Reinforcement Learning for Container Shuffling

Intuition

A form of simulation

Agent:

Decision-making

Maximise long-term reward

Many interactions with the environment

Environment:

Provide feedback in a loop

Can be stochastic

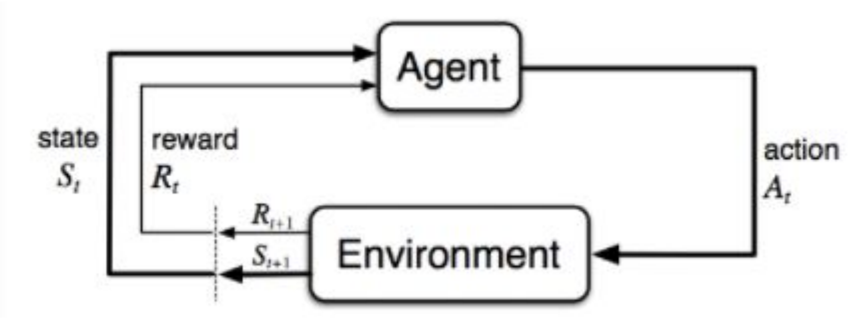
Policy:

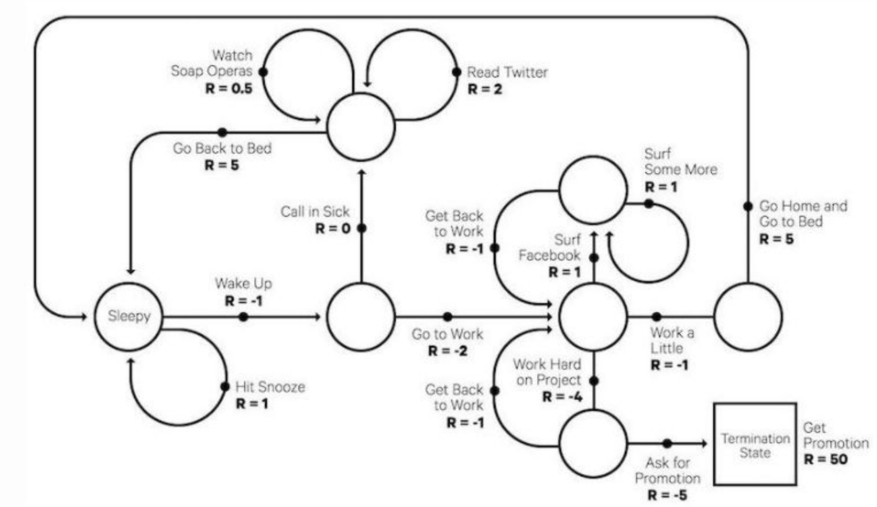
Influenced by reward feedback from environment

Actions are parameterized by some weights

Balance exploration vs exploitation

Aim to: learn an optimal policy over time





Application to Industry Applications

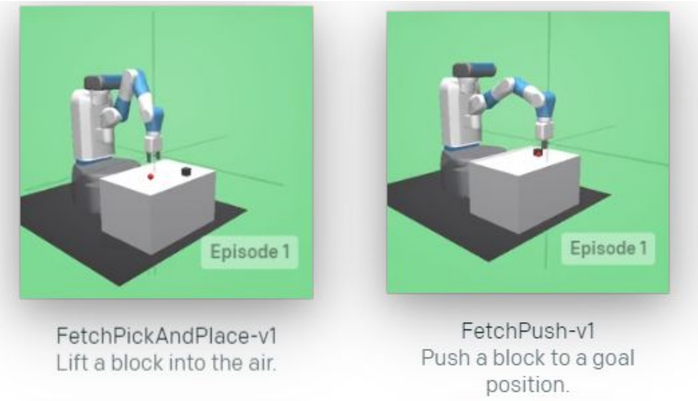
When it is useful

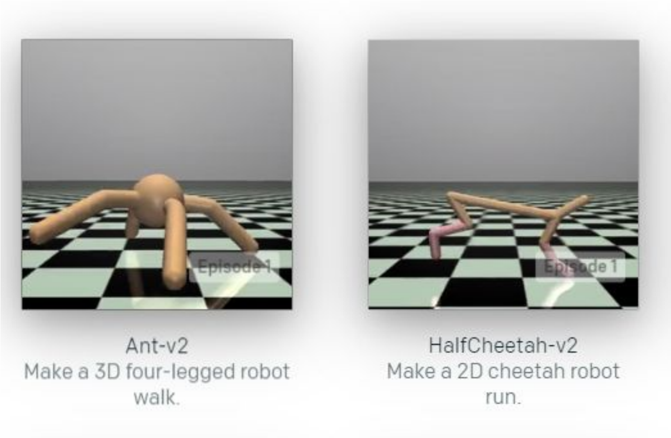
In a well-defined, complex environment with stochastic elements

