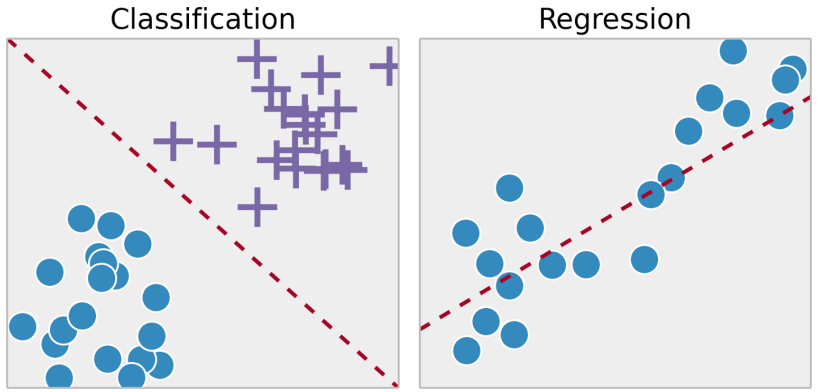
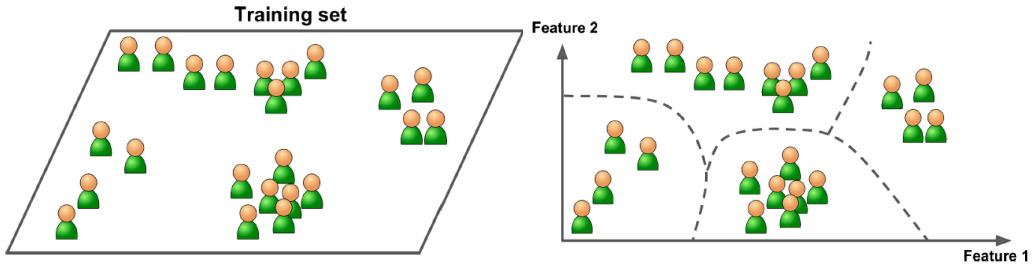
Reinforcement Learning in Transportation

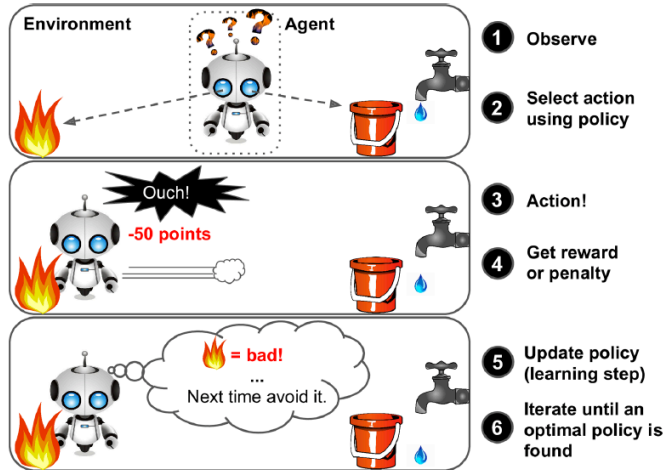
Peyman Noursalehi

Three applications of ML

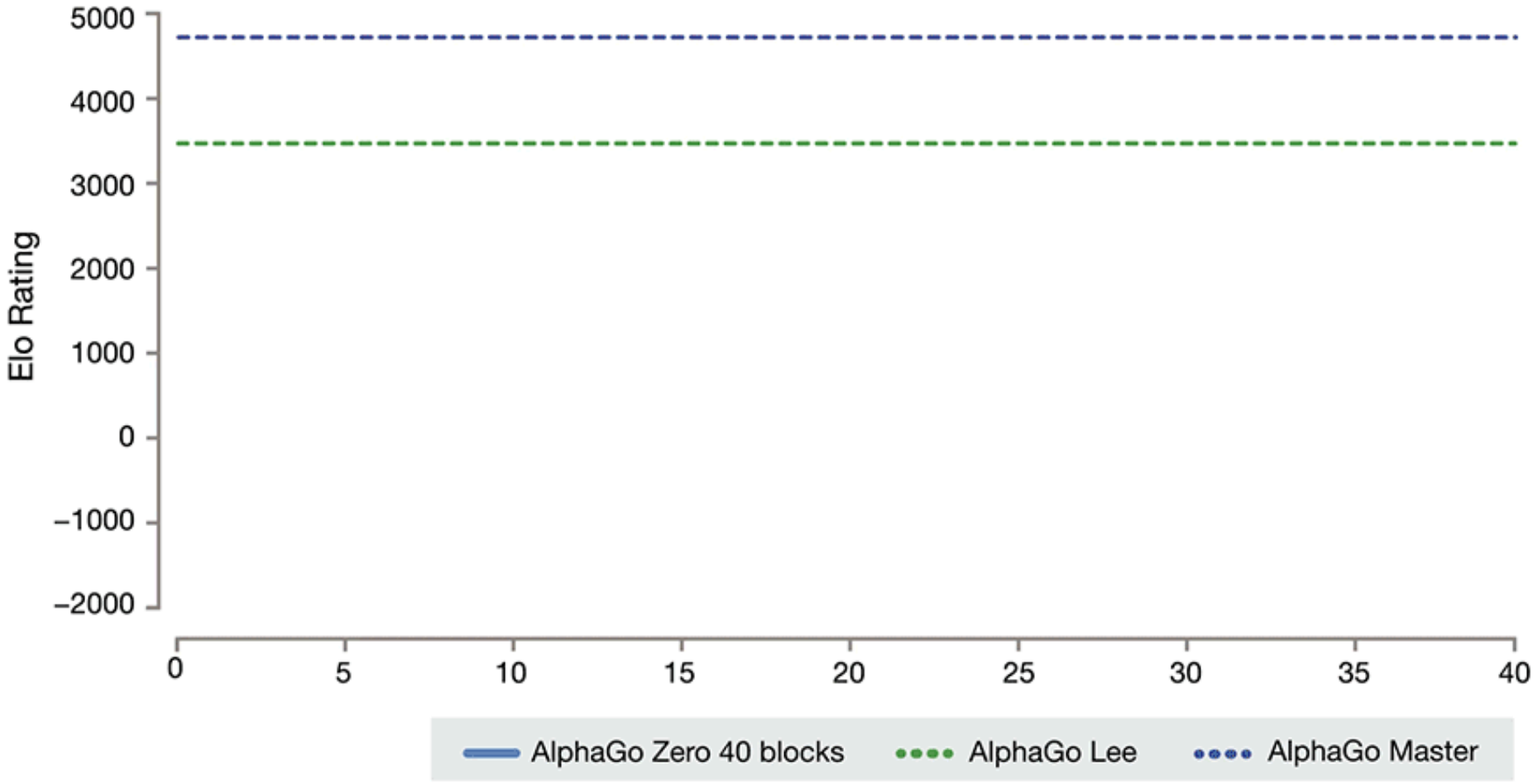
- Predictive
  - Forecast certain quantities based on features
  - Demand, traffic flow, ETA, etc.
- Descriptive
  - Characterize key properties of a dataset
  - PCA, LDA, Kmeans
- Prescriptive
  - What's the best action to take?
  - Optimization task!



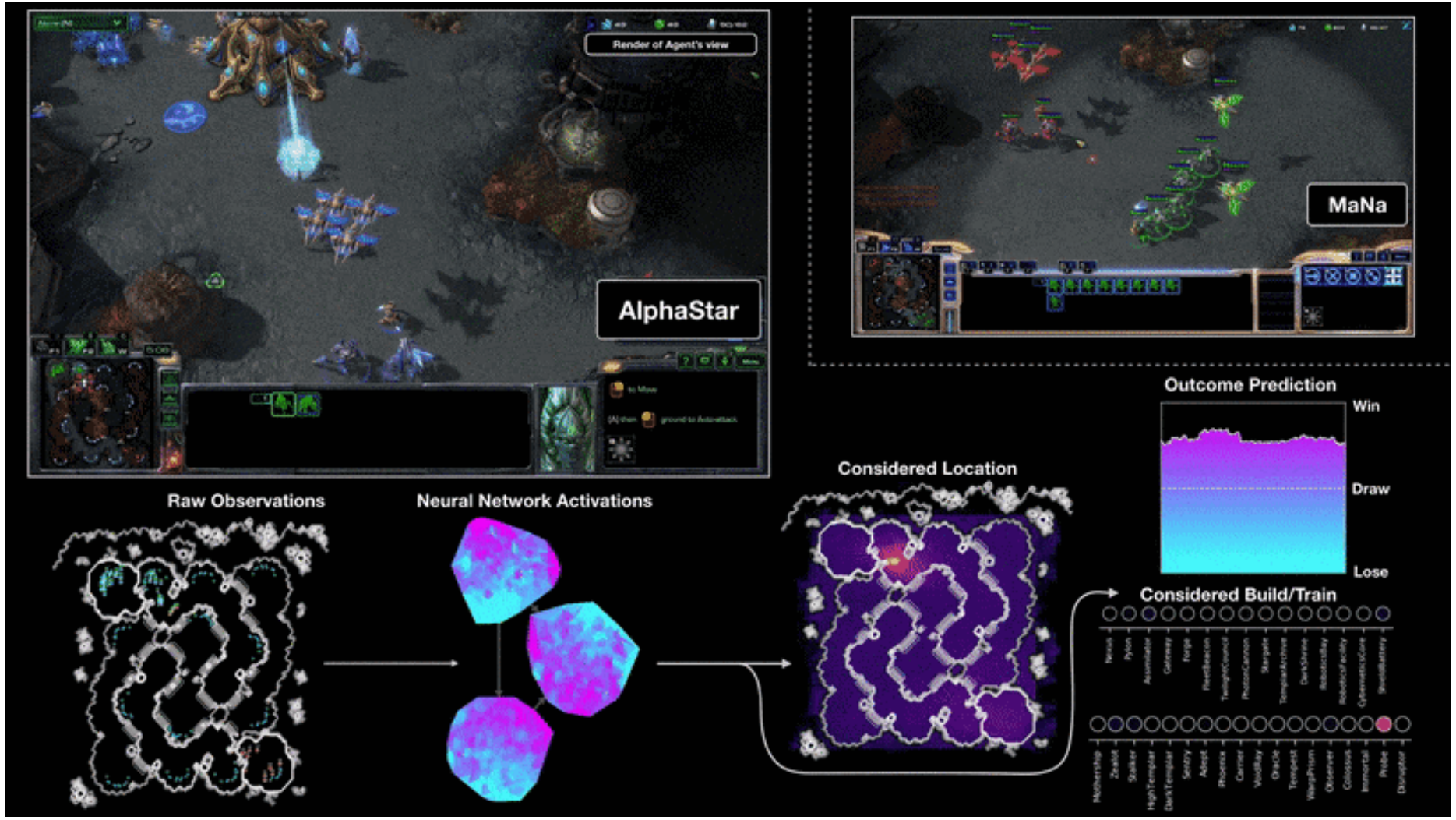




AlphaGO


5000
4000
3000
2000
1000
0
−1000
−2000
Elo Rating
0
5
10
15
20
25
30
35
40
AlphaGo Zero 40 blocks
AlphaGo Lee
AlphaGo Master

AlphaStar


Render of Agent's view
AlphaStar
MaNa
Outcome Prediction
Win
Draw
Lose
Considered Location
Raw Observations
Neural Network Activations
Considered Build/Train

