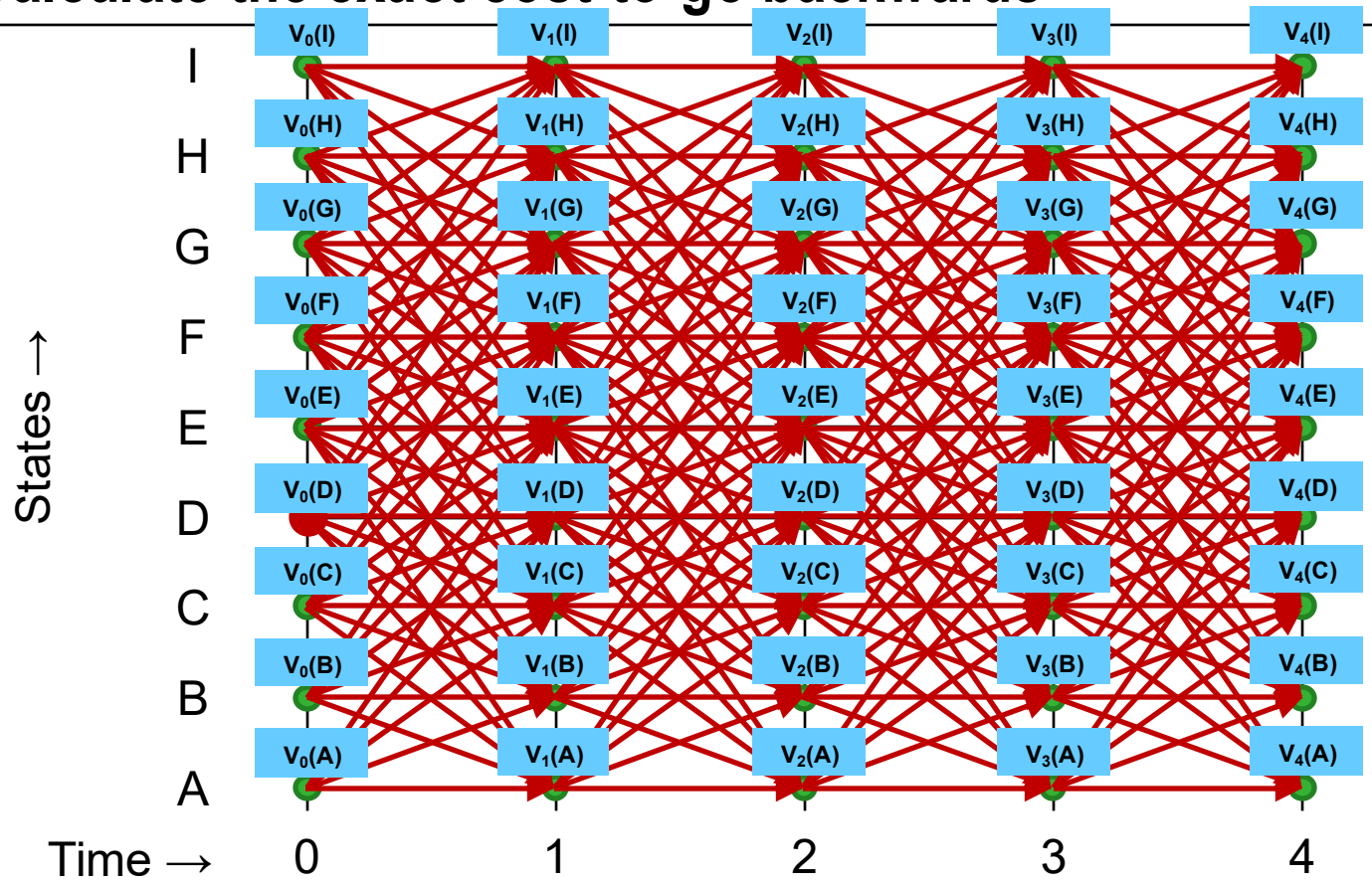
$$\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1}|S_t, x_t)\, \bar{V}_{t+1}^{n-1}(S_{t+1})$$

where

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}(\omega^n))$$

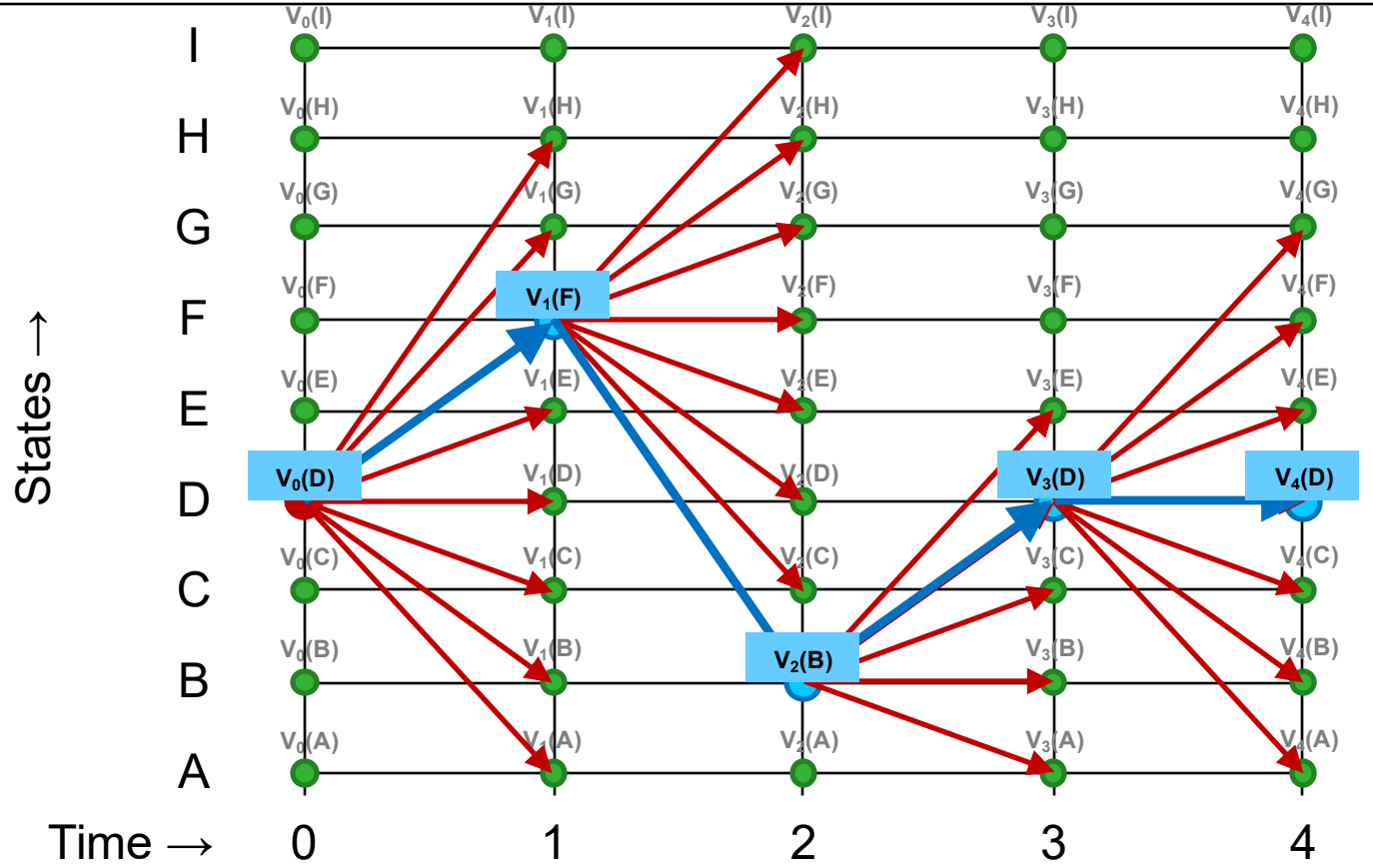
DYNAMIC PROGRAMMING (DP)