Large state spaces

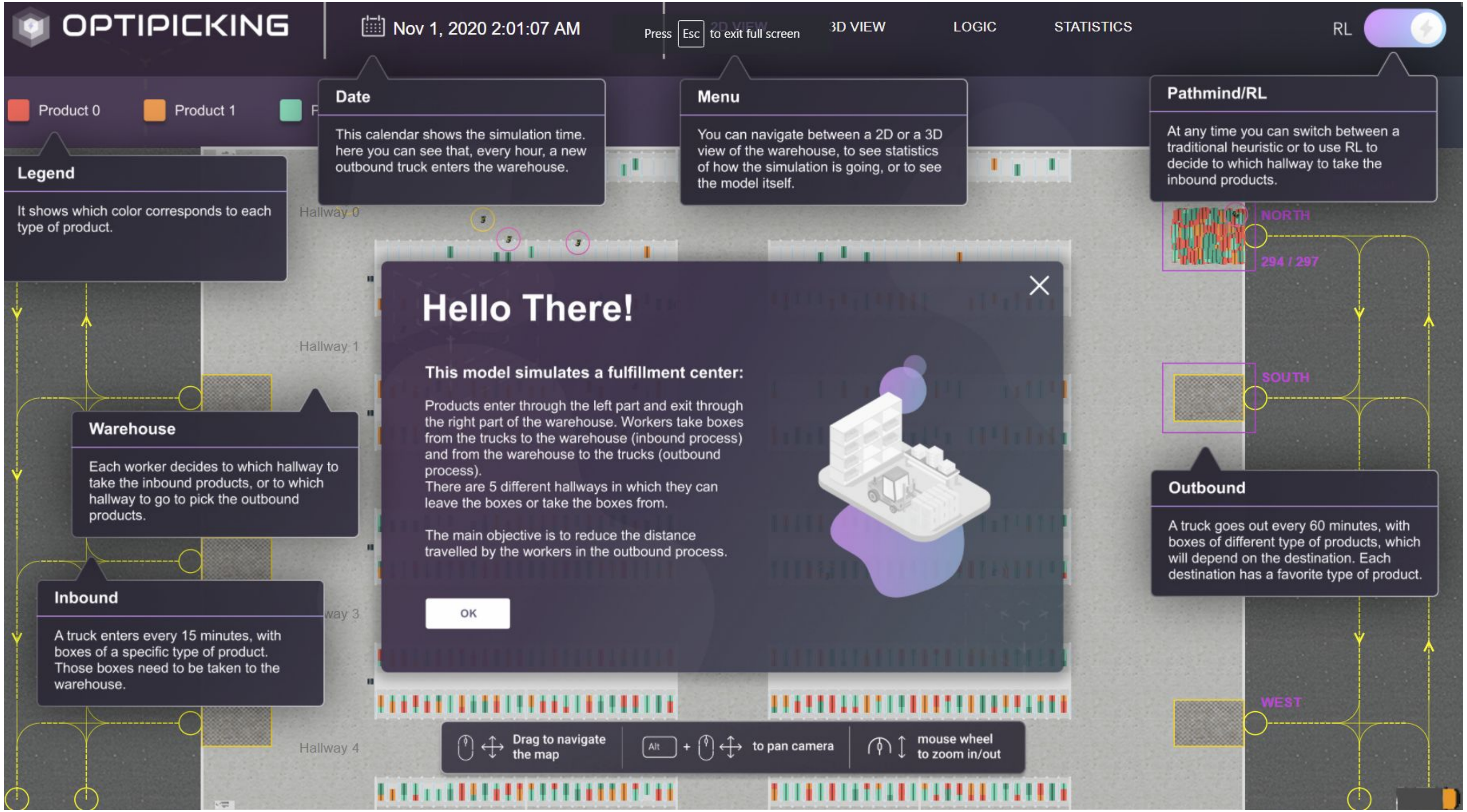
Can accommodate multiple contradictory objectives

Discrete/Continuous actions





Application to Industry Applications



Application to Industry Applications

Application to Container Re-shuffling

Steps taken:

Formulate as an RL task

Enable

Problem

Many current container slots are not stacked optimally

Reasons:

- Vessels arrive at different times, have different leaving times
- Limited space
- Difficult to plan ahead

Problems:

- Long wait time when second carrier arrives -> costly

Goals:

- Container stacking is “clean” when second carrier arrives

			720:744_INMAA_40_GP_U	22:47_DEHAM_40_HC_M	
22:47_DEHAM_40_HC_M		22:47_DEHAM_40_HC_L	720:744_INMAA_40_HC_U	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_X
22:47_DEHAM_40_GP_L		78:109_TRIST_40_HC_X	720:744_INMAA_40_HC_X	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_GP_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_M	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_HC_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_X	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	10:73_PHMNL_40_HC_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

Problem

Container re-shuffling in down times:

- Carried out by human operators using intuition and experience -> model this process
- Cost metric: minimize violations
- One container moved at a time -> sequential decision-making process

			720:744_INMAA_40_GP_U	22:47_DEHAM_40_HC_M	
22:47_DEHAM_40_HC_M		22:47_DEHAM_40_HC_L	720:744_INMAA_40_HC_U	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_X
22:47_DEHAM_40_GP_L		78:109_TRIST_40_HC_X	720:744_INMAA_40_HC_X	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_GP_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_M	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_HC_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_X	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	10:73_PHMNL_40_HC_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

			34:51_INMAA_40_HC_M	10:73_PHMNL_40_HC_X	
22:47_DEHAM_40_HC_M	82:111_GRPIR_40_HC_H		34:51_INMAA_40_HC_X	10:73_PHMNL_40_AB_U	
22:47_DEHAM_40_HC_M	82:111_GRPIR_40_HC_H	720:744_INMAA_40_HC_U	78:109_TRIST_40_HC_X	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_X
22:47_DEHAM_40_HC_L	82:111_GRPIR_40_HC_H	720:744_INMAA_40_HC_X	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_L
22:47_DEHAM_40_HC_L	82:111_GRPIR_40_HC_H	720:744_INMAA_40_GP_U	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_L
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

Cost Metric

- Overstow

    Containers to be unloaded later are stacked on those earlier

- Mix Portmark

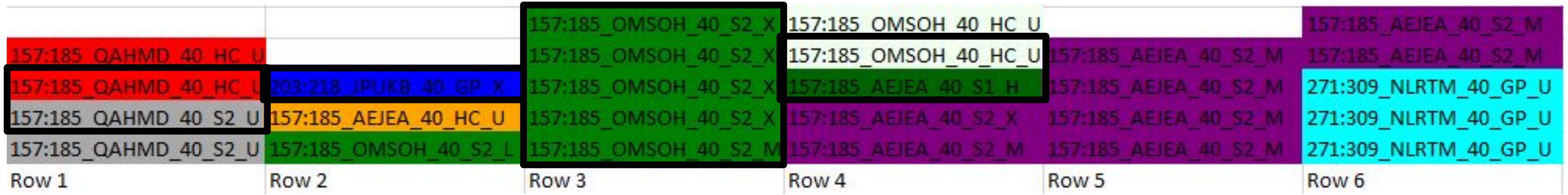
    Containers loaded onto same vessel but arriving to different destinations

- Mix Category

    Containers loaded onto same vessel but are of different categories

- Mix Weight Order

    Containers loaded onto same vessel, same portmark and category but wrong order



Reinforcement Learning application

Formulate this as a RL task

- **Agent**:

  Make decisions for each slot by interacting with the simulation environment

- **Environment**:

  Provide feedback during the simulation, implemented with OpenAI's Gym interface

- **Action**:

  Discrete action of dimension rows * (rows - 1)