Reinforcement Learning

Learning to make good sequences of decisions
under uncertainty

Reinforcement Learning

Learning to make good sequences of decisions under uncertainty

Reinforcement Learning

Learning to make good sequences of decisions
under uncertainty

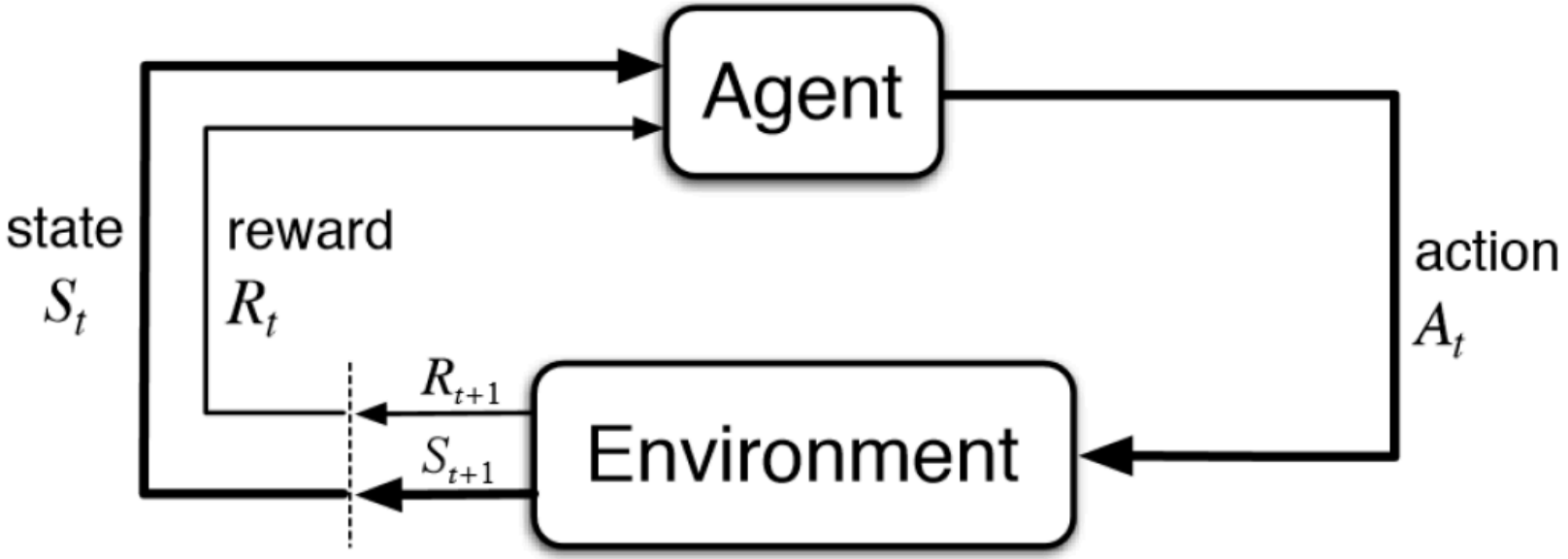
Reinforcement Learning

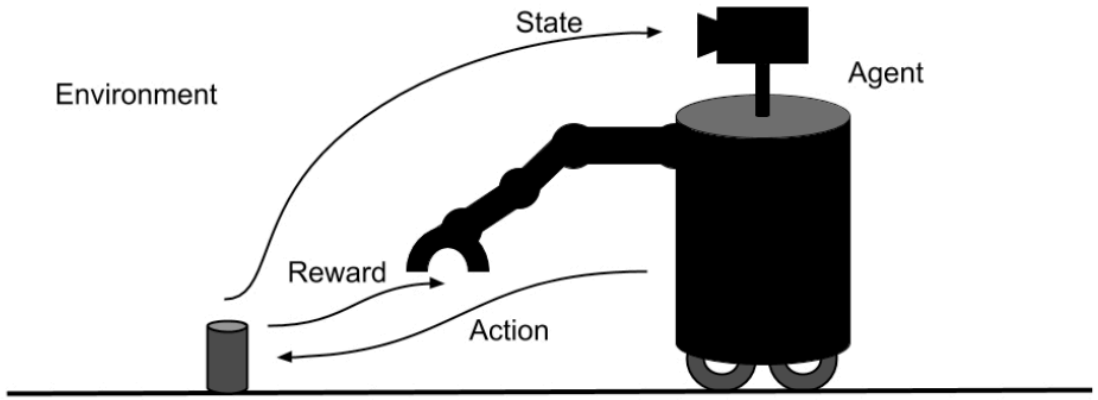
Learning to make good sequences of decisions
under uncertainty

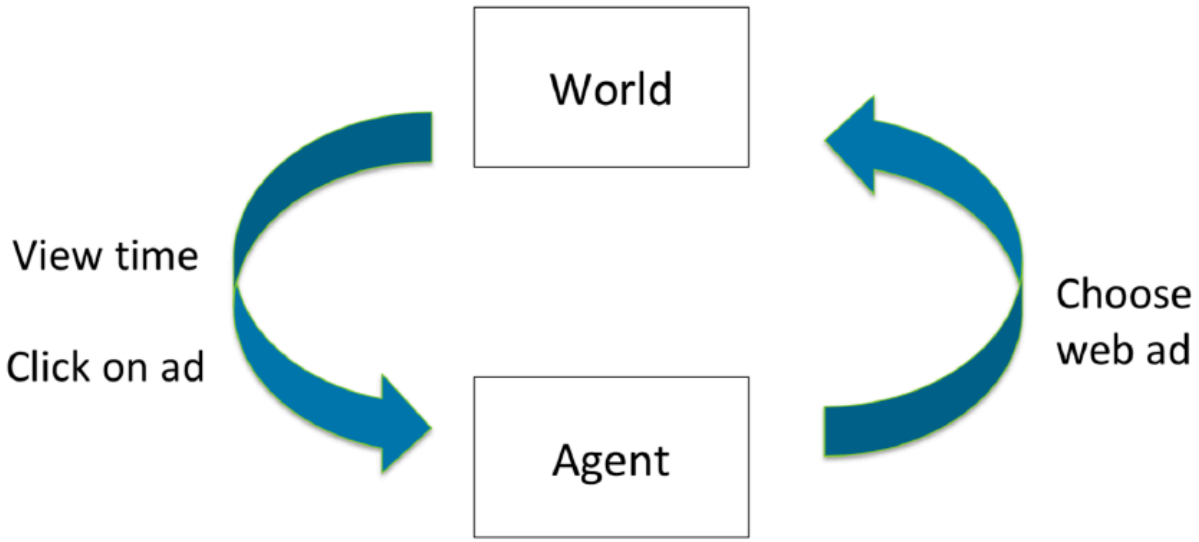
Reinforcement Learning

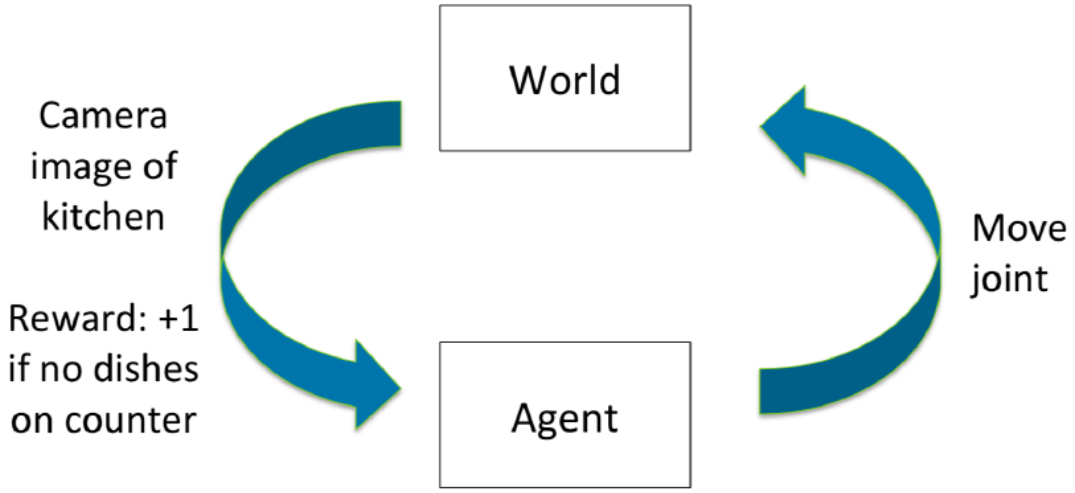
Learning to make good sequences of decisions
under uncertainty

Agent-environment interaction









Source: Emma Brunskill

Agent-environment interaction

- The decision maker is called an agent
    - Takes an action, affects the environment
- Everything else is part of the environment
    - Any changes that are not due to the actions of the agent
- Single agent vs multi agent

Episode

- The agent repeatedly interacts with the environment
  - Learn by trial and error
  - Ends when the goal is achieved (Episodic tasks)
    - Win/lose a game
  - Goes on indefinitely
    - Control a robot over its lifetime

- Each episode (or trajectory) consists of

$$S_0, A_0, R_0, S_1, A_1, R_1, \ldots, S_n, A_n, R_n$$

- This sequence forms a stochastic process

Objective

- Maximize the expected reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N} = \sum_{k=0}^{N-1} \gamma^k R_{t+k+1}$$

- γ is the discount factor

Markov decision processes (MDP)

- A Markov decision process is a tuple (S, A, R, P, γ) where
- S is the set of states (discrete or continuous)
  - E.g.,, agent's location, battery charge
  - Fully observable environment: $o_t = s_t$
  - Partially observable: $o_t = \phi(s_t)$
- A is the set of actions (discrete or continuous)
  - E.g., Which zone to go to next, what should the surge multiplier be
- R is the reward
- P is the transition probability
- γ is the discount factor

Markov decision processes (MDP)

- A Markov decision process is a tuple (S, A, R, P, γ) where

- Markov Property:

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

- State representation has big implications for:
  - Computational complexity
  - Data required
  - Resulting performance

Main components of a RL algorithm

- Model: how the world changes in response to the agent's actions
- Policy: a mapping from states to probabilities of selecting each possible action
  - E.g., if battery is low, recharge
- Value function: expected future rewards from being in a state and/or action when following a particular policy

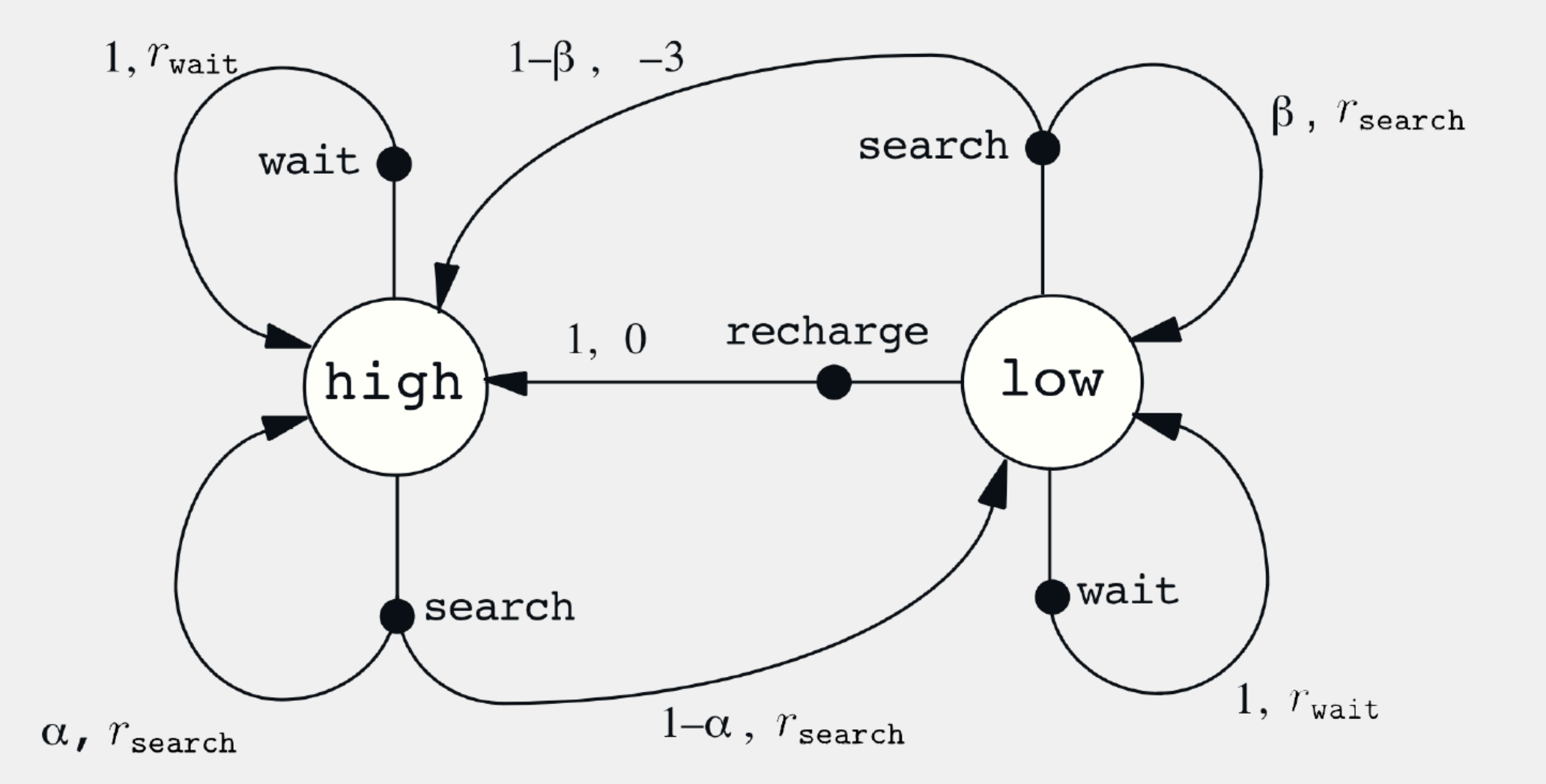
Categorizing RL algorithms

- Value based
  - No policy (implicit)
  - Value function
- Policy based
  - Policy
  - No value function
- Actor critic
  - Policy
  - Value function