Calculate the exact cost-to-go backwards



APPROXIMATE DYNAMIC PROGRAMMING (ADP)

Learn cost-to-go forwards iteratively: single pass



APPROXIMATE DYNAMIC PROGRAMMING (ADP)

Learn cost-to-go forwards iteratively: double pass

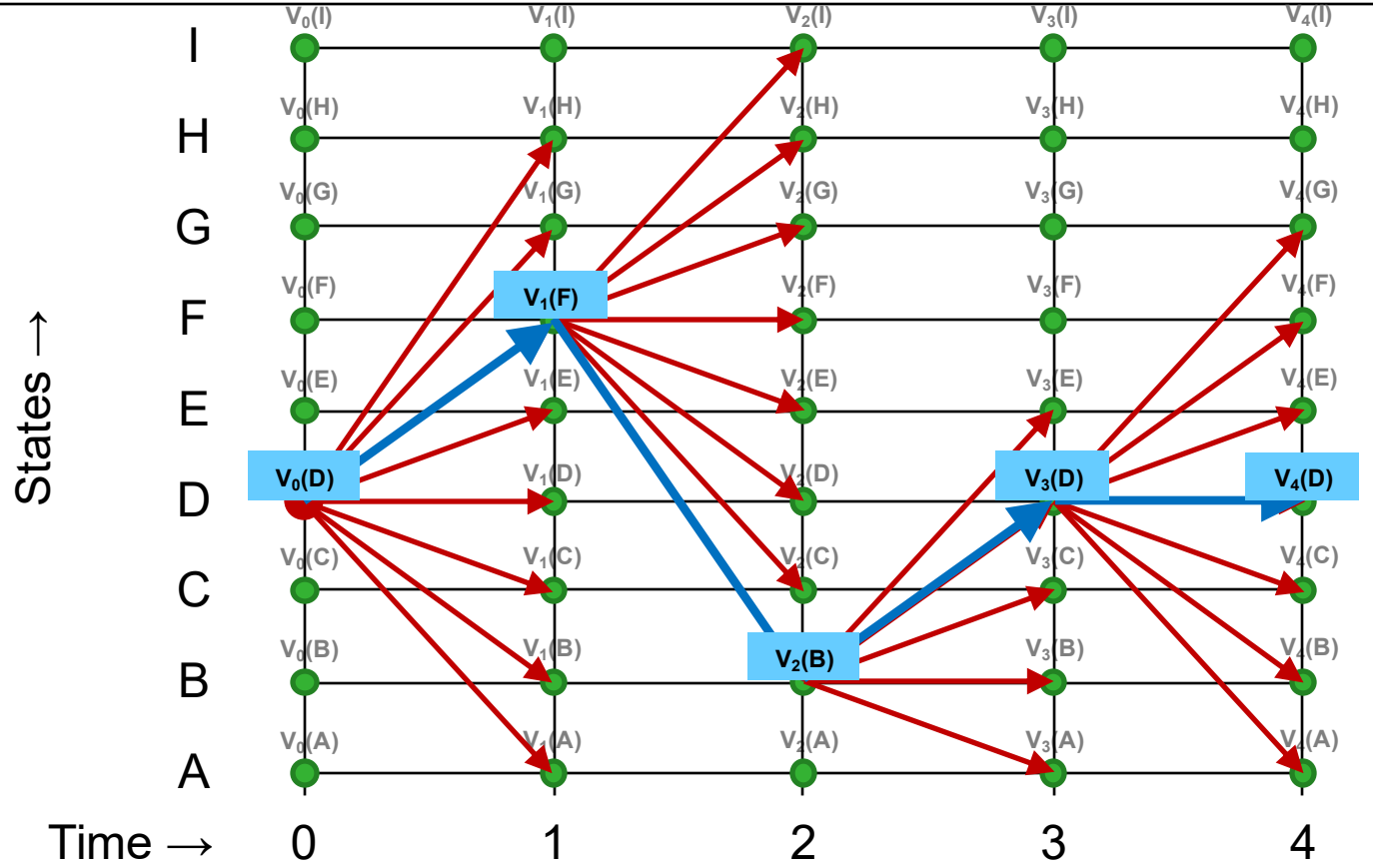
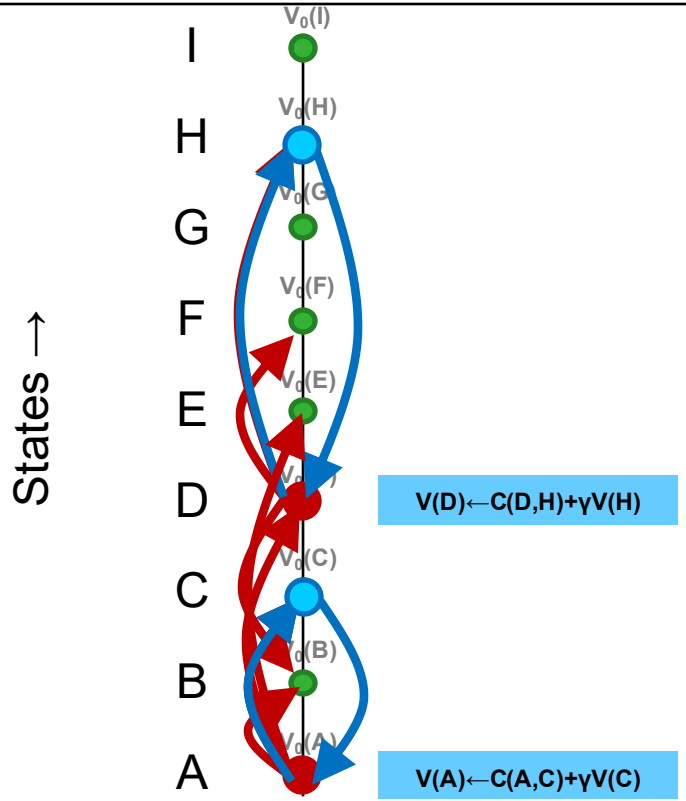