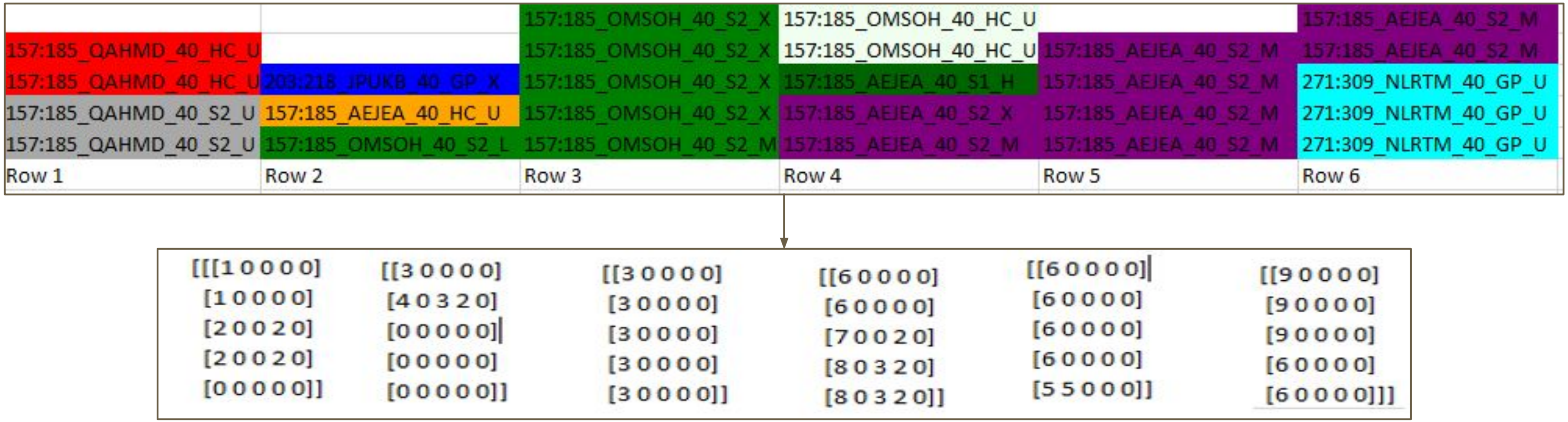
  Each container may move to any other rows except for itself

		157:185_OMSOH_40_S2_X	157:185_OMSOH_40_HC_U		157:185_AEJEA_40_S2_M
157:185_QAHMD_40_HC_U		157:185_OMSOH_40_S2_X	157:185_OMSOH_40_HC_U	157:185_AEJEA_40_S2_M	157:185_AEJEA_40_S2_M
157:185_QAHMD_40_HC_U	203:218_JPUKB_40_GP_X	157:185_OMSOH_40_S2_X	157:185_AEJEA_40_S1_H	157:185_AEJEA_40_S2_M	271:309_NLRTM_40_GP_U
157:185_QAHMD_40_S2_U	157:185_AEJEA_40_HC_U	157:185_OMSOH_40_S2_X	157:185_AEJEA_40_S2_X	157:185_AEJEA_40_S2_M	271:309_NLRTM_40_GP_U
157:185_QAHMD_40_S2_U	157:185_OMSOH_40_S2_L	157:185_OMSOH_40_S2_M	157:185_AEJEA_40_S2_M	157:185_AEJEA_40_S2_M	271:309_NLRTM_40_GP_U
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

Reinforcement Learning application

Current State:

A numerical representation of the current situation, flattened for neural network



Policy:

A neural network with input being the current state and output being a probability distribution over actions

Action masking:

Mask away invalid actions at each step, helps with convergence

Reinforcement Learning application

- **Rewards**:

    Positive rewards for every reduction in a global **current cost**

    Small negative penalty for undesirable moves such as further worsening overstow

    Small negative penalty at every step to minimise the number of moves

overstow

			720:744_INMAA_40_GP_U	22:47_DEHAM_40_HC_M	22:47_DEHAM_40_GP_X
22:47_DEHAM_40_HC_M		22:47_DEHAM_40_HC_L	720:744_INMAA_40_HC_U	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_X
22:47_DEHAM_40_GP_L		78:109_TRIST_40_HC_X	720:744_INMAA_40_HC_X	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_GP_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_M	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_HC_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_X	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	10:73_PHMNL_40_HC_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

	1. Row 5 to row 2. Before cost: 83. After cost: 73. Reward: 11.0				
[[[1 0 0 0 0] [2 0 0 2 0] [2 0 0 2 0] [1 0 0 2 0] [0 0 0 0 0]] [[3 0 0 0 0] [3 0 0 0 0] [1 0 0 0 0] [0					
			720:744_INMAA_40_GP_U		22:47_DEHAM_40_GP_X
22:47_DEHAM_40_HC_M		22:47_DEHAM_40_HC_L	720:744_INMAA_40_HC_U	10:73_PHMNL_40_AB_U	22:47_DEHAM_40_GP_X
22:47_DEHAM_40_GP_L	22:47_DEHAM_40_HC_M	78:109_TRIST_40_HC_X	720:744_INMAA_40_HC_X	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_GP_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_M	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	82:111_GRPIR_40_HC_H
22:47_DEHAM_40_HC_L	82:111_GRPIR_40_HC_H	34:51_INMAA_40_HC_X	111:147_CNTAO_40_HC_U	10:73_PHMNL_40_AB_U	10:73_PHMNL_40_HC_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

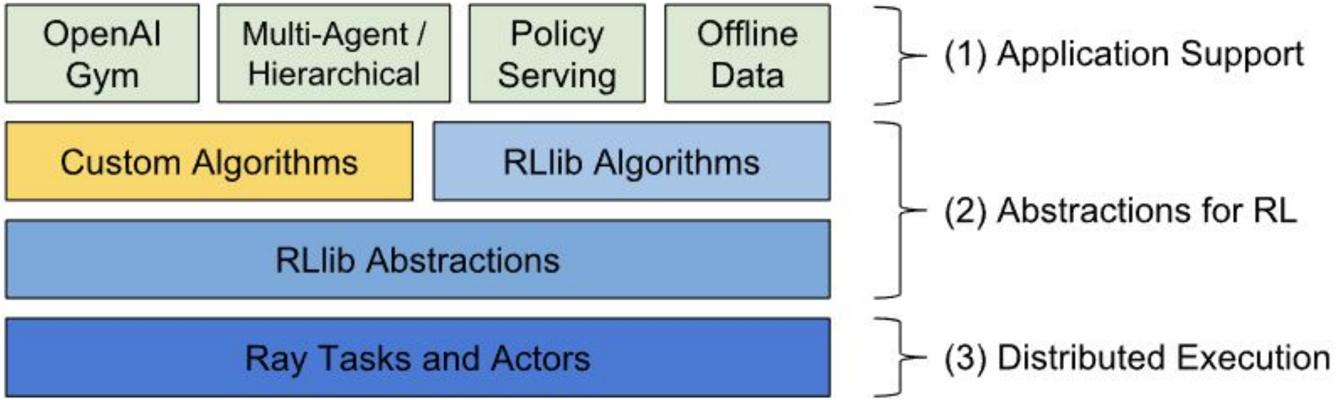
Reinforcement Learning application

Algorithm:

Proximal Policy Optimization: can overcome the problems of poor sample efficiency and high gradient variance in policy gradient methods while being simpler to implement and tune

Framework:

Ray's RLlib



Implementation

Input:

Data in Excel form

Different configuration files: shuffling types + locations + customisable constraints + hyperparameters

Execution:

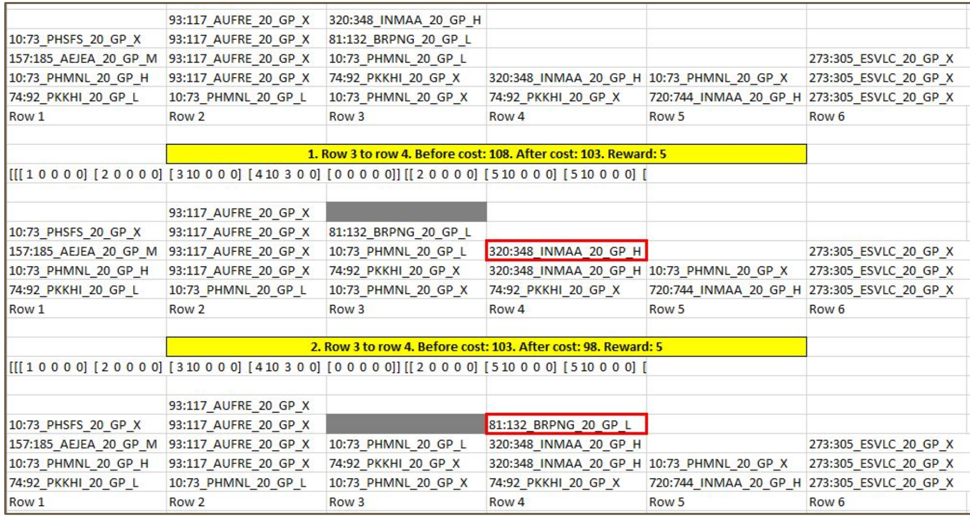
Validity and feasibility check

Solving each slot of container or multiple-slot

Output:

Sequence of movements for each slot

Visualization for step-by-step transition of the slot



	93:117_AUFRE_20_GP_X	320:348_INMAA_20_GP_H			
10:73_PHSFS_20_GP_X	93:117_AUFRE_20_GP_X	81:132_BRPNG_20_GP_L			
157:185_AEJEA_20_GP_M	93:117_AUFRE_20_GP_X	10:73_PHMNL_20_GP_L			273:305_ESVLC_20_GP_X
10:73_PHMNL_20_GP_H	93:117_AUFRE_20_GP_X	74:92_PKKHI_20_GP_X	320:348_INMAA_20_GP_H	10:73_PHMNL_20_GP_X	273:305_ESVLC_20_GP_X
74:92_PKKHI_20_GP_L	10:73_PHMNL_20_GP_L	10:73_PHMNL_20_GP_X	74:92_PKKHI_20_GP_X	720:744_INMAA_20_GP_H	273:305_ESVLC_20_GP_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6
	1. Row 3 to row 4. Before cost: 108. After cost: 103. Reward: 5				
[[[1 0 0 0 0] [2 0 0 0 0]	[3 10 0 0 0] [4 10 3 0 0]	[0 0 0 0 0]] [[2 0 0 0 0]	[5 10 0 0 0] [5 10 0 0 0] [		
	93:117_AUFRE_20_GP_X				
10:73_PHSFS_20_GP_X	93:117_AUFRE_20_GP_X	81:132_BRPNG_20_GP_L			
157:185_AEJEA_20_GP_M	93:117_AUFRE_20_GP_X	10:73_PHMNL_20_GP_L	320:348_INMAA_20_GP_H		273:305_ESVLC_20_GP_X
10:73_PHMNL_20_GP_H	93:117_AUFRE_20_GP_X	74:92_PKKHI_20_GP_X	320:348_INMAA_20_GP_H	10:73_PHMNL_20_GP_X	273:305_ESVLC_20_GP_X
74:92_PKKHI_20_GP_L	10:73_PHMNL_20_GP_L	10:73_PHMNL_20_GP_X	74:92_PKKHI_20_GP_X	720:744_INMAA_20_GP_H	273:305_ESVLC_20_GP_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6
	2. Row 3 to row 4. Before cost: 103. After cost: 98. Reward: 5				
[[[1 0 0 0 0] [2 0 0 0 0]	[3 10 0 0 0] [4 10 3 0 0]	[0 0 0 0 0]] [[2 0 0 0 0]	[5 10 0 0 0] [5 10 0 0 0] [		
	93:117_AUFRE_20_GP_X				
10:73_PHSFS_20_GP_X	93:117_AUFRE_20_GP_X		81:132_BRPNG_20_GP_L		
157:185_AEJEA_20_GP_M	93:117_AUFRE_20_GP_X	10:73_PHMNL_20_GP_L	320:348_INMAA_20_GP_H		273:305_ESVLC_20_GP_X
10:73_PHMNL_20_GP_H	93:117_AUFRE_20_GP_X	74:92_PKKHI_20_GP_X	320:348_INMAA_20_GP_H	10:73_PHMNL_20_GP_X	273:305_ESVLC_20_GP_X
74:92_PKKHI_20_GP_L	10:73_PHMNL_20_GP_L	10:73_PHMNL_20_GP_X	74:92_PKKHI_20_GP_X	720:744_INMAA_20_GP_H	273:305_ESVLC_20_GP_X
Row 1	Row 2	Row 3	Row 4	Row 5	Row 6

Difficulties and Extensions

Runtime:

Long runtime for enough exploration and convergence

Strict requirements from Operations team

Generalization:

Each slot as a new task

Different slots have different stating states

Extensions:

Running on cloud clusters

Meta-learning for generalization

Different formulations of the problem

Learning Points

Independent learning:

Independent research and experimentation

Mathematical appreciation:

A strong mathematical foundation helps deal with complexity

Operations research problems:

Exposure to industrial engineering workflows

Reinforcement learning knowledge:

Understand the evolution of this field

Understand how it is in some aspects similar to Supervised Learning

Understand the considerations made

Simulation, distributed computing, cloud computing

Smart Container Shuffling System Project

- **Algorithm:**

Proximal Policy Optimization: Actor-Critic

Linear Search (Gradient Descent):
simple and fast but in RL, if step size too big can fall off cliff, hurt training

Trust Region:
determine max step size to explore, local optimal point within trust region and resume search from there

start with initial guess, re-adjust dynamically (shrink if policy divergence gets large)