- Model based
  - Transition and reward model
- Model free
  - No transition and no reward model (implicit)

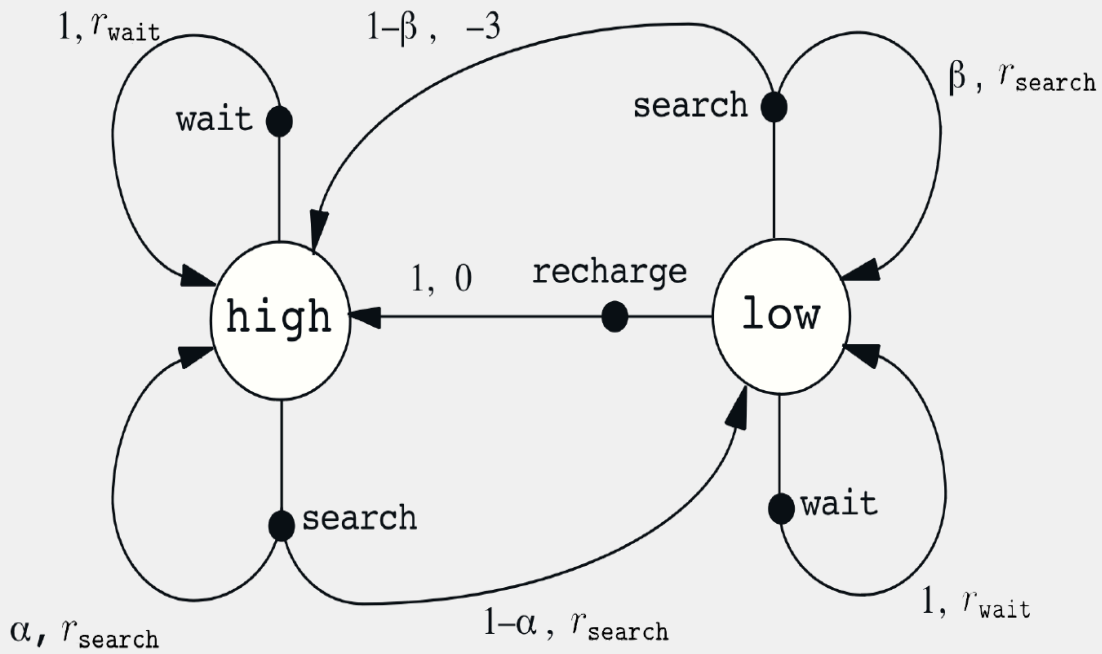
Example: cleaning robot (Model-based)

Example: cleaning robot



Model

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



Value functions

- State-value function of a policy

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{N} \gamma^k R_{t+k+1}|S_t = s]$$

- Action-value function of a policy

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{N} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

Value functions

- We can estimate the value functions from experience
    - Maintain an average per state (or state-action)
- What if there are millions of states?
    - Store a parameterized function of the values functions

Bellman equations

- State-value function of a policy

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{N} \gamma^k R_{t+k+1}|S_t = s]$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N}$$

Bellman equations

- State-value function of a policy

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{N} \gamma^k R_{t+k+1}|S_t = s]$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N}$$

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] :$$

Bellman equations

- State-value function of a policy

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{N} \gamma^k R_{t+k+1}|S_t = s]$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N}$$

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] =$$

$$\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']]$$

Bellman equations

- State-value function of a policy

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{N} \gamma^k R_{t+k+1}|S_t = s]$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{N-1} R_{t+N}$$

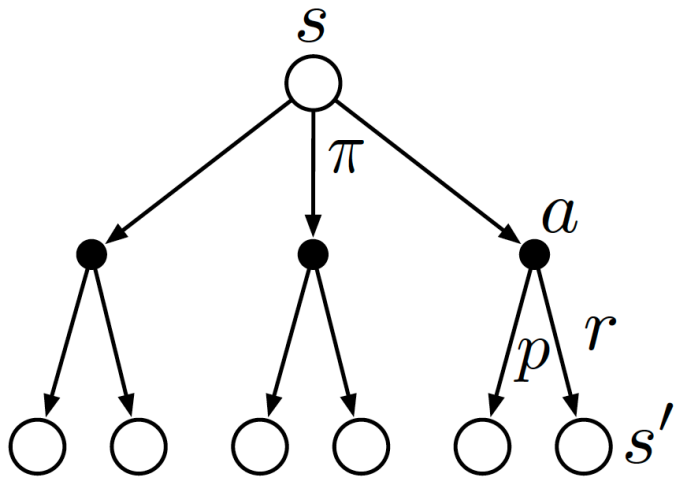
$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] =$$

$$\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_\pi[G_{t+1}|S_{t+1} = s']] =$$

$$\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Bellman equations

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

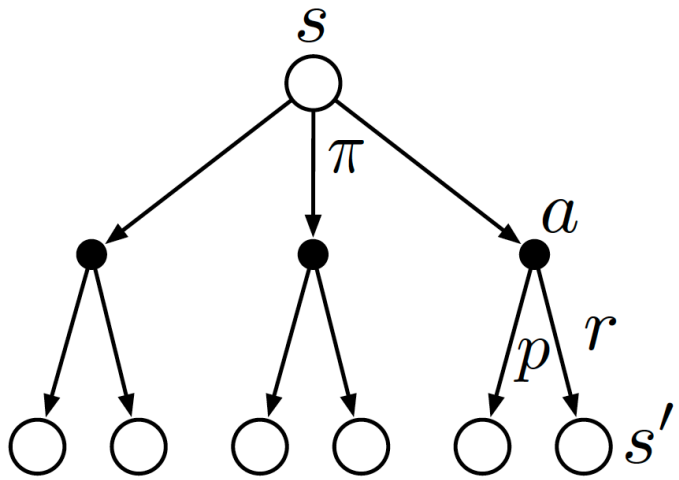


Backup diagram for v_π

Bellman equations

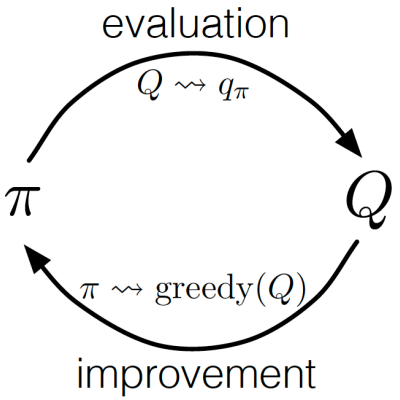
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Model-based!



Backup diagram for v_π

Value iteration



- Iteratively update the state-value function

$$v_{k+1}(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(s_{t+1}) | S_t = s, A_t = a\right] =$$

$$= \max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v_k(s')\right]$$

- Act greedy according to the new value function

$$\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$$

What if we don’t have a model?

- Learn purely from experience
- Model-free methods
- We focus on learning state-action, rather than state value function
- Two categories:
  - Monte Carlo method
  - Temporal-difference method

Monte Carlo Methods

- Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns.

$$v_\pi(s) = E_\pi[\sum_t \gamma^t r_t] = \frac{1}{N}\sum_{k=1}^{N}[\sum_t \gamma^t r_t^k]$$

- Only on the completion of an episode are value estimates and policies changed
- Monte Carlo methods can thus be incremental in an episode-by episode sense, but not in a step-by-step (online) sense
- Slow, but eventually converge

MC policy evaluation

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Temporal Difference (TD)

- TD update estimates based in part on other learned estimates, without waiting for a final outcome

- Monte Carlo

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

- TD (0)

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

MC vs TD

Monte Carlo

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

Temporal-difference

TD(0)

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

On-policy vs Off-policy

- On-policy methods attempt to evaluate or improve the policy that is used to make decisions

- Off-policy methods evaluate or improve a policy different from that used to generate the data

Q-learning

- One of the earliest breakthroughs in reinforcement learning (1989)

- Q, directly approximates Q*, the optimal action-value function, independent of the policy being followed

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

Updated Q-value

Current Q-value

Learning rate

Observed reward

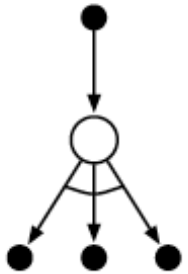
Discount factor

Q-learning

- One of the earliest breakthroughs in reinforcement learning (1989)

- Q, directly approximates $Q*$, the optimal action-value function, independent of the policy being followed

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



Q-learning

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Exploration vs exploitation

- If an agent always chooses the action with the highest value then it is exploiting

Exploration vs exploitation

- If an agent always chooses the action with the highest value then it is exploiting
  - Stochasticity of the environment, no model
  - Q function might underestimate the value of a state-action pair
  - Leads to suboptimal results