ILLUSTRATION INFINITE HORIZON



PROBLEM

- In the previous slides, we used $V_{t+1}(S_{t+1})$ to make a decision in $V_t(S_t)$, but this requires calculating the expectation.
- Calculating the expectation:

$$\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t(S_t)} C(S_t, x_t) + \gamma \mathbb{E}\{\bar{V}_{t+1}^n(S_{t+1}) | S_t\}$$

$$\mathbb{E}\{\bar{V}_{t+1}^n(S_{t+1}) | S_t\} = \sum_{S_{t+1} \in \mathcal{S}} \mathbb{P}(S_{t+1} | S_t, x_t)\, \bar{V}_{t+1}^n(S_{t+1})$$

SOLUTION 1: Q-LEARNING

- Q-value $\bar{Q}_t^n(S_t, x_t)$: estimate of the value $Q_t(S_t, x_t)$ of taking action x_t in state S_t
- Decision: $x_t^n = \arg\max_{x_t \in \mathcal{X}_t} \bar{Q}_t^{n-1}(S_t, x_t)$
- Observation: contribution $\hat{C}(S_t, x_t^n)$ and sample $W_{t+1}(\omega^n)$
- Update value:

$$\hat{q}_{t+1}^n = \hat{C}(S_t, x_t^n) + \gamma \max_{x_{t+1} \in \mathcal{X}_{t+1}} \bar{Q}_{t+1}^{n-1}(S^M(S_t, x_t^n, W_{t+1}(\omega^n)), x_{t+1})$$

- Updating the Q-value:

$$\bar{Q}_t^n(S_t, x_t^n) = (1-\alpha)\bar{Q}_t^{n-1}(S_t, x_t^n) + \alpha \hat{q}_{t+1}^n$$

- Popular strategy for model-free problems with small state and action spaces

SOLUTION 2: POST-DECISION STATE [1/2]

- State S_t^x that is reached, directly after a decision has been made in the current "pre-decision" state S_t, but before any new information W_{t+1} has arrived.
- Used as a single representation for the value of all the different states at $t+1$, based on S_t and the decision x_t.
- Instead of learning the values $\bar{V}_t^n(S_t)$ for all pre-decision states S_t, we can learn the values $\bar{V}_t^{x,n}(S_t^x)$ for all post-decision states S_t^x.

(sometimes I simply leave out the x in the approximate value function if its clear from the context we are dealing with the post-decision state: $\bar{V}_t^n(S_t^x) = \bar{V}_t^{x,n}(S_t^x)$)

SOLUTION 2: POST-DECISION STATE [2/2]

- Optimality equations using the post-decision state:
  - $\bar{V}_{t-1}^{x,n}(S_{t-1}^x) = \mathbb{E}\{\bar{V}_t^n(S_t)|S_{t-1}^x\}$ (1)
  - $\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t} \left(C_t(S_t, x_t) + \gamma \bar{V}_t^{x,n}(S_t^x)\right)$ (2)
  - $\bar{V}_t^{x,n}(S_t^x) = \mathbb{E}\{\bar{V}_{t+1}^n(S_{t+1})|S_t^x\}$
- Substitute (3) into (2) → Bellman's e
  - $\bar{V}_t^n(S_t) = \max_{x_t \in \mathcal{X}_t} (C_t(S_t, x_t) + \gamma \mathbb{E}\{\bar{V}_{t+1}^{x,n}(S_{t+1})|S_t^x\})$
- Substitute (2) into (1) → Optimality equation around the post-decision state:
  - $\bar{V}_{t-1}^{x,n}(S_{t-1}^x) = \mathbb{E}\left\{\max_{x_t \in \mathcal{X}_t} \left(C_t(S_t, x_t) + \gamma \bar{V}_t^{x,n}(S_t^x)\right) \middle| S_{t-1}^x\right\}$

We can estimate this part based on observations!

OUTLINE OF THE TYPICAL ADP ALGORITHM

Step 0. Initialize $\bar{V}_t^0$ and S_0^1, set $n = 1$

Step 1. Choose a sample path ω^n

Step 2. For $t = 0,1,\dots,T$ do:

Deterministic optimization

Step 2a. Solve:

$$\hat{v}_t^n = \max_{x_t \in \mathcal{X}_t} \left(C_t(S_t, x_t) + \gamma \bar{V}_t^{n-1}\left(S^{M,x}(S_t, x_t)\right)\right)$$

Let x_t^n be the best decision and let $S_t^x = S^{M,x}(S_t, x_t)$

Statistics

Step 2b. Update the value function:

$$\bar{V}_{t-1}^n(S_{t-1}^x) = (1-\alpha)\bar{V}_{t-1}^{n-1}(S_{t-1}^x) + \alpha \hat{v}_t^n$$

Simulation

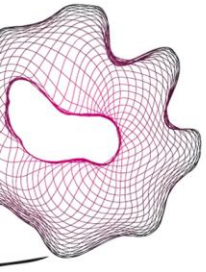
Step 2c. Compute new pre-decision state:

$$S_{t+1} = S^M\left(S_t, x_t^n, W_{t+1}(\omega^n)\right)$$

Update the previous post-decision state!

Step 3. Increment n. If $n \leq N$ go to Step 1.

Step 4. Return the value functions $(\bar{V}_t^N)_{t=1}^T$



IMPORTANT CHARACTERISTICS

- The essence of approximate dynamic programming is to replace the true value function $V_t(S_t)$ with some sort of statistical approximation that we refer to as $\bar{V}_t^n(S_t)$.
- Instead of working backwards through time (computing the value of being in each state), ADP steps forward through time, following a sequence of sample realizations (using simulation) and learning an approximation of the value function (using statistical techniques).
- However, there are different variations that combine stepping forward in time with backward sweeps to update the value of being in a state (see TD(0)/TD(1)).

CHALLENGES

Randomize initial state? Learn from a given state or for all possible starting states?

Step 0. Initialize $\bar{V}_t^0$ and S_0^1, set $n = 1$

Step 1. Choose a sample path ω^n

Step 2. For $t = 0,1,\dots,T$ do:

Step 2a. Solve:

$$\hat{v}_t^n = \max_{x_t \in \mathcal{X}_t} \left(C_t(S_t, x_t) + \gamma \bar{V}_t^{n-1}\left(S^{M,x}(S_t, x_t)\right)\right)$$

Let x_t^n be the best decision and let $S_t^x = S^{M,x}(S_t, x_t)$

Step 2b. Update the value function:

$$\bar{V}_{t-1}^n(S_{t-1}^x) = (1-\alpha)\bar{V}_{t-1}^{n-1}(S_{t-1}^x) + \alpha\hat{v}_t^n$$

Step 2c. Compute new pre-decision state:

$$S_{t+1} = S^M\left(S_t, x_t^n, W_{t+1}(\omega^n)\right)$$

Step 3. Increment n. If $n \leq N$ go to Step 1.