Proven: calculated optimal policy within trust region always better than old policy

KL-divergence:
difference between two distributions (repurpose to two policies)

Smart Container Shuffling System Project

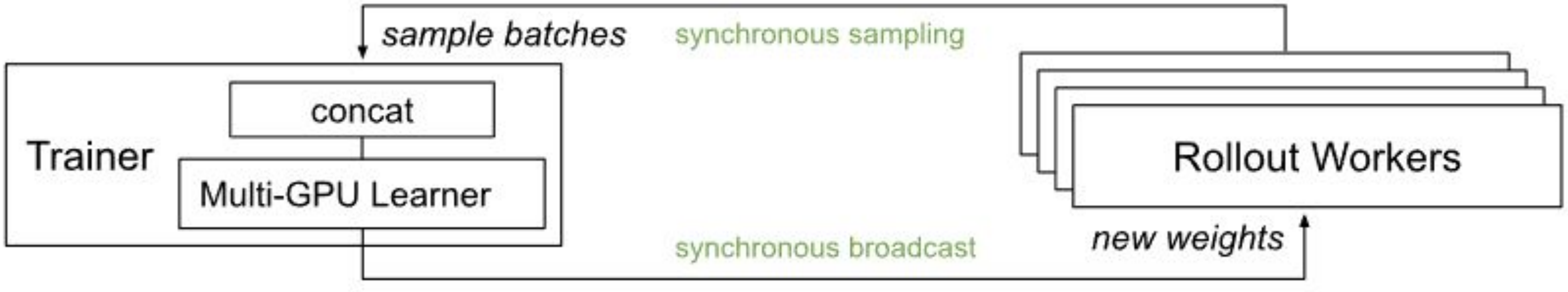
- **Framework:**

RLLib

Parallel training:
Actor: Python processes

Task: Remote function

Multiple environments



PPO architecture

Smart Container Shuffling System Project

- **RL Research:**

Study the fundamentals from literature and existing RL technologies / frameworks

- **Experimentation:**

Map container shuffling problem to RL using OpenAI Gym interface

Define agent / state / action / reward / episode termination

Experiment with different algorithms (Q-learning variants)

- **Increments:**

Problem: Overstow -> mix Pk + Sz + Cat -> mixWt

Difficulties: Non-convergence for difficult cases + determine termination + non-distributed framework (TF-Agents)

Learning: Action masking + state-of-the-art Rainbow DQN

Smart Container Shuffling System Project

- **Framework Switch:**

Experiment with distributed computing using Ray RLLib

Re-think observation representation and reward function, change algorithm (**Direct policy optimization**)

Success: Parallel training and convergence for 6-row slots

Difficulties: Observation representation and reward engineering (**dense** or **sparse** rewards)

Learning: How RL is trained in parallel in a cluster (**local** or **cloud**)

- **Implementation:**

Implement locally, treating each slot as a new game

Implement UI

Experiment with 10-row slots, multiple slots

Difficulties: Slow convergence for larger state space + slow to train for each slot + generalization + lack of industrial examples

Min	Max	Avg
2	5	3

Smart Container Shuffling System Project

- **Further Research:**

Study AnyLogic simulation and application of RL to existing simulation

Study AWS reinforcement learning examples

- **Model-Agnostic Meta-Learning:**

Trains a model on a variety of tasks such that it can learn a new task with only a **small** number of **training samples**

Goal:

Quickly acquire a policy for a new test task using a small out of experience

New task might involve achieving a **new goal** or succeeding a **previously trained goal** in a **new environment**

Application:

Design incremental examples for the slot and train level by level

Smart Container Shuffling System Project

- **Further Directions:**
  - **Sparse reward:**

    Intrinsic reward (Curiosity)
  - **Curriculum Learning:**

    Incremental exposure to environments of different difficulties to generalize learning
  - **Multi-Agent Learning:**

    Training multiple agents at once, each having a separate policy or share same policy
  - **Simulation:**

    Using Unity Engine or AnyLogic
  - **Cloud Computing:**

    Scale to the Cloud from local computer

Moving Further: Case Study

- **Simple Product Delivery:**

3 manufacturers, each with 3 vehicles 15 distributors, demand 500 to 1000 goods every 1 or 2 days

RL agent: which manufacturer can fulfill order most quickly

Manufacturer:

Not enough inventory: need more time

Delivery time: distance despite having enough inventory

Metrics:

Select manufacturer to fulfill order, minimize **wait times** and **distance driven**

Best scenario:

Nearest manufacturer to an ordering distributor has enough inventory to complete order
Min wait times for both production and travel

- **Observation:**

Stock levels available trucks order amounts for each distributor

Moving Further: Case Study

- **Metrics:**

Goal: minimize delivery delays

Track:

Average wait times

Average kilometers traveled

- **Actions:**

15 decision points (each of the 15 distributors order products) with 3 possible actions (which of 3 manufacturers)