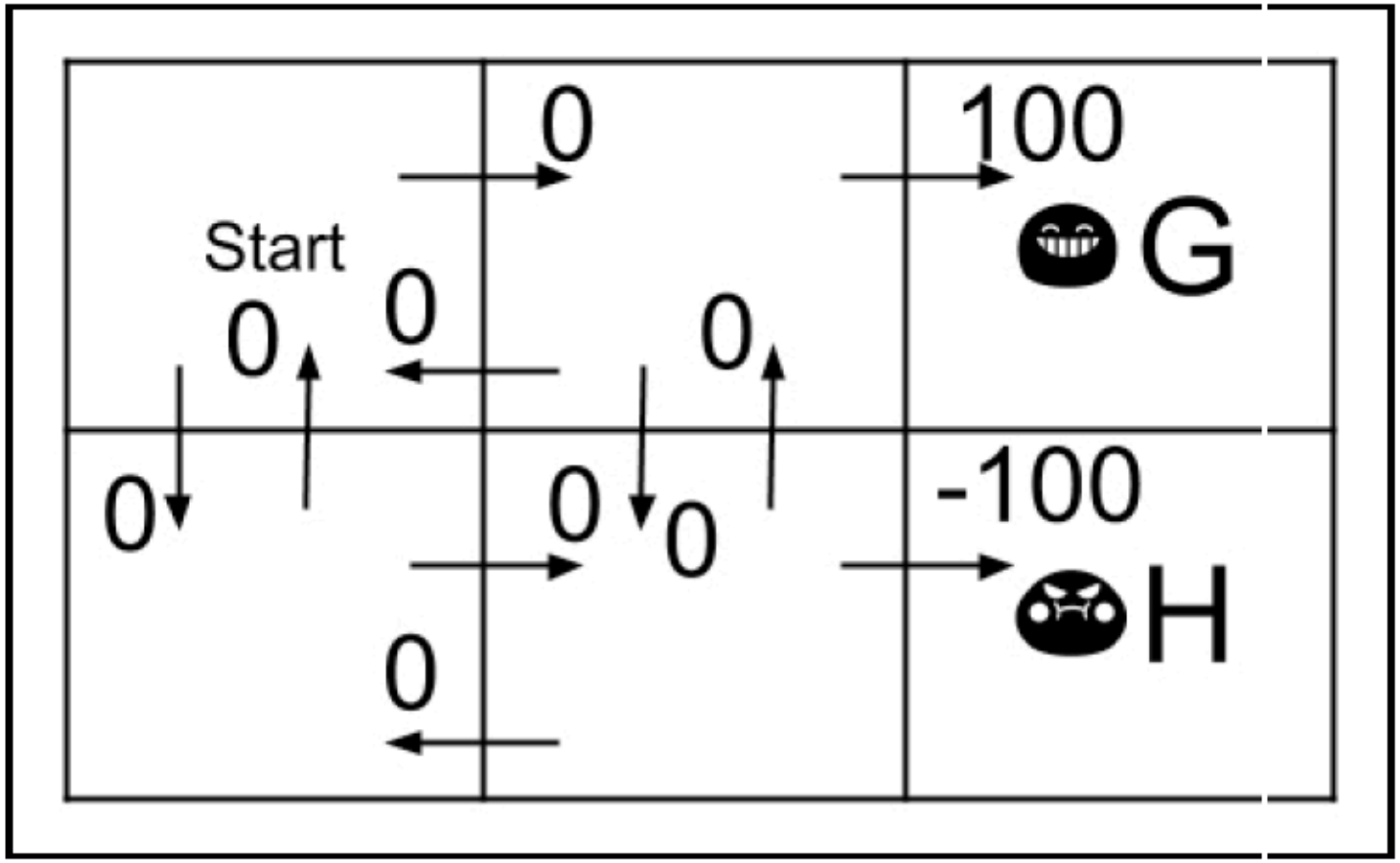
Exploration vs exploitation

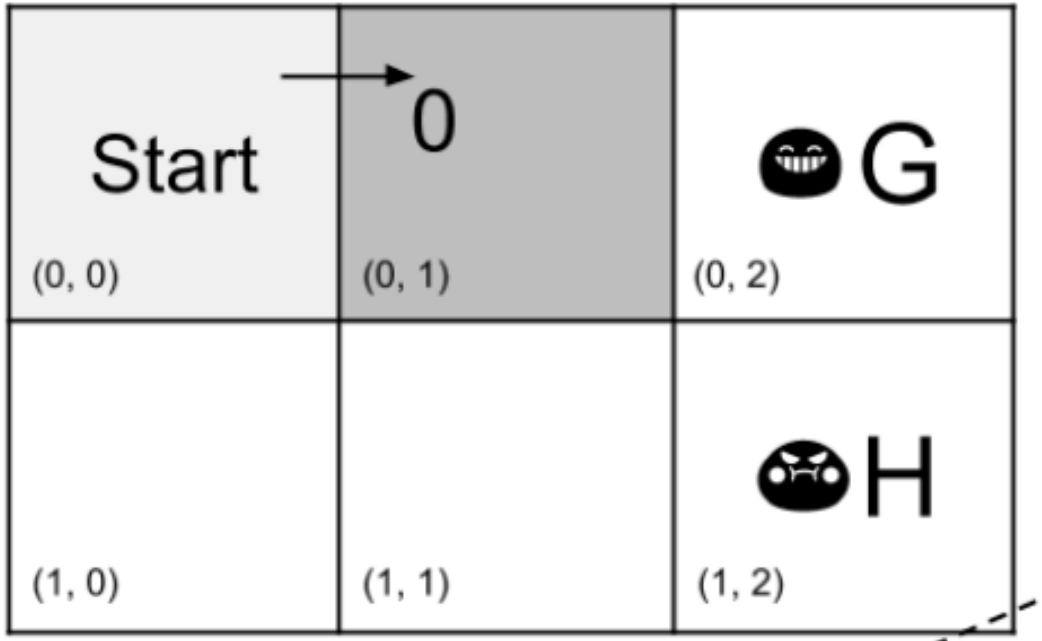
- If an agent always chooses the action with the highest value then it is exploiting
  - Stochasticity of the environment, no model
  - Q function might underestimate the value of a state-action pair
  - Leads to suboptimal results
- Need to explore states and actions that may not look attractive, because we have not visited them often enough

With probability 1 - ϵ choose the greedy action

With probability ϵ choose an action at random

Example: a simple, deterministic environment





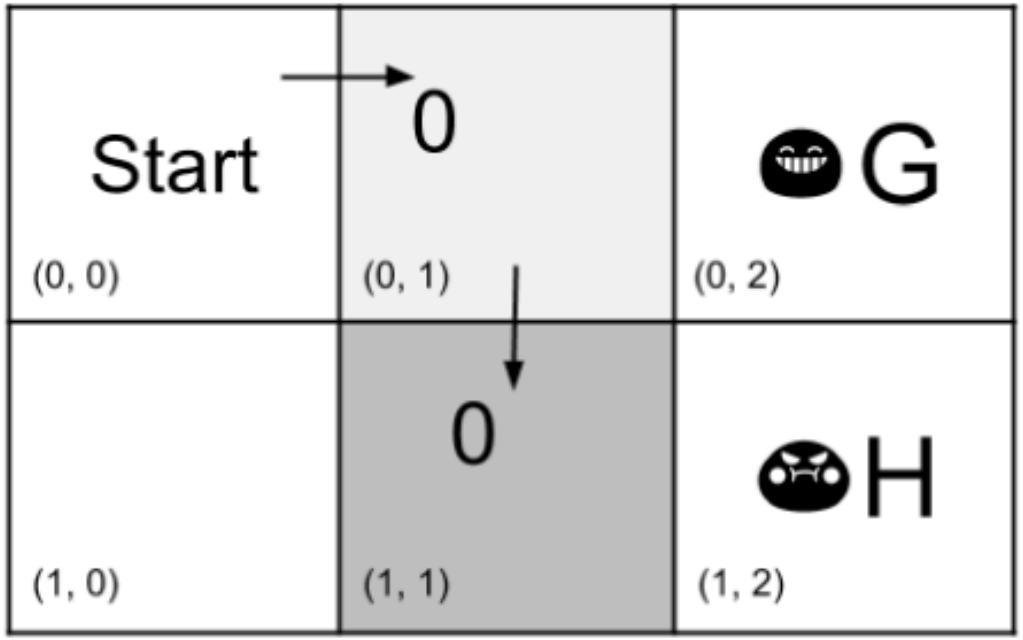
Mode: Exploration

Action / State	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	0	0
(1,2)	0	0	0	0

Q Table

$Q((0,0),\rightarrow) = \text{reward} + \gamma * \max_{a'} Q((0,1), a')$

$Q((0,0),\rightarrow) = 0 + 0.9 * \max(0, 0, 0, 0) = 0$



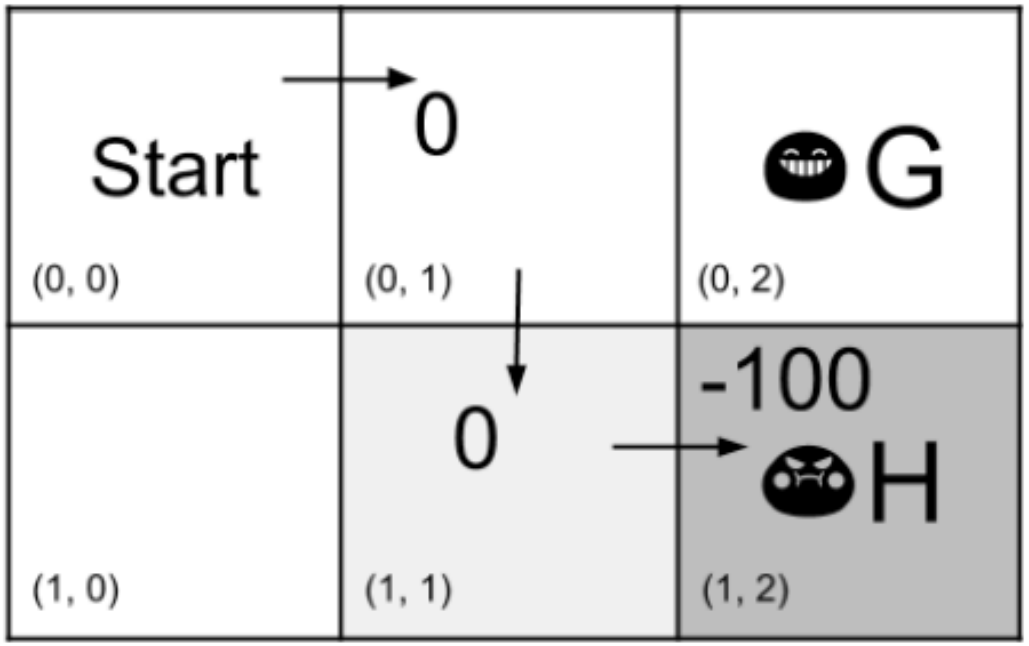
Mode: Exploration

State \ Action	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	0	0
(1,2)	0	0	0	0

Q Table

$Q((0,1),\downarrow) = \text{reward} + \gamma * \max_{a'} Q((1,1), a')$

$Q((0,1),\downarrow) = 0 + 0.9 * \max(0, 0, 0, 0) = 0$



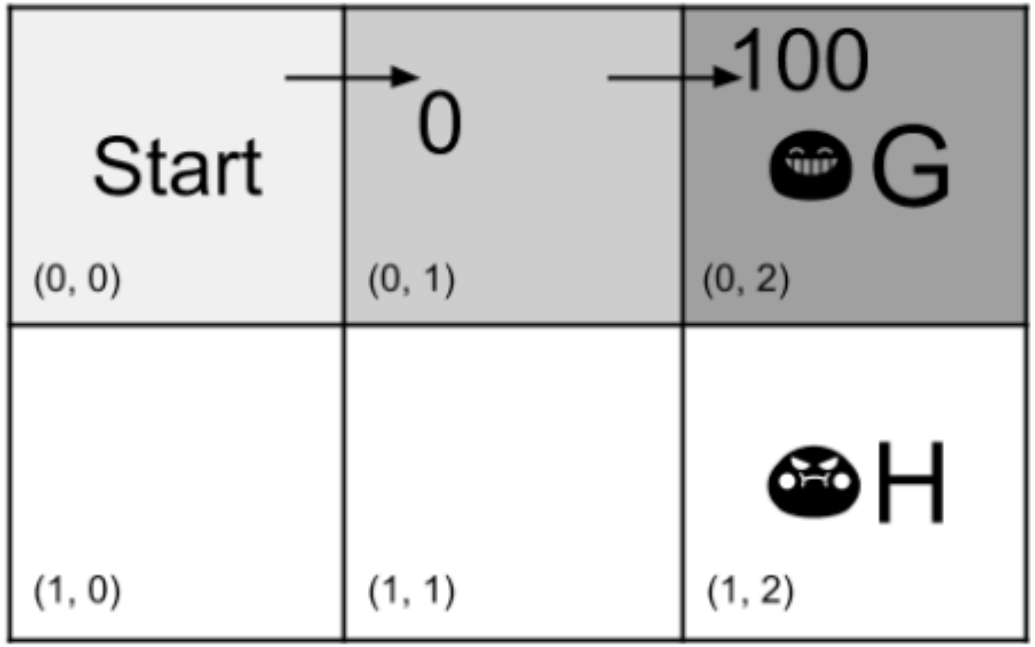
Mode: Exploration

State \ Action	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

$Q((1,1),\rightarrow) = \text{reward} + \gamma * \max_{a'} Q((1,2), a')$

$Q((1,1),\rightarrow) = -100 + 0.9 * \max(0, 0, 0, 0) = -100$

Q Table



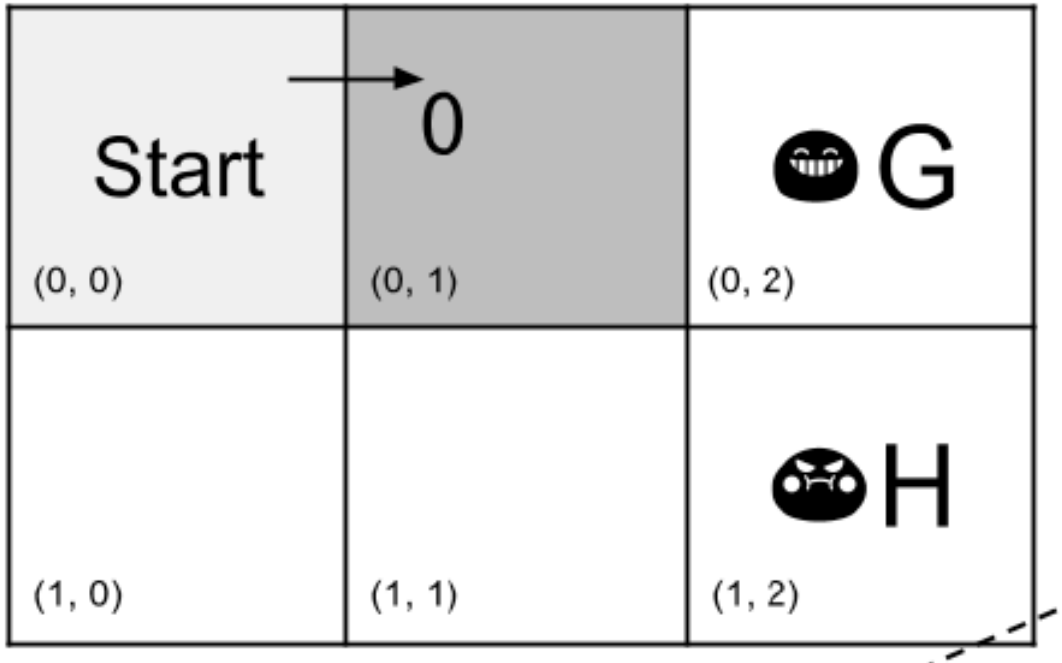
Mode: Exploration

State \ Action	←	↓	→	↑
(0,0)	0	0	0	0
(0,1)	0	0	100	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table