Step 4. Return the value functions $(\bar{V}_t^N)_{t=1}^T$

Explore vs. exploit and on-policy or off-policy?

What value-function approximation (VFA) to use?

How to update the VFA?

CHALLENGE 1: EXPLORATION VS EXPLOITATION

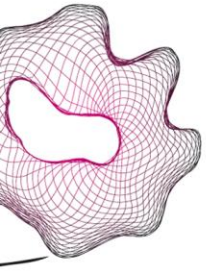
- Exploration vs. exploitation:
  - Exploitation: we do what we currently think is best
  - Exploration: we choose to try something and learn more (information collection)
  - To avoid getting stuck in a local optimum, we have to explore. But what do we want to explore and for how long? Do we need to explore the whole state space?
  - Do we update the value functions using the results of the exploration steps (on-policy) or do we want to perform off-policy control?
  - Typically for infinite horizon is to be off-policy in case of exploration. For finite we might need to be on-policy…
- Simple recipe: ε-Greedy (explore with probability ε)

CHALLENGE 2: UPDATING

- Stepsize for $\bar{V}_t^n(S_t^x, x_t^n)=(1-\alpha_{n-1})\bar{V}_t^{n-1}(S_t^x, x_t^n)+\alpha_{n-1}\hat{v}_{t+1}^n$
- Options:
  - Constant stepsize $\alpha_{n-1} = \begin{cases} 1 \; if \; n = 1 \\ \bar{\alpha} \; otherwise \end{cases}$
  - Variable stepsize $\alpha_{n-1} = 1/n$
  - Harmonic stepsize $\alpha_{n-1} = {}^{a}/_{a+n-1}$
  - Polynomial stepsize $\alpha_{n-1} = n^{-\beta}$
  - McClain's formula $\alpha_n = {}^{(\alpha_{n-1})}/_{(1+\alpha_{n-1}-\bar{\alpha})}$
  - Stochastic stepsizes: depend on the observations $\hat{v}_{t+1}^n$. If we have high fluctuations or an increasing/decreasing pattern, use relatively large stepsizes.

CHALLENGE 3: VALUE FUNCTION APPROXIMATION

- Next part…



PART 5

Value Function Approximation

CHALLENGE

- Typically not realistic to learn the value of each state separately
- So, we need some approximation for the 'future' costs $\bar{V}_t^n(S_t^x)$
- This approximation should be…
  - Computationally tractable
  - Provide a good approximation of the actual value
  - Is able to generalize across the state space

TYPICAL SOLUTIONS

1. Lookup-table: one entry per state S_t for each stage t
2. Aggregation: reduction of the state space
3. Regression models (parametric representations)
4. Piecewise Linear Approximations
5. Any machine learning model, e.g., neural networks

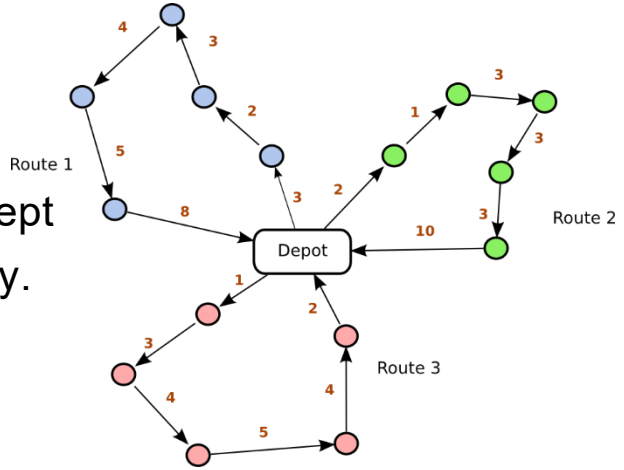
➢ Let's focus on the first 3 options for now… (more on option 5 in the other presentations today)

LOOKUP-TABLE

- One entry per state S_t for each stage t
- To generalize across states:
  - Update multiple entries with each observation
  - Or use approximations for unobserved states
- Within the lookup table, you can use:
  - Local approximations like nearest neighbour (value of an unknown state is given by a weighted average of its nearest neighbours)
  - Kernel regression (include correlations between states).

AGGREGATION [1/3]

- Simplest form: reduction of the state space:
  - Create clusters/groupings (e.g., instead of having all possible speeds, we now distinguish between slow, normal, fast).
  - Define features (e.g., instead of having all possible schedules of all of our trucks, we just keep track of the amount of slack time for our complete fleet of vehicles).
- Example of the use of features:
  - Dynamic VRP
  - Online decision whether to accept a customer for next day delivery.
  - What is the state?
  - Possible aggregated states?



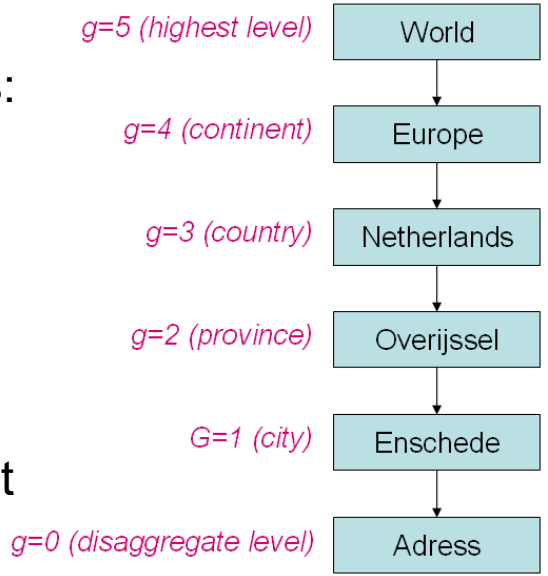
AGGREGATION [2/3]