- **Event trigger:**

Once per day

- **Done:**

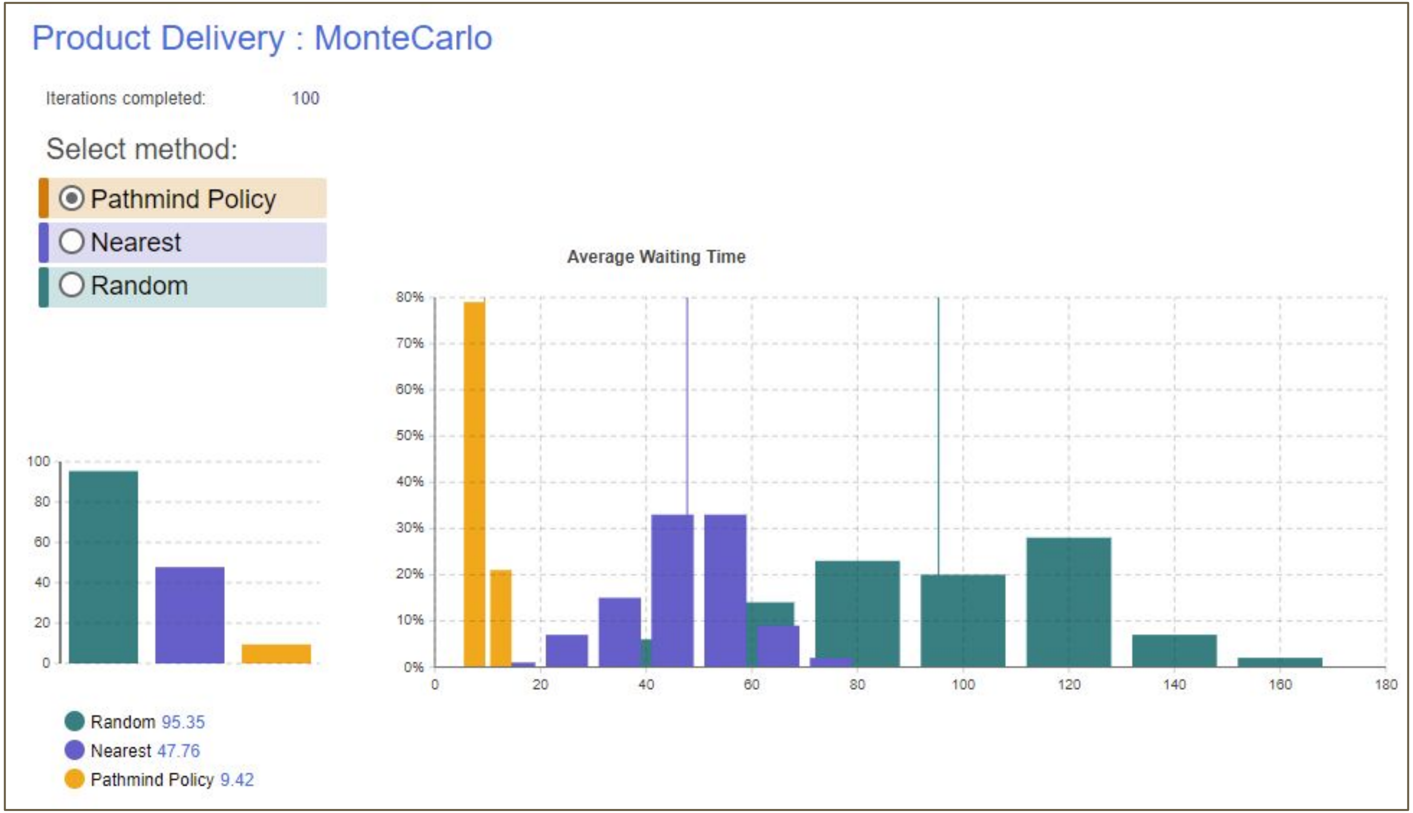
Set to run for four months

- **Reward:**

Minimize average wait times

Moving Further: Case Study

Moving Further: Case Study



Moving Further: Case Study

- **Multi-Echelon Product Delivery:**

2 manufacturing centers 4 distributors 8 retailers

Consumer demand fluctuates randomly, time for new inventory depends on which manufacturer or distributor

Objective:

Maximize profit by ensuring **all entities** have **sufficient inventory** to fulfill demand at **any given moment**

If not enough inventory -> sales are lost

If excess inventory -> storage costs

- **Observation:**

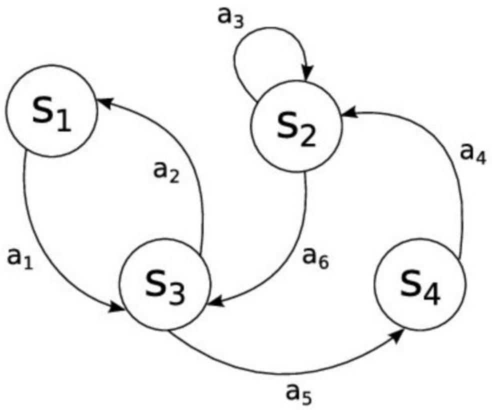
Manufacturer: stock and order backlog

Distributor: stocker, order backlog, expected deliveries

Retailer: stock, order backlog, expected deliveries

Time: day of week and day of year

In-depth



- **Policy:**

A function π determines how agent behaves in any given time: π(S2) -> a6

E.g. A lookup table / simple function / entire search process

Often stochastic: **probability distribution**

- **Reward Signal:**

Primary basis for altering policy -> **reward engineering ?**

Depends on: state + action -> **numerical** high/low reward

- **Value Function:**

Long-run evaluation for current **state**: expected total future accumulative reward from current state

A state might give low immediate reward but if state-value function is **high** -> still go to this state

In-depth

- **Model-free vs Model-based:**

Explicit trial-and-error learners vs predicting environment behavior = (state, action) -> (next state, next reward)

- **Exploration vs Exploitation:**

Constantly takes best expected action or explore some new actions (**ε-greedy**)

Exploration thus can help find higher reward actions

- **Episodic / Continuous tasks:**

Agent maximise expected return E(Gt) where Gt = R(t + 1) + R(t + 2) + ... + R(T)

Requires **expected discounted return** $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{0}^{\infty} \gamma^k R_{t+k+1}$

Key idea for **Bellman equation**:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

In-depth

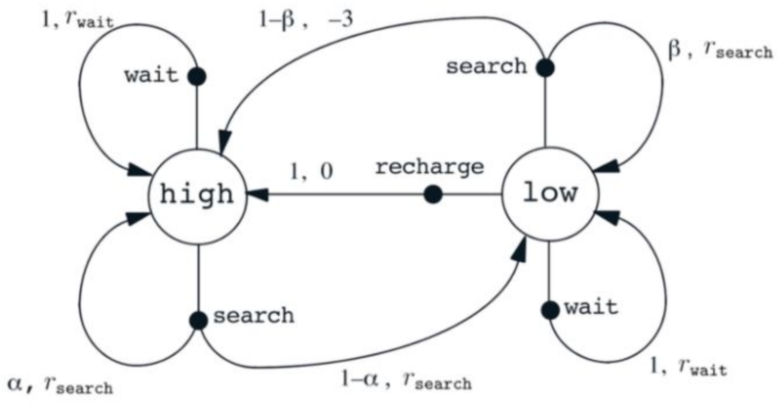
- **Markov Property:**

A state representation that summarizes all relevant information of the past -> predict future: **Markov** (e.g. TicTacToe state)

Dynamic function: $$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

Markov Decision Process (MDP): a RL problem that satisfies Markov property (finite MDP)

- **Transition Diagram:**



An example of a transition diagram [ref]

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

In-depth

- **State-value function for policy π:**

How good for agent to be in a given state.