$Q((0,1),\rightarrow) = \text{reward} + \gamma * \max_{a'} Q((0,2), a')$

$Q((0,1),\rightarrow) = 100 + 0.9 * \max(0, 0, 0, 0) = 100$



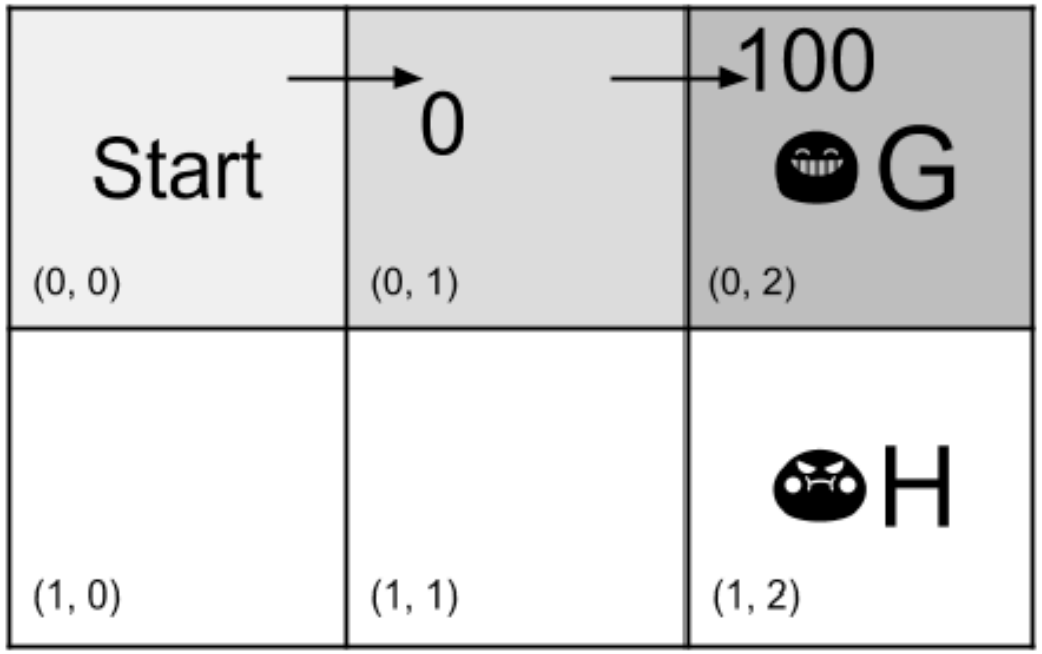
Mode: Exploration

State \ Action	←	↓	→	↑
(0,0)	0	0	90	0
(0,1)	0	0	100	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table

$Q((0,0),\rightarrow) = \text{reward} + \gamma * \max_{a'} Q((0,1), a')$

$Q((0,0),\rightarrow) = 0 + 0.9 * \max(0, 0, 0, 100) = 90$



Mode: Exploitation

$\max_{a'} Q((0,0),a') = 90 \rightarrow$ with s'=(0,1)

$\max_{a'} Q((0,1),a') = 100 \rightarrow$ with s'=(0,2) or Goal

State \ Action	←	↓	→	↑
(0,0)	0	0	90	0
(0,1)	0	0	100	0
(0,2)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	-100	0
(1,2)	0	0	0	0

Q Table

Optimize taxi driving strategies based on reinforcement learning

Yong Gao[a], Dan Jiang[a] and Yan Xu[b]

[a]Institute of Remote Sensing and Geographic Information System, Peking University, Beijing, China; [b]Spatial Sciences Institute, University of Southern California, Los Angeles, CA, USA

Problem description

- Goal:
  - Find the optimal driving strategy with maximum long-term profits for cab drivers

- Data:
  - GPS traces from 12,000 taxicabs in Beijing (20% of the total taxis)
  - Minute-by-minute data, including:
    - location
    - Status (e.g., vacant, occupied, parking)
  - 371,189 valid trajectories

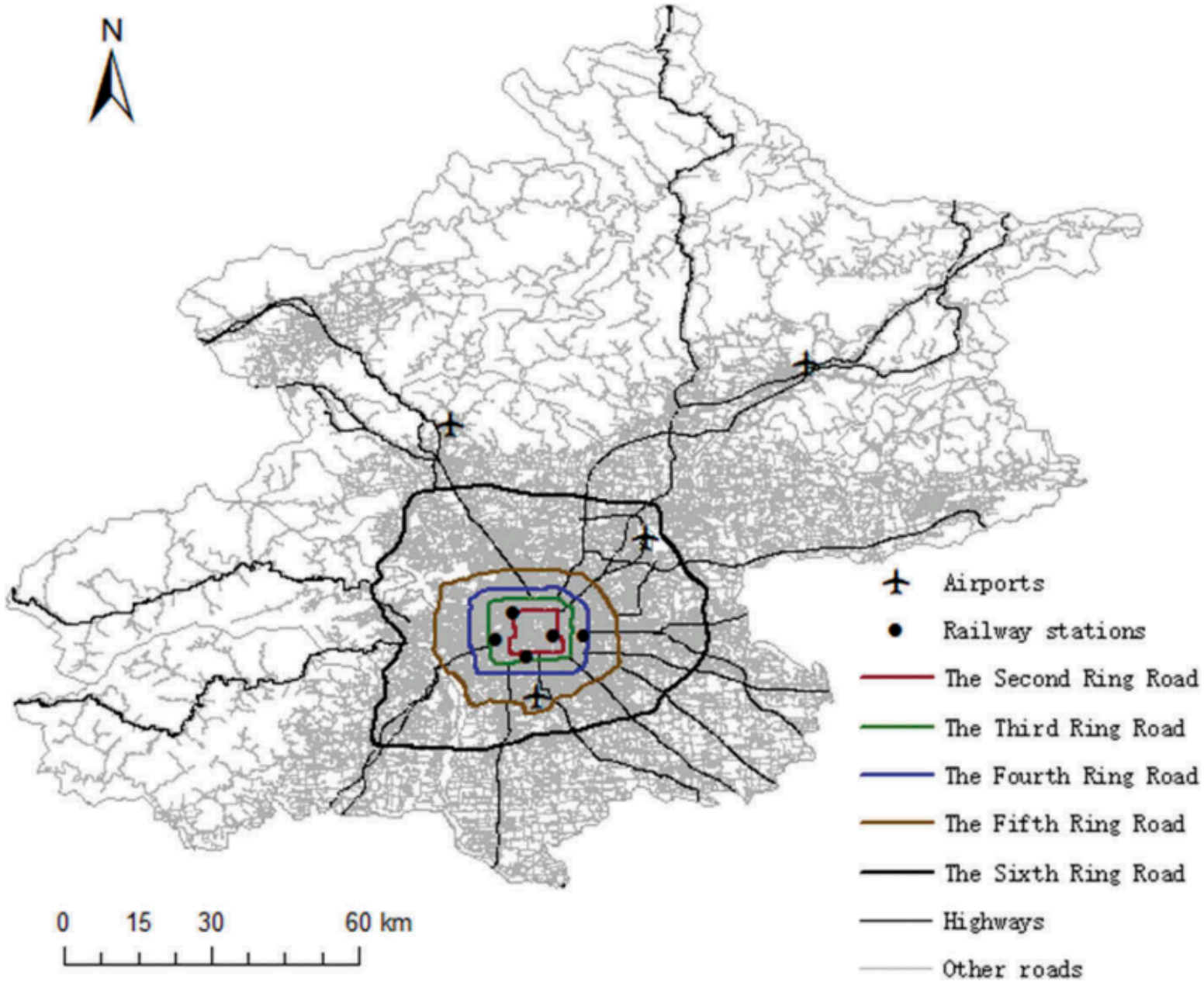


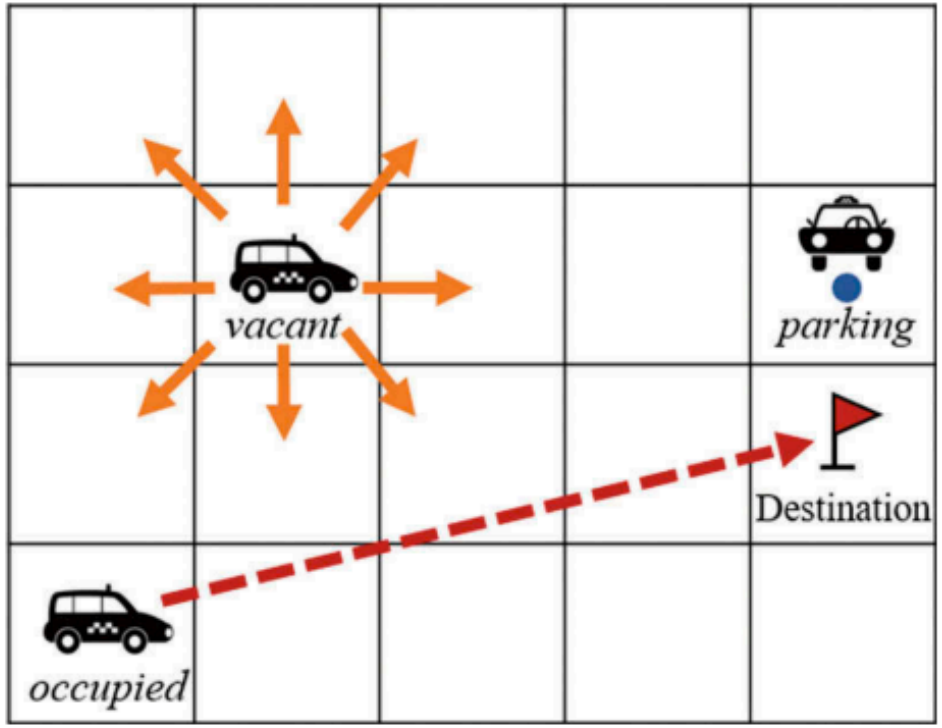
Figure 2. The roads in Beijing.

MDP formulation

- State
  - [location, status]
  - (cell_id, {occupied; vacant; parking})
  - Total number of states: N * 3

- Actions:
  - {empty_driving, carrying_passengers, waiting}
  - Size of the Q table?

- Reward: $R = \frac{d_o}{d_e}$



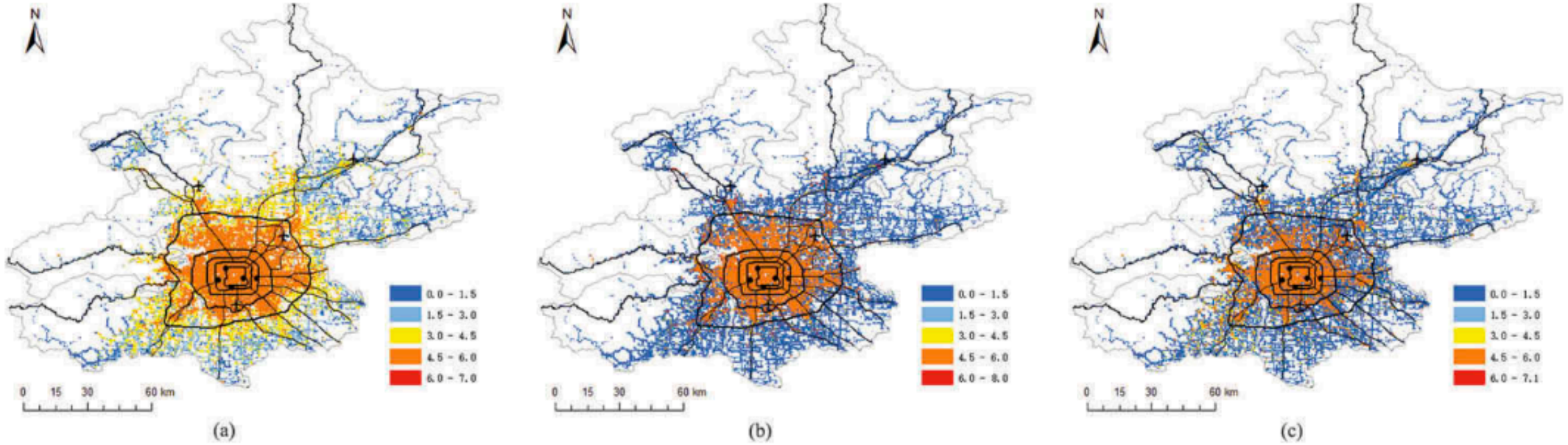


Figure 3. The scores of the three actions: (a) *empty_driving*, (b) *carrying_passengers*, (c) *waiting*.