- Hierarchical: combining different levels of aggregation

- Our estimate is a weighted combination of the estimates at different aggregation levels:

$$V_x^n = \sum_{g \in G} w_x^{g,n} V_x^{g,n}$$

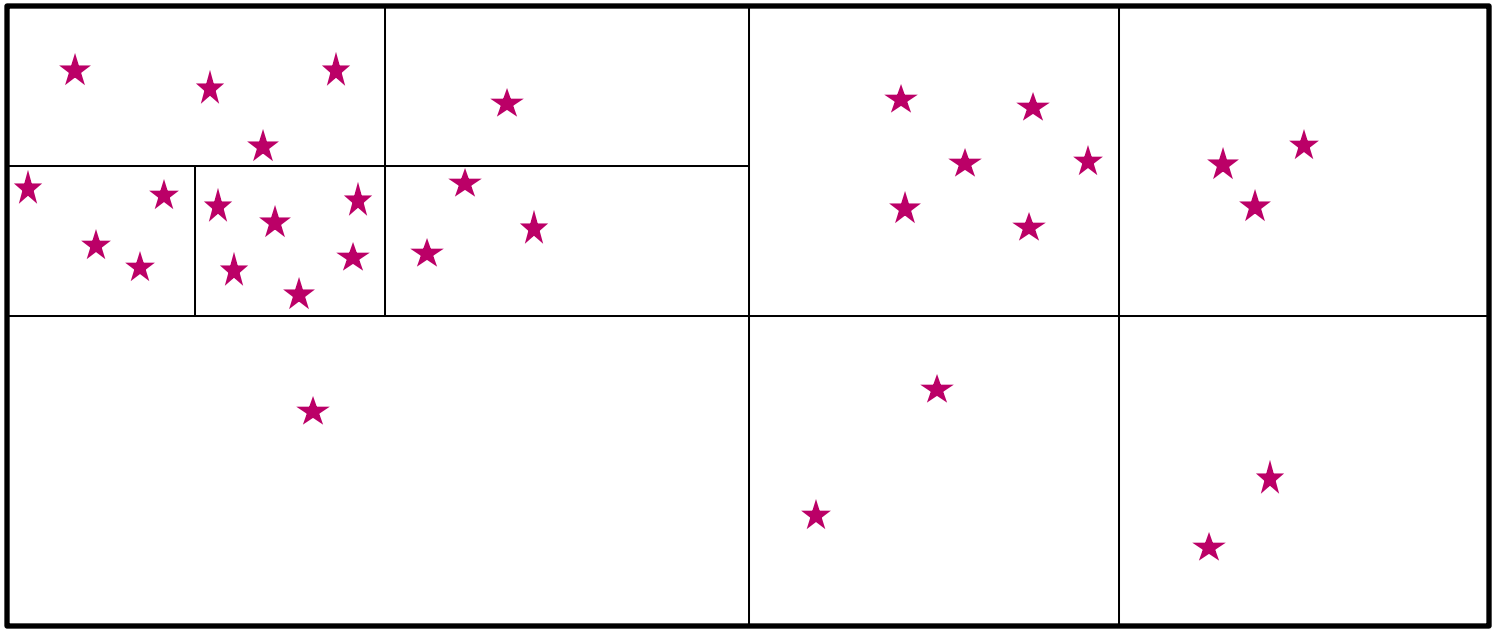
- Intuition: highest weight to levels with lowest sum of variance and bias, see [1] and [2]



[1] M.R.K. Mes, W.B. Powell, and P.I. Frazier (2010). Hierarchical Knowledge Gradient for Sequential Sampling.
[2] A. George, W.B. Powell, and S.R. Kulkarni (2008). Value Function Approximation using Multiple Aggregation for Multiattribute Resource Management.

AGGREGATION [3/3]

- Dynamic: change the aggregation level depending on the observations



REGRESSION: PARAMETRIC VFA [1/2]

- Basis functions:
  - Particular features of the state have a significant impact on the value function
  - Create basis functions for each individual feature
  - A basis function extracts the value of a feature from a state. For example, if the state consists of the planned vehicle routes so far, then a basis function might extract the remaining time in the planned vehicles routes

REGRESSION: PARAMETRIC VFA [2/2]

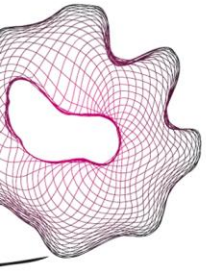
- We now define the value function approximation as:

$$\bar{V}_t^n(S_t^x) = \sum_{f \in \mathcal{F}} \theta_f^n \phi_f(S_t^x), \qquad \forall t \in \mathcal{T}$$

- Where θ_f^n is a weight for each feature $f \in \mathcal{F}$, and $\phi_f(S_t^x)$ is the value of the particular feature given the post-decision state S_t^x.
- Typically #combinations of feature values < #states (so we can also use them in lookup table approach), but with regression we even do not have to learn the value of combinations, just one weight per basis function!

FEATURES?

State variable [R1]
State variable [R2]
State variable [R3]
State variable [R1-nr]
State variable [R2-nr]
State variable [R3-nr]
State variable [G1]
State variable [G2]
State variable [G3]
State variable [G1-nr]
State variable [G2-nr]
State variable [G3-nr]
State variable [B1]
State variable [B2]
State variable [B3]
State variable [B1-nr]
State variable [B2-nr]
State variable [B3-nr]
Number destinations MUST
Sum freights MUST
Number destinations MAY
Sum freights MAY
Number destinations FUTURE
Sum freights FUTURE
Sum All Freights
Constant
Destination Must Future [No]
Destination Must Future [B]
Destination Must Future [G]
Destination Must Future [R]
Destination Must Future [GB]
Destination Must Future [RB]
Destination Must Future [RG]
Destination Must Future [RGB]
Destination May Future [No]
Destination May Future [B]
Destination May Future [G]
Destination May Future [R]
Destination May Future [GB]
Destination May Future [RB]
Destination May Future [RG]
Destination May Future [RGB]



EXAMPLES OF FEATURES

1. Each state variable: number of freights with specific attributes.
2. Number of delivery and pickup freights that are not yet released for transport, per destination (future freights).
3. Number of delivery and pickup freights that are released for transport and whose due-day is not immediate, per destination (may-go freights).
4. Binary indicator for each destination to denote the presence of urgent delivery or pickup freights (must-visit destination).
5. Some power function (e.g., ^2) of each state variable (non-linear components in costs).

➢ We test various combinations…

EXPERIMENTS TO EVALUATE VFA DESIGN

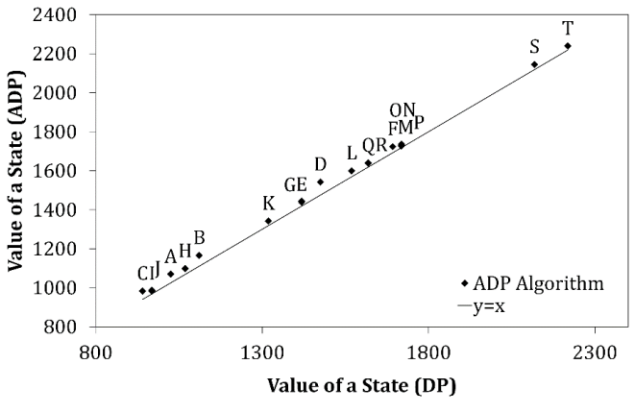
- Small instance:

	R^2				Performance			
Type	I1S	I2S	I1R	I2R	I1S	I2S	I1R	I2R
VFA1	0.89	0.89	0.63	0.64	16.0%	8.0%	5.2%	6.6%
VFA2	0.89	0.90	0.69	0.68	14.0%	7.0%	5.9%	7.7%
VFA3	0.89	0.89	0.55	0.55	8.0%	7.0%	5.3%	6.8%

- Large instance:

Type	I3S	I4S	I3R	I4R
VFA1	-22.4%	-34.3%	-6.5%	-7.4%
VFA2	-14.7%	-18.5%	-7.0%	-5.8%
VFA3	-30.0%	-36.4%	-7.8%	-5.6%

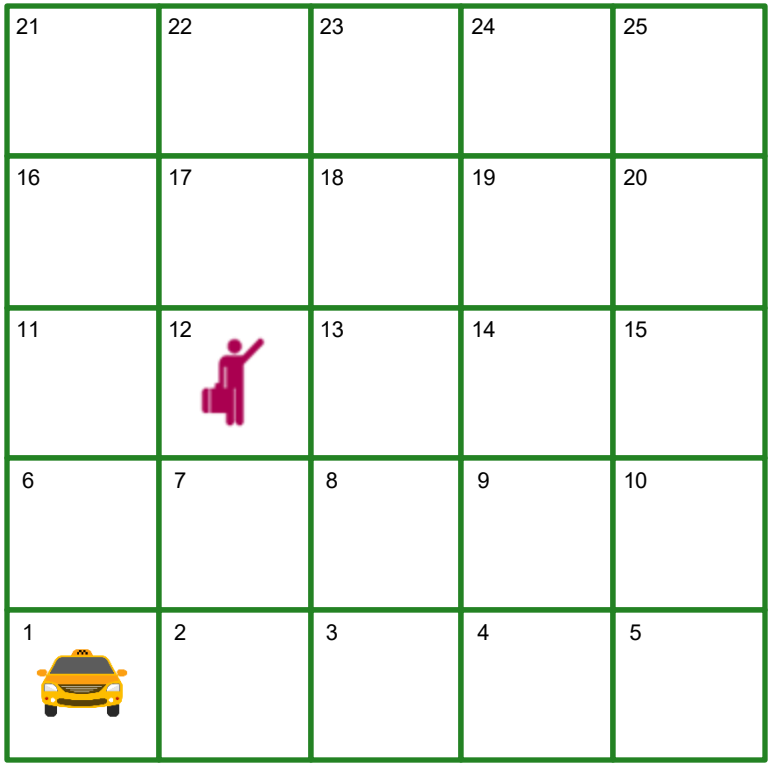
- Example convergence:



PART 6

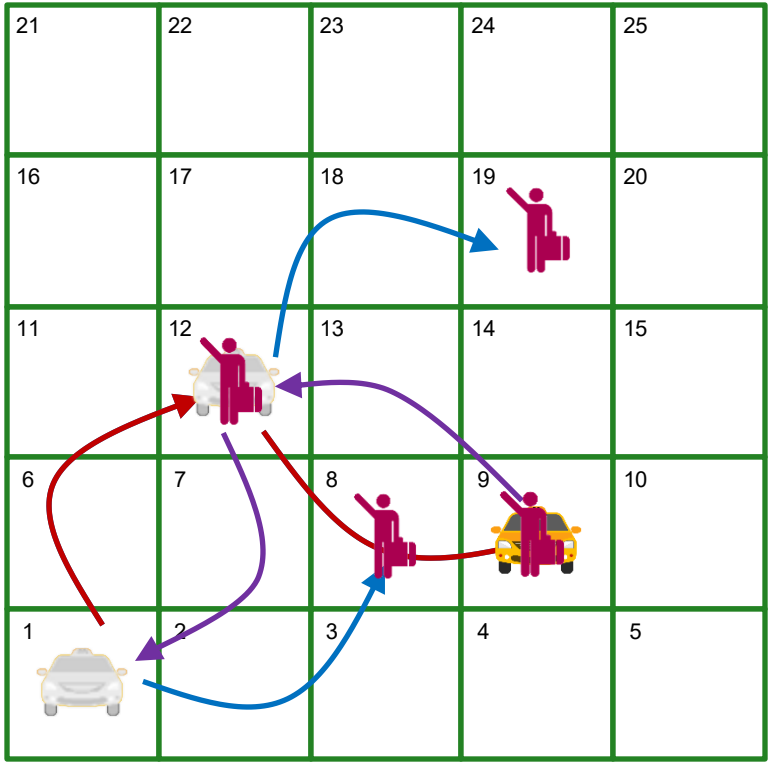
Preview of different forms of RL…

ILLUSTRATION RL OPTIONS



- Nomadic trucker problem
- 1 taxi
- Imbalanced network
- Objective: maximize infinite horizon discounted rewards
- Learn which trips to accept and when/where to move empty to
- Here simplified to just visiting a customer

ILLUSTRATION RL OPTIONS



- Approximate Value iteration
- 1-step lookahead & update
- ADP, SARSA, TD(0)
- Post-decision state

$$\tilde{a} = \arg\max_{a \in \{8,12\}} \{R(1,a) + \bar{V}^n(a)\}$$

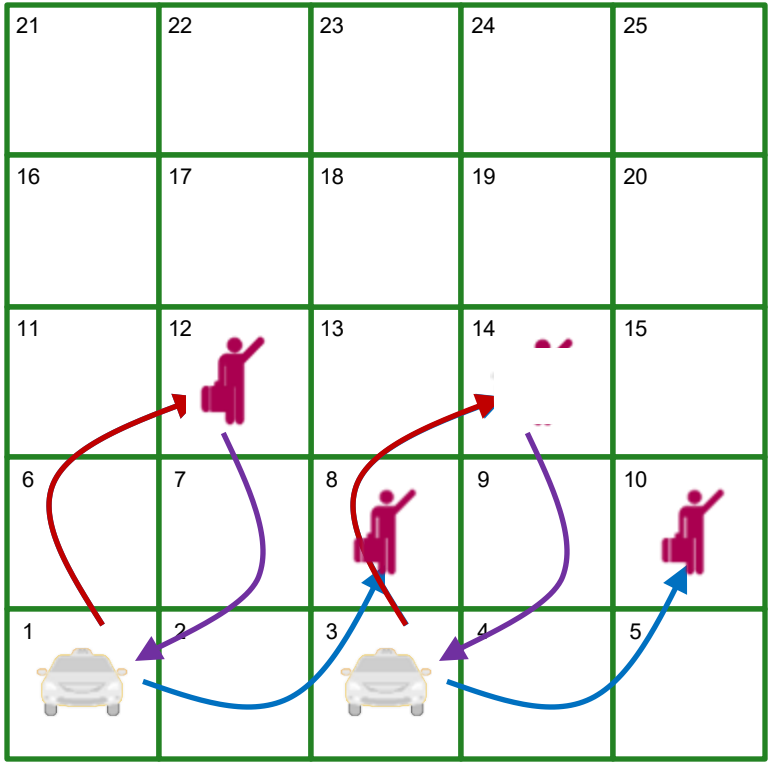
$$\bar{V}^{n+1}(1) \leftarrow R(1,12) + \bar{V}^n(12)$$