Value of a state **S** under a policy **π** :

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Expected return when starting in S and following **π** afterwards

- **Action-value function for policy π:**

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Expected return when starting in S, taking action A and following policy **π** afterwards

- **Estimation from experience:**

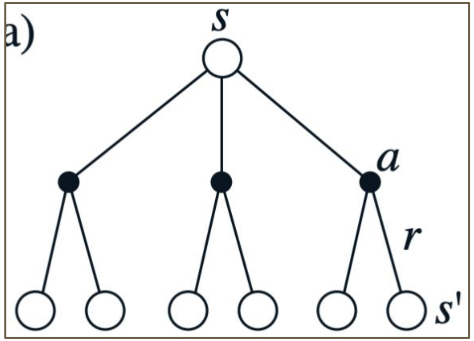
Continuously interacts with env + keeps an **average** of actual returns following each **state / action in a state**

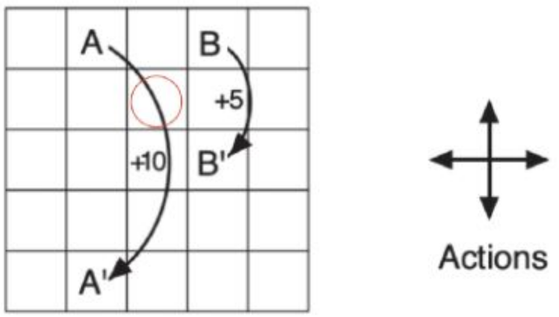
As interaction -> **infinity**: convergence to value functions

In-depth

- **Bellman Equation:** Estimate value of states using successor states

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]
\end{aligned}
$$





3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

0 + 0.9* [(0.25 * 4.4) + (0.25*1.9) + (0.25*0.7) + (0.25*3.0)] = 2.25

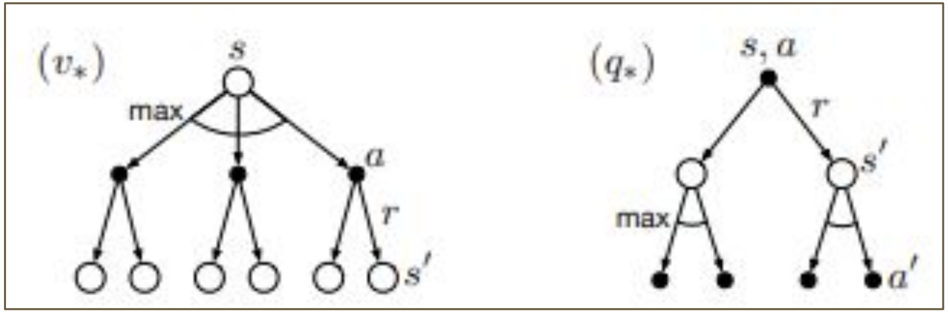
In-depth

- **Optimal Value Functions:**

Bellman optimality equations: solve for the optimal value of states

$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a)\Big[r + \gamma v_*(s')\Big]
\end{aligned}
$$

$$
\begin{aligned}
q_*(s, a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \Bigm| S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s', r | s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big],
\end{aligned}
$$



Can find **optimal policy** once we found optimal value functions **v*** and **q***

However, still currently intractable to solve for every state

In-depth

- **Policy Evaluation:**

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big]$$

Compute state-value function Vπ for an arbitrary policy π.

For each iteration, back up value of every state once to produce new approximate V(k + 1)

- **Policy Improvement:**

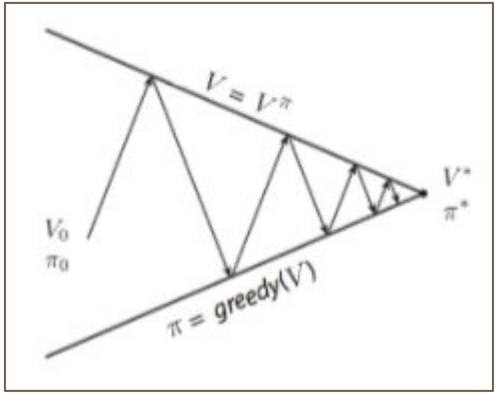
$$\pi'(s) = \arg\max_a q_\pi(s, a)$$

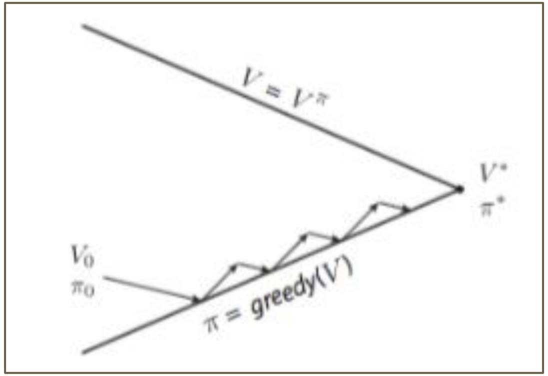
Taking an old policy, make a new & improved one by selecting greedy actions w.r.t the value function of original policy