- Best actions per zone:
  - Downtown: pick up requests, cruise around for passengers
  - Suburbs: don’t wait, cruise around

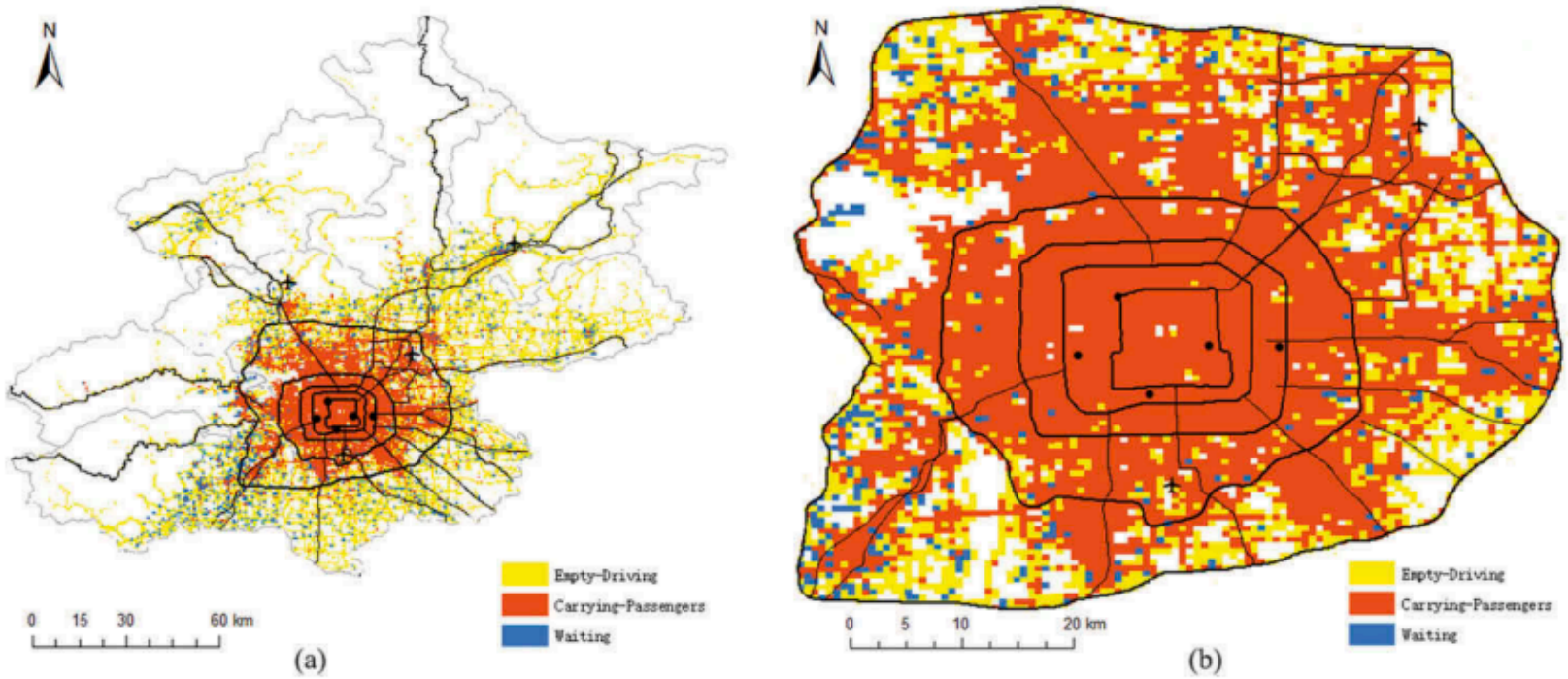


Figure 4. The distribution of the optimal policies (a) in Beijing and (b) within the Sixth Ring Road.

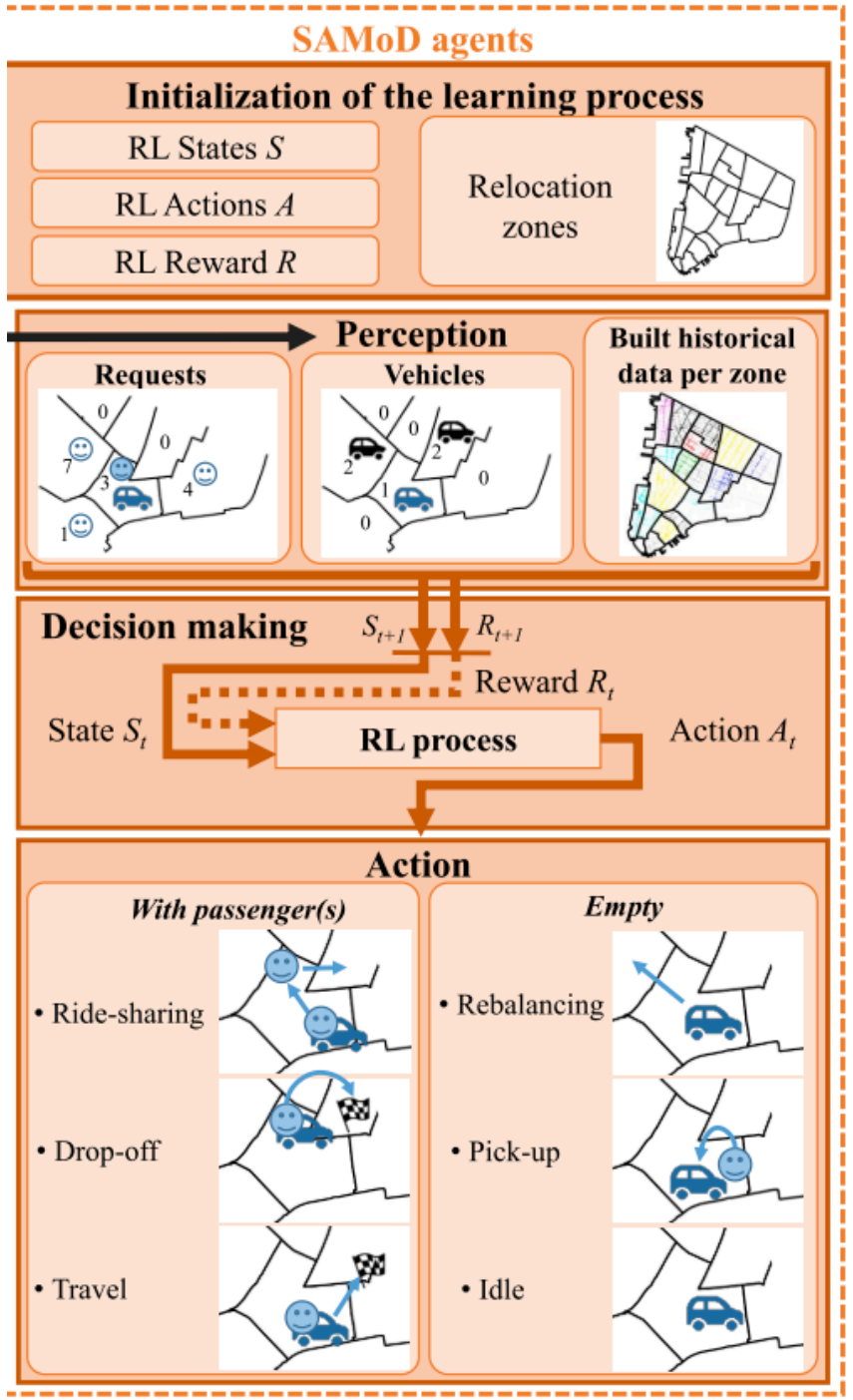
What are the issues with this paper?

SAMoD: Shared Autonomous Mobility-on-Demand using Decentralized Reinforcement Learning

Maxime Guériau and Ivana Dusparic
Enable - CONNECT Research Centre
School of Computer Science and Statistics, Trinity College Dublin
maxime.gueriau@scss.tcd.ie, ivana.dusparic@scss.tcd.ie

Problem description

- Learn the best behavior given the state
- 200 taxis in NYC
- 50 consecutive Tuesdays, 7-10 am
- Decentralized approach: each taxi learns on its own
  - Each taxi only knows about its neighboring zones
  - No historical info, builds up gradually



MDP formulation

- State:
  - {empty, hasPassengers, full}
  - Active requests in agent's zone? Yes/No
  - Active requests in neighboring zones? Yes/No
- Actions:
  - {pickUp, rebalance, doNothing}
- Reward:
  - 100 if has pax, 0 otherwise

Algorithm 1 SAMoD Agent Learning Process

```
1: // execute on learning event trigger
2: state ← MAPLOCALENVTOSTATE(env)
3: action ← QLEARNING.PICKACTION(state)
4: // if picking up new passengers
5: if action == "pickUp" then
6:     PICKUPPASSENGERS(nearestRequestID)
7: end if
8: state ← GETVEHICLESTATE
9: if state! = "full" then
10:    LISTENFORRIDESHARINGREQUESTS
11:    PICKUPPASSENGERS(nearestRequestID)
12: end if
13: // serve requests and finish learning episode
14: DROPOFFPASSENGERS(RequestID)
15: UPDATEENVIRONMENTHISTORICALDATA(RequestID)
16: UPDATEQLEARNINGPROCESS(RequestID)
17: TRIGGERNEWEVENT(DropOff)
18: // if rebalancing
19: if action == "rebalance" then
20:    //determine the zone to rebalance to
21:    current ← GETPENDINGREQUESTS(zones)
22:    historical ← GETHISTORICALREQUESTS(zones)
23:    zone ←SELECTZONE(current, historical)
24:    REBALANCETO(zone)
25:    TRIGGERNEWEVENT(Rebalanced)
26: end if
```

Results

		No RB, No RS		Rebalancing		Ride-sharing		RB and RS		SAMoD			
		C	D	C_RB	D_RB	C_RS	D_RS	C_RB_RS	D_RB_RS	S_RB	S_RB_RS	S_RB_RS+1	S_RB2_RS+1
System	Satisfied requests	29667	35388	30191	36913	38327	38368	38346	38407	35691	37790	37679	36159
	% of total requests	76.4	91.13	77.75	95.06	98.7	98.81	98.75	98.91	91.91	97.32	97.03	93.12
	Not served requests	8675	3098	8150	1590	0	54	0	11	2903	693	726	2242
	% of total requests	22.34	7.98	20.99	4.09	0	0.14	0	0.03	7.48	1.78	1.87	5.77
Riders	Avg t_w (min)	11.63	5.48	11.07	4.57	2.41	2.56	2.1	2.6	2.87	2.46	2.27	2.49
	Avg TT (min)	5.8	5.69	5.79	5.72	10.31	9.21	10.19	8.73	5.69	9.11	12.03	12.12
	Avg t_d (min)	0	0	0	0	4.57	3.47	4.44	2.99	0	3.39	6.31	6.49
Vehicles	Avg VMT	863.8	735.79	884.71	861.4	690.28	716.49	760.06	845.02	882.85	865.94	869.94	644.32
	Avg empty VMT	428.48	228.29	442.24	330.04	117.02	147.9	181.56	268.52	371.95	352.6	335.81	147.37
	Avg engaged VMT	435.32	507.5	442.47	531.36	573.26	568.59	578.5	576.5	510.91	513.34	534.13	496.95
	Avg shared VMT	103	120.55	103.78	125.54	382.75	324.74	376.86	301.96	115.84	330.3	433.86	409.11
	Avg occupancy	1.47	1.48	1.47	1.48	2.67	2.39	2.63	2.27	1.45	2.52	3.13	3.19

Transportation Research Part B 86 (2016) 250–267

Contents lists available at ScienceDirect

Transportation Research Part B

journal homepage: www.elsevier.com/locate/trb

Reinforcement learning approach for train rescheduling on a single-track railway

D. Šemrov[a,*], R. Marsetič[a], M. Žura[a], L. Todorovski[b], A. Srdic[a]

[a] University of Ljubljana, Faculty of Civil and Geodetic Engineering, Jamova 2, 1000 Ljubljana, Slovenia
[b] University of Ljubljana, Faculty of Administration, Gosarjeva 5, 1000 Ljubljana, Slovenia

Motivation

- The initial train timetable takes into account possible smaller disturbances, which can be compensated within the schedule.