- Alternatively (Q-learning)

$$\tilde{a} = \arg\max_{a \in \{8,12\}} \{\bar{Q}^n((1,8,12),a)\}$$

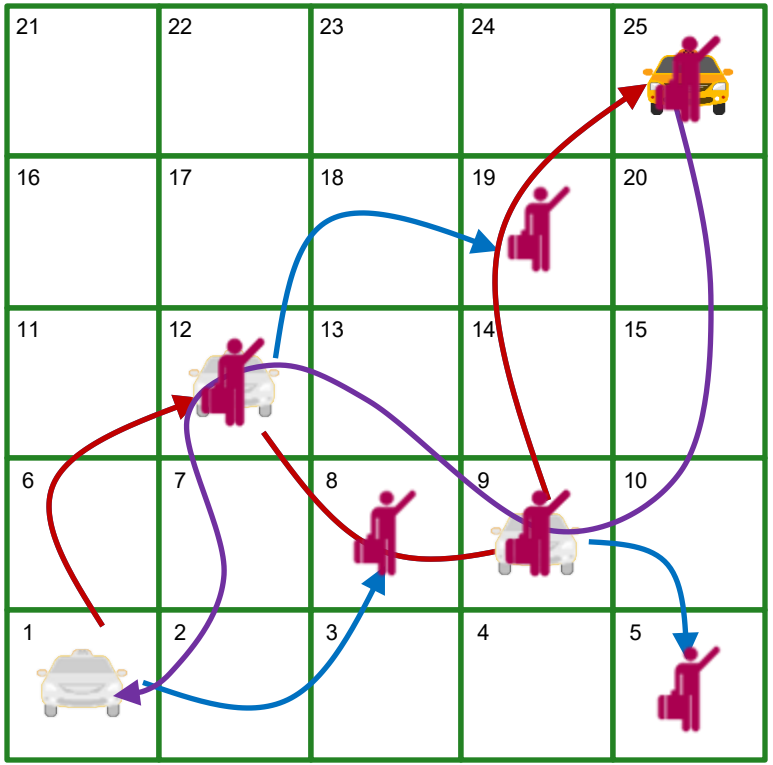
$$\bar{Q}^n\big((1,8,12),12\big) \leftarrow R(1,12) + \arg\max_{a \in \{?,?\}} \{\bar{Q}^n((12,?,?),a)\}$$

ILLUSTRATION RL OPTIONS



- Approximate Value iteration
- 1-step lookahead & update
- Explore, off-policy learning

ILLUSTRATION RL OPTIONS



- Approximate Value iteration
- n-step lookahead & update
- n-SARSA, MC learning, TD(1)

$$a_1^* = \arg\max_{a\in\{8,12\}} \{R(1,a) + \bar{V}^n(a)\}$$

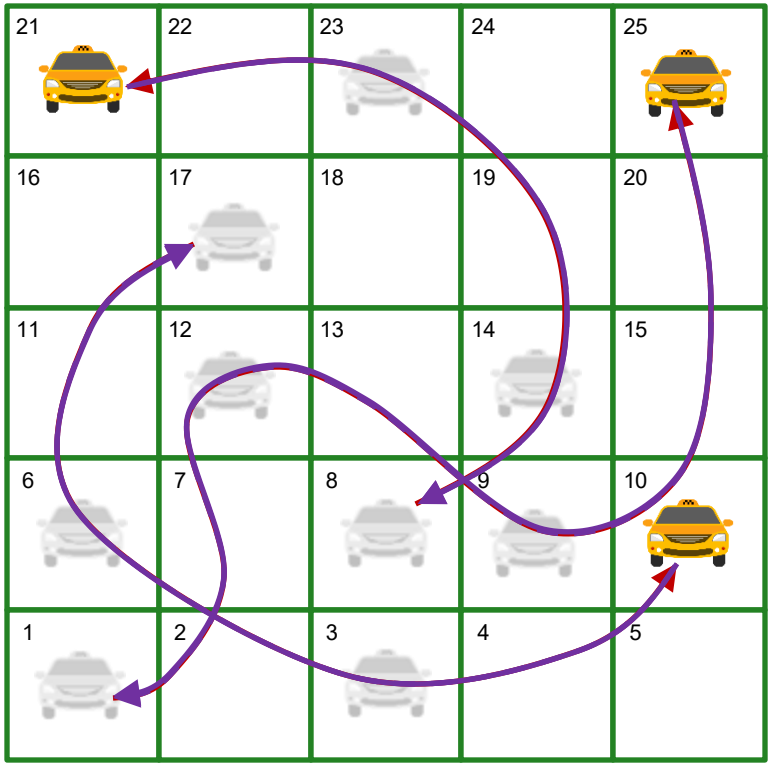
$$a_2^* = \arg\max_{a\in\{9,19\}} \{R(12,a) + \bar{V}^n(a)\}$$

$$a_3^* = \arg\max_{a\in\{5,25\}} \{R(9,a) + \bar{V}^n(a)\}$$

$$\bar{V}^{n+1}(1) \leftarrow R(1,12) + R(12,9) + R(9,25) + \bar{V}^n(25)$$

- Possibly use replications…

ILLUSTRATION RL OPTIONS



- Approximate Value iteration
- n-step lookahead with batch updating

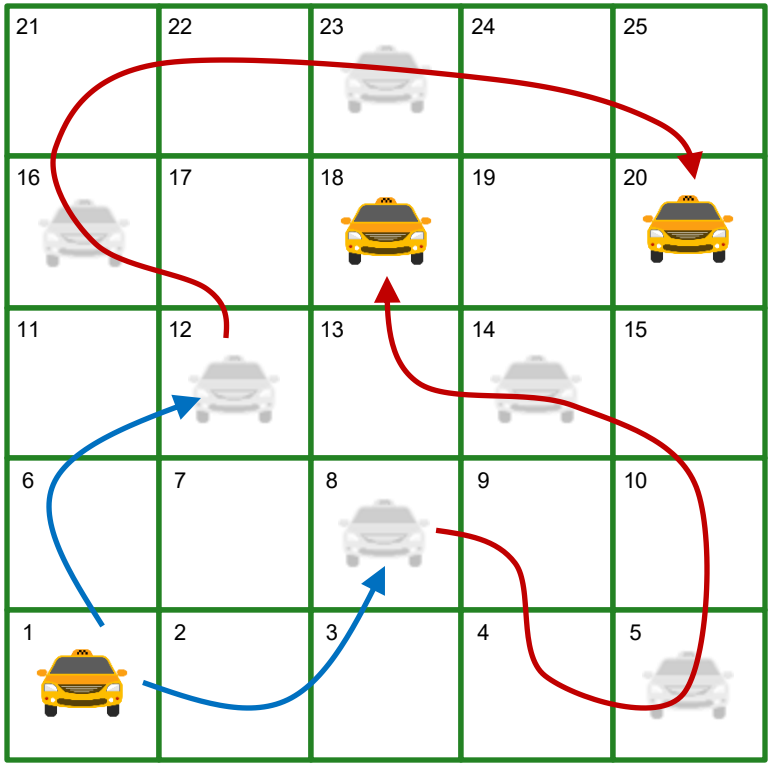
$$\bar{V}^{n+1}(1) \leftarrow R(1,12) + R(12,9) + R(9,25) + \bar{V}^{n}(25)$$