Works for both deterministic and stochastic policy π

- **Policy Iteration:**

- **Value Iteration**





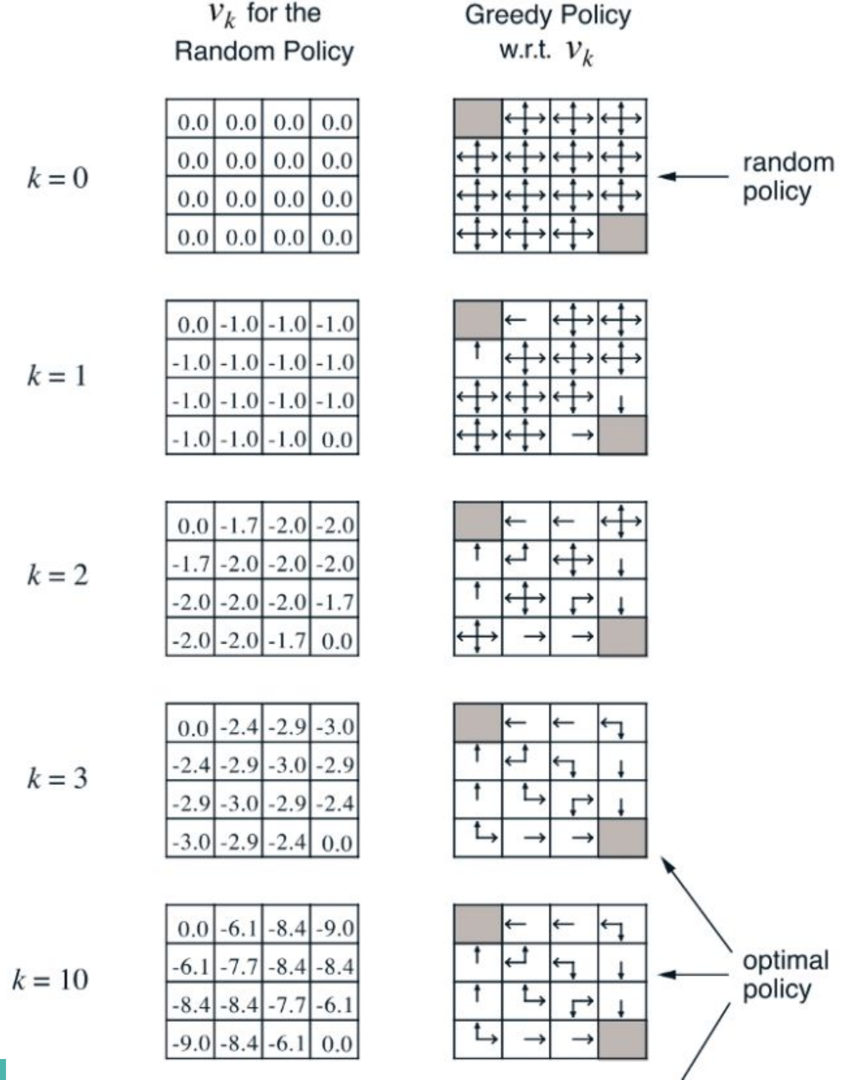
In-depth

- **Generalized Policy Iteration (Dynamic Programming):**

Randomly initialize our value function estimates of every state + start with a random policy

Evaluate value of every state with this policy

Update policy with greedy action choices w.r.t value functions (take action that moves to highest value state)



In-depth

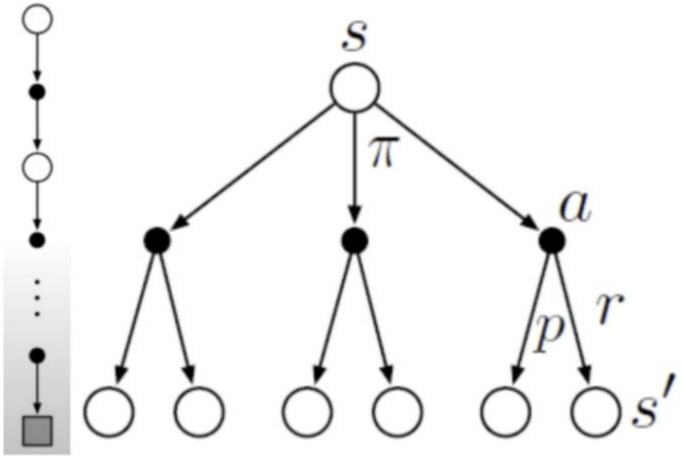
- ## Monte Carlo Method (Model-Free Learning):

No longer have complete knowledge of the environment: no transition function $\mathbf{p(s', r \mid s, a)}$

Estimate value functions and find optimal policies based on experience in **episodic** tasks

Randomized algo: sample **states + actions + rewards** from interaction -> use **average sample returns** to update **s**

Key idea:
More interaction + more returns, average converges to expected value



- ## Monte Carlo Estimation of Action Values:

Since no model, estimate action values q* is better than state values v*

- ## Monte Carlo Control:

Approximate optimal policies

Follow generalized policy iteration: approx policy + approx value function

In-depth

- **On-policy Learning:**

Try to evaluate and improve the policy we have

- **Off-policy Learning:**

2 policies: evaluate and improve one + use the other for directions

A **target policy** π tries to behave optimally

A **behavior policy b** for exploration by generating episodes to update target policy

Estimate V$\boldsymbol{\pi}$ or q$\boldsymbol{\pi}$

In-depth

- **Temporal-Difference (Model-Free Learning):**

Combines Monte Carlo and Dynamic Programming

Update value estimates based partially on other estimates, without going through entire episode (**bootstrap**)

- **TD Prediction:**

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

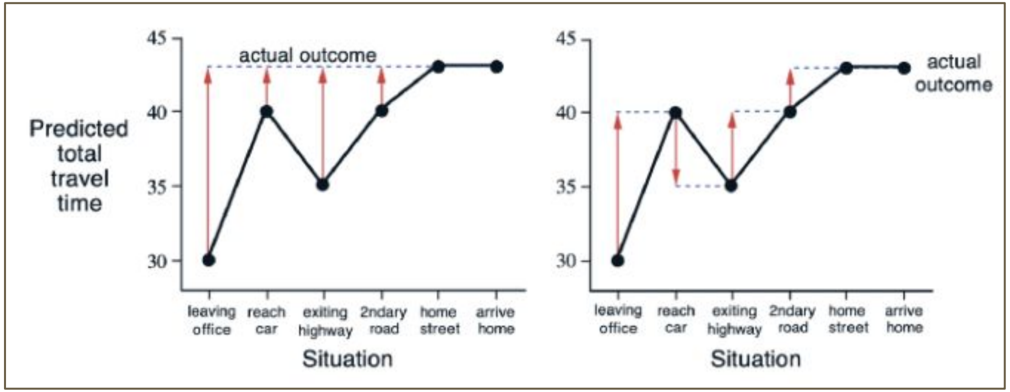
MC update rule [ref]

Update just at next time step **TD(0)** or **one-step TD**

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

TD update rule [ref]

Bootstrapping: an estimate updates based on another estimate



In-depth

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

- **SARSA (On-Policy TD Control):**

Sarsa

Instead of state-value, learn action-value function $q_\pi(s, a)$

SARSA: $(\mathbf{S}_t, \mathbf{A}_t, \mathbf{R}_{t+1}, \mathbf{S}_{t+1}, \mathbf{A}_{t+1})$

Update at every time step

- **Q-Learning (Off-Policy TD Control):**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

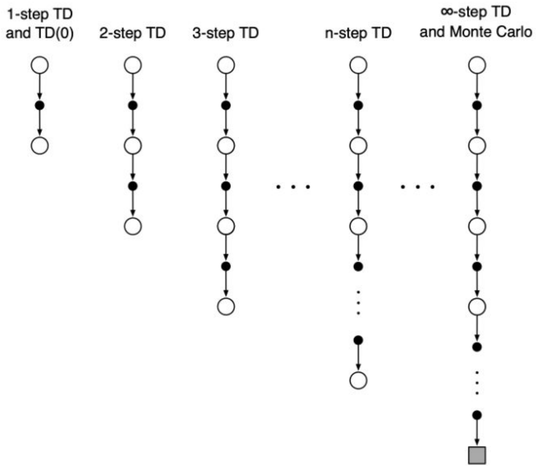
Similar to SARSA, except takes **max** over next state-action pairs

The learned action-value **Q** directly approximates **q*** the optimal action-value, **independent** of the **policy** being followed

In-depth

- **N-step Bootstrapping:**

Unifies TD and MC methods - something in between



- **N-step SARSA:**

Estimates action-value