- Bigger disruptions, such as accidents, rolling stock breakdown, prolonged passenger boarding, and changed speed limit cause delays that require train rescheduling.

- Objective is to find a feasible revision of the train timetable (within a reasonable time frame) for a delay detected by a dispatcher. The

Problem description

- 3 elements:
  - [fixed] Blocks, trains, safety procedures [single train in a block]
- Only one train can occupy a given block section at a given time, a moving train can only stop at the signals

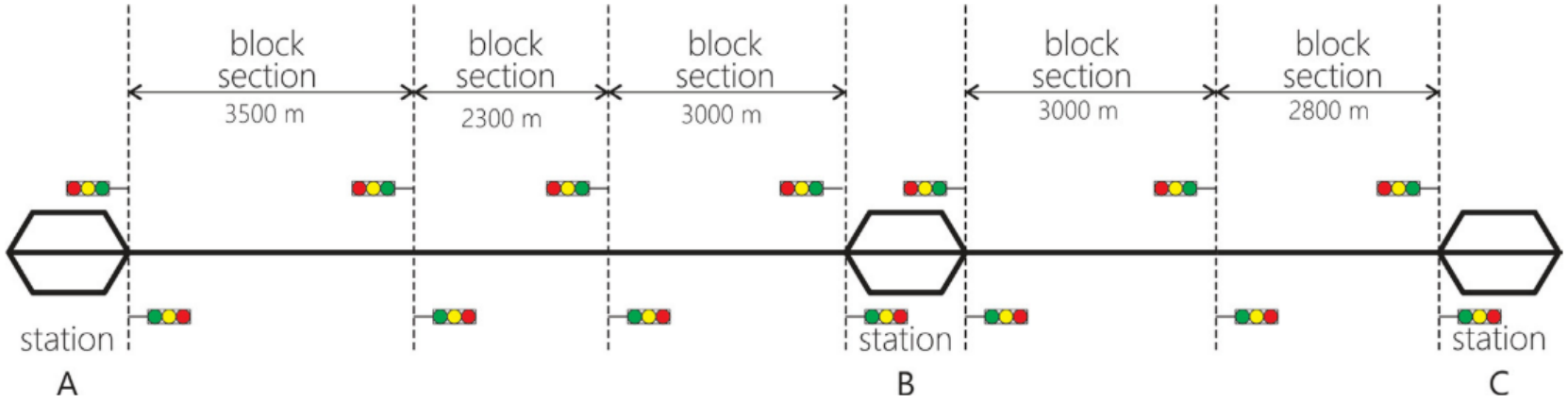


Fig. 2. Layout of a simple railway infrastructure with three stations (A, B, and C) and five block sections.

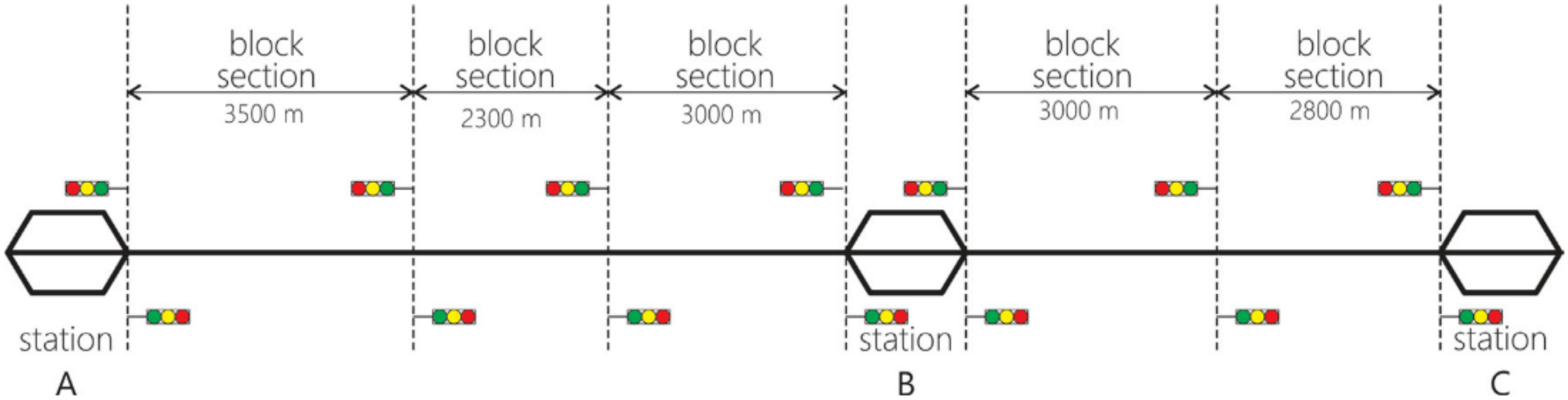


Fig. 2. Layout of a simple railway infrastructure with three stations (A, B, and C) and five block sections.

MDP formulation

- State: current locations of the trains (block section), the current infrastructure (block section) availability and time

MDP formulation

- State: current locations of the trains (block section), the current infrastructure (block section) availability and time

- Actions: The dispatcher decides which trains can move on to the next section.
  There are two possible actions for each signaling element in the infrastructure: setting it to red (“Stop”) or green (“Go”) color

MDP formulation

- State: current locations of the trains (block section), the current infrastructure (block section) availability and time
- Actions” the dispatcher decides which trains can move on to the next section.
  There are two possible actions for each signaling element in the infrastructure: setting it to red (»Stop«) or green (»Go«) color

- Reward: – delay

The delayed reward is given to the agent at the end of the training episode, when the environment reaches its goal state (when all trains arrive at their end stations).

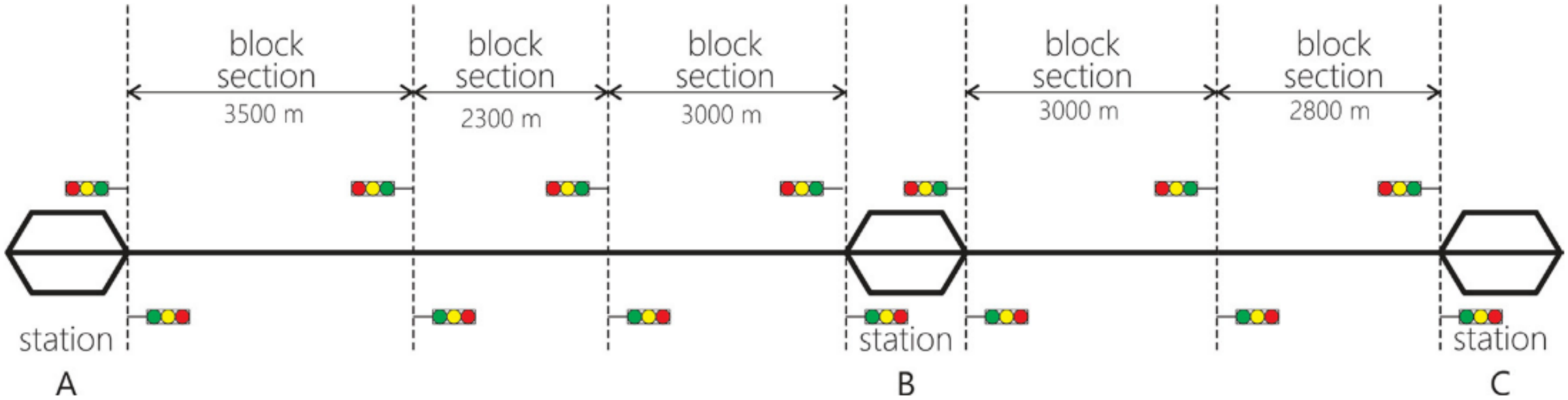


Fig. 2. Layout of a simple railway infrastructure with three stations (A, B, and C) and five block sections.

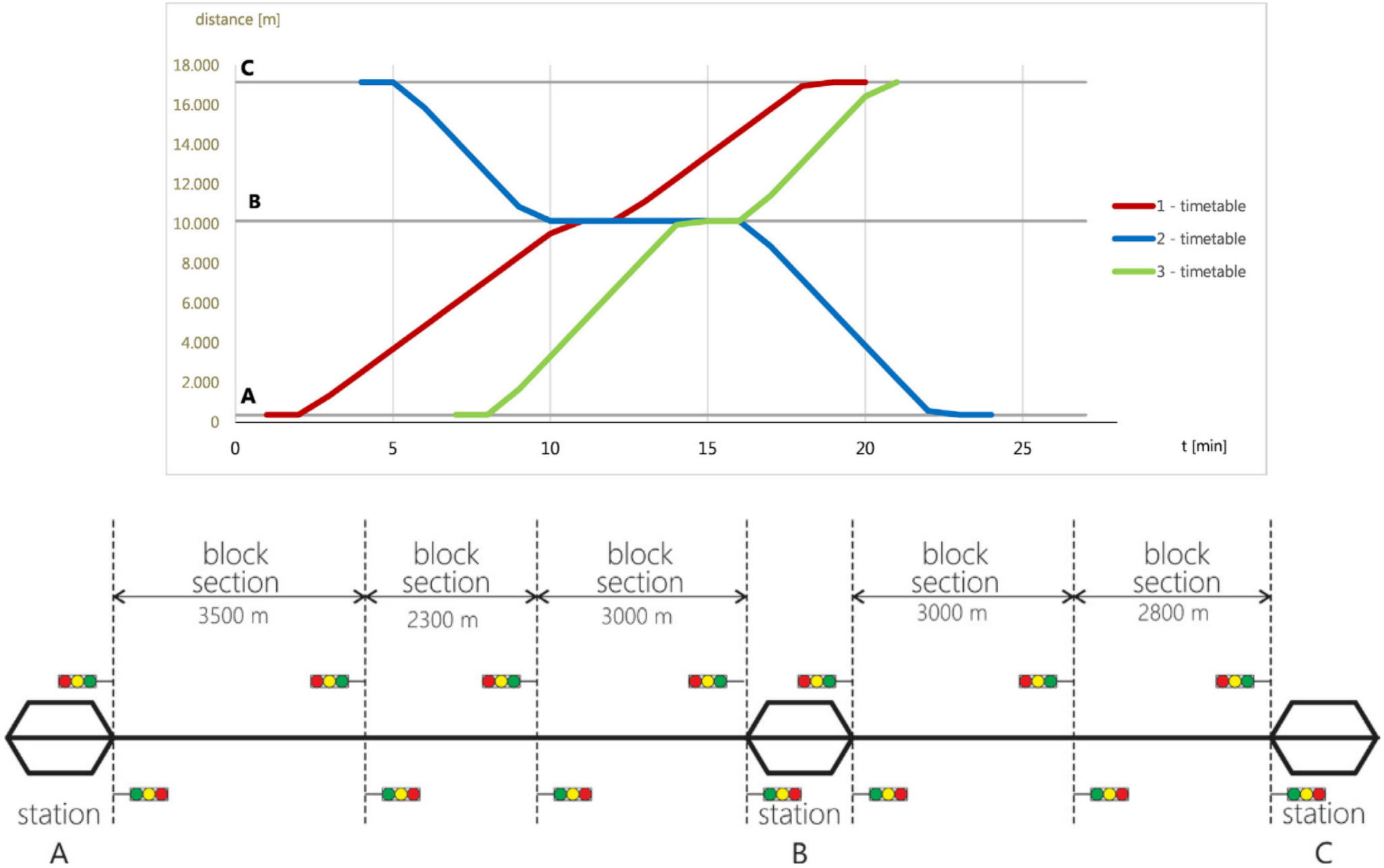
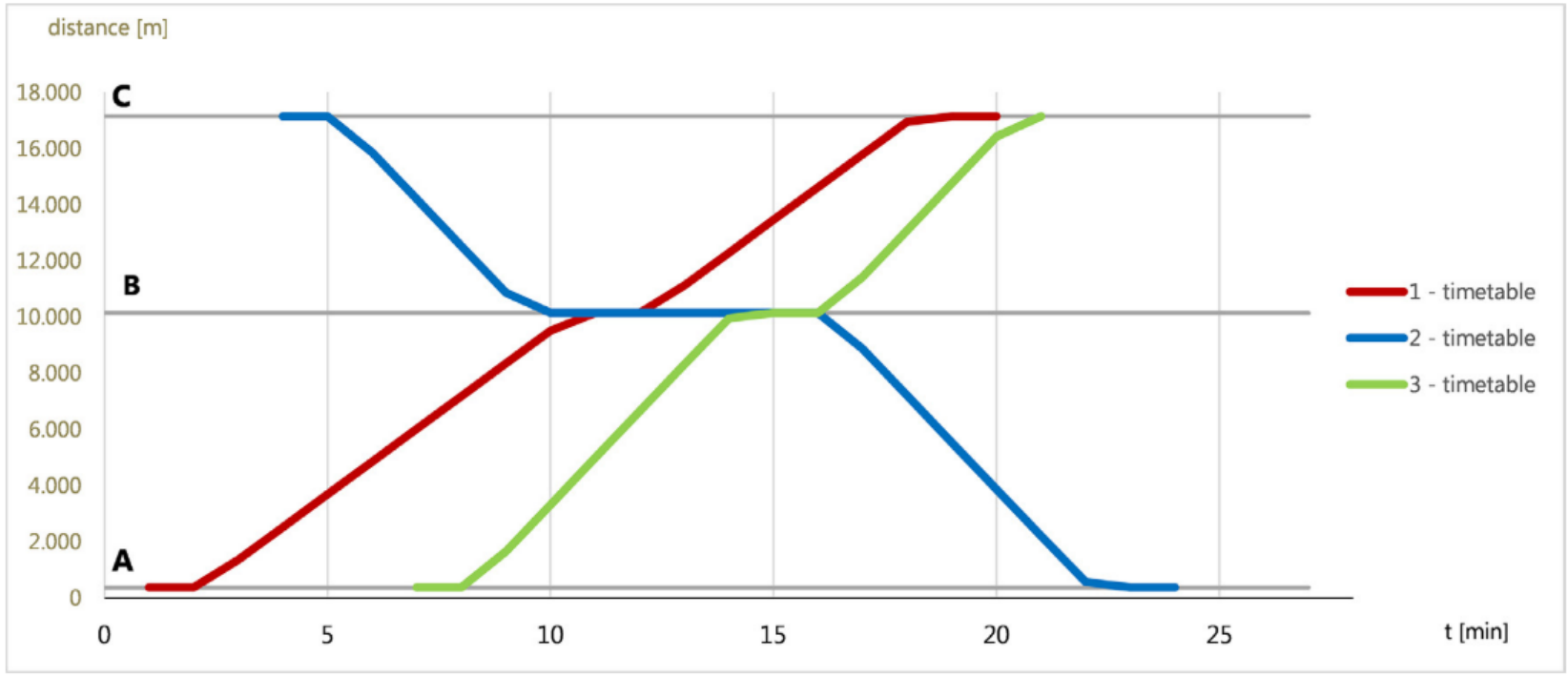


Fig. 2. Layout of a simple railway infrastructure with three stations (A, B, and C) and five block sections.



Train A is 6 minutes delayed

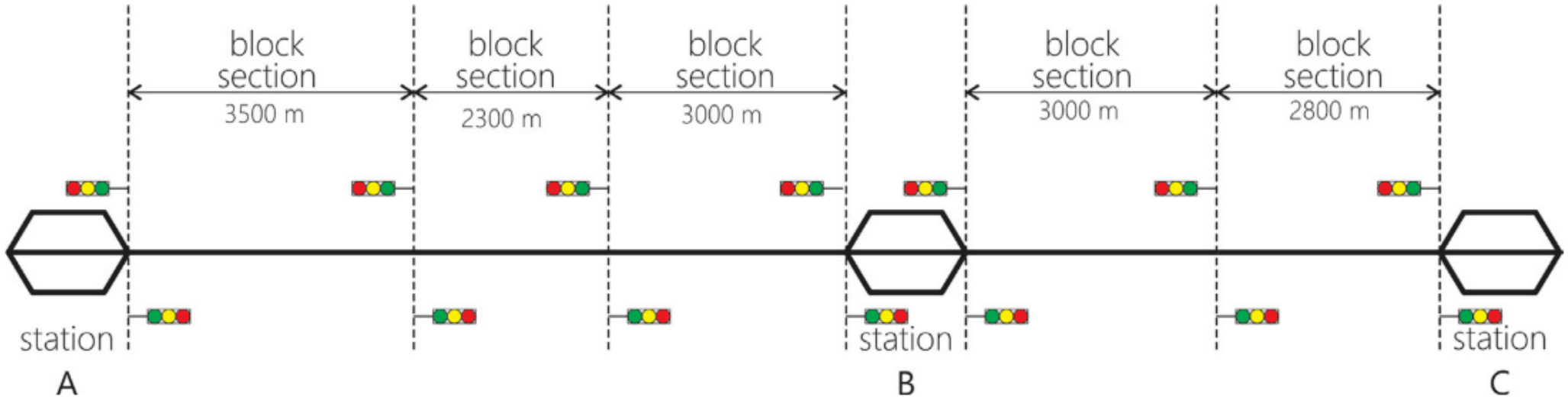
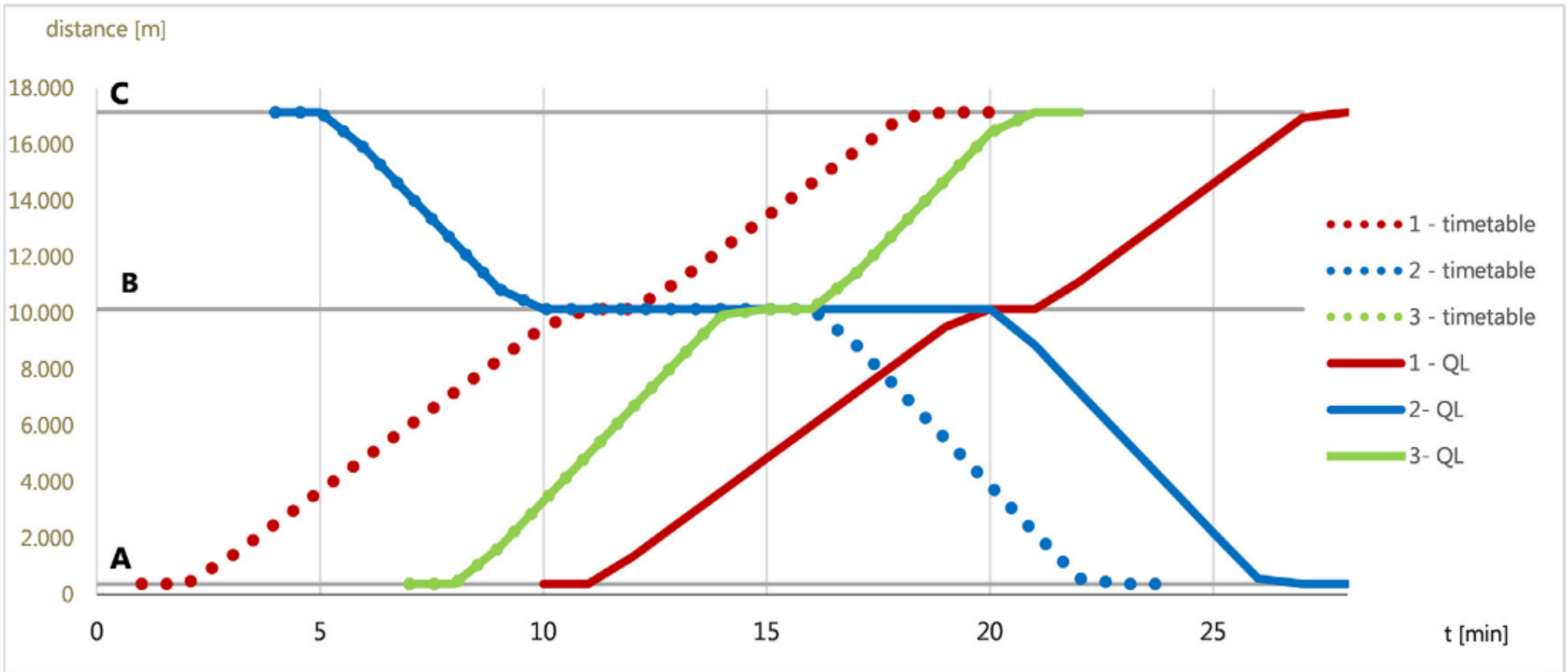
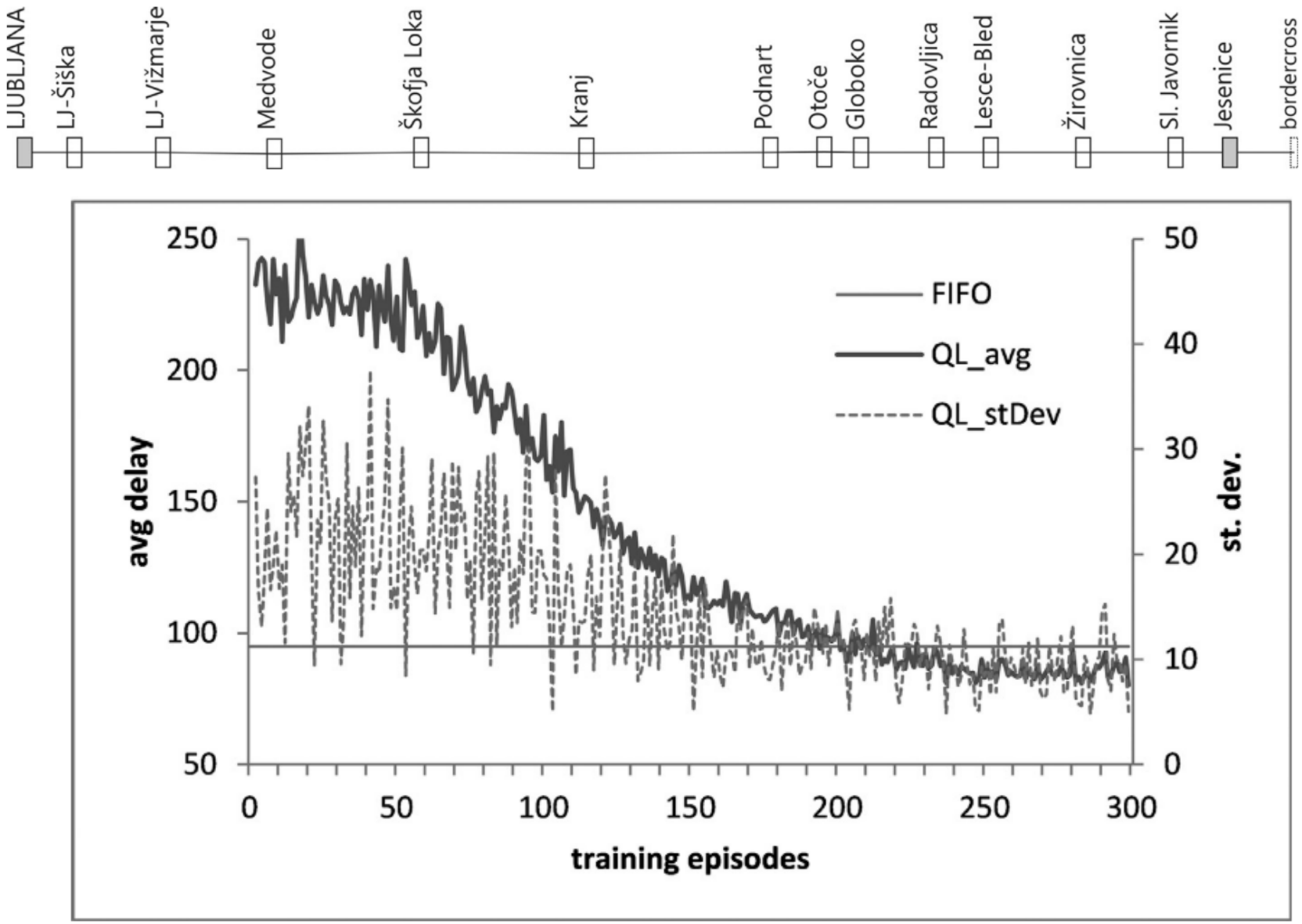


Fig. 2. Layout of a simple railway infrastructure with three stations (A, B, and C) and five block sections.


distance [m]
18.000
16.000
14.000
12.000
10.000
8.000
6.000
4.000
2.000
0
C
B
A
0
5
10
15
20
25
t [min]
1 - timetable
2 - timetable
3 - timetable
1 - QL
2- QL
3- QL


LJUBLJANA
LJ-Šiška
LJ-Vižmarje
Medvode
Škofja Loka
Kranj
Podnart
Otoče
Globoko
Radovljica
Lesce-Bled
Žirovnica
Sl. Javornik
Jesenice
bordercross
250
200
150
100
50
avg delay
50
40
30
20
10
0
st. dev.
FIFO
QL_avg
QL_stDev
0
50
100
150
200
250
300
training episodes