$$\bar{V}^{n+1}(8) \leftarrow R(8,14) + R(14,23) + R(23,21) + \bar{V}^{n}(21)$$

$$\bar{V}^{n+1}(17) \leftarrow R(17,6) + R(6,3) + R(3,10) + \bar{V}^{n}(10)$$

- Ideal for NN-based VFA
- Or Deep Q-Learning

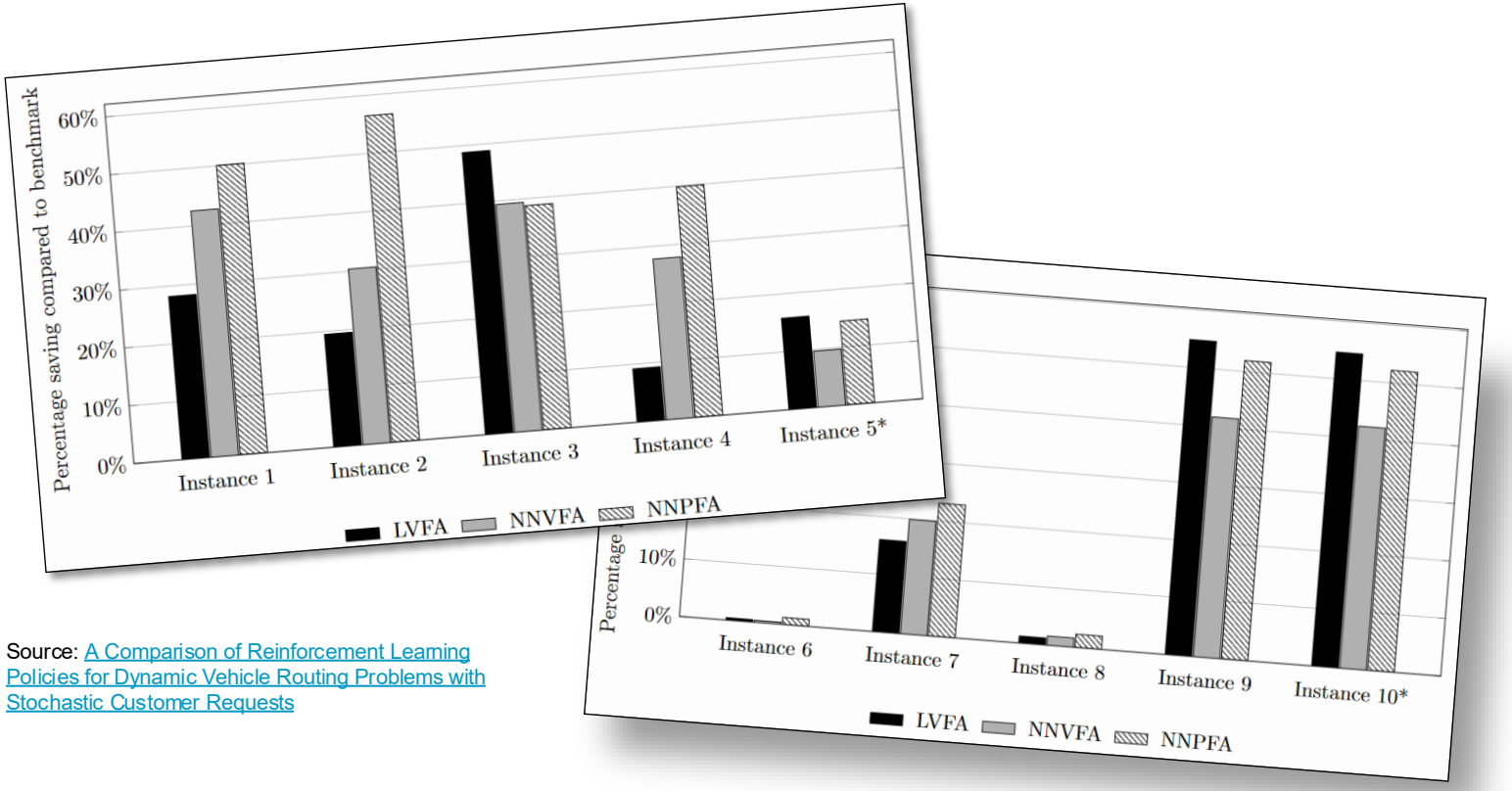
ILLUSTRATION RL OPTIONS



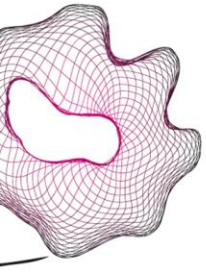
- Approximate Policy iteration
- n-step lookahead (rollout) with batch updating

1. Sample a state
2. Evaluate all possible actions
3. After the action, run multiple long simulations following the current policy
4. Evaluate best action
5. Repeat the above for many states
6. Update NN mapping states to actions

NOT CLEAR UP FRONT WHAT FORM IS BEST



Source: A Comparison of Reinforcement Learning Policies for Dynamic Vehicle Routing Problems with Stochastic Customer Requests



FURTHER READING

- Introduction Approximate Dynamic Programming:
  - Approximate Dynamic Programming by Practical Examples
- Trucks & Barges:
  - Anticipatory freight selection in intermodal long-haul round-trips
  - A Simulation Game for Anticipatory Scheduling of Synchromodal Transport
  - Comparison of Manual and Automated Decision-Making with a Logistics Serious Game
- Patient Admission Planning:
  - Patient admission planning using Approximate Dynamic Programming
- Comparison of different RL forms:
  - A Comparison of Reinforcement Learning Policies for Dynamic Vehicle Routing Problems with Stochastic Customer Requests

UNIVERSITY OF TWENTE.

QUESTIONS?

Martijn Mes
Full professor
University of Twente
Faculty of Behavioural, Management and Social sciences
Dept. Industrial Engineering and Business Information Systems

Contact
+31-534894062
m.r.k.mes@utwente.nl
https://www.utwente.nl/en/bms/iebis/staff/mes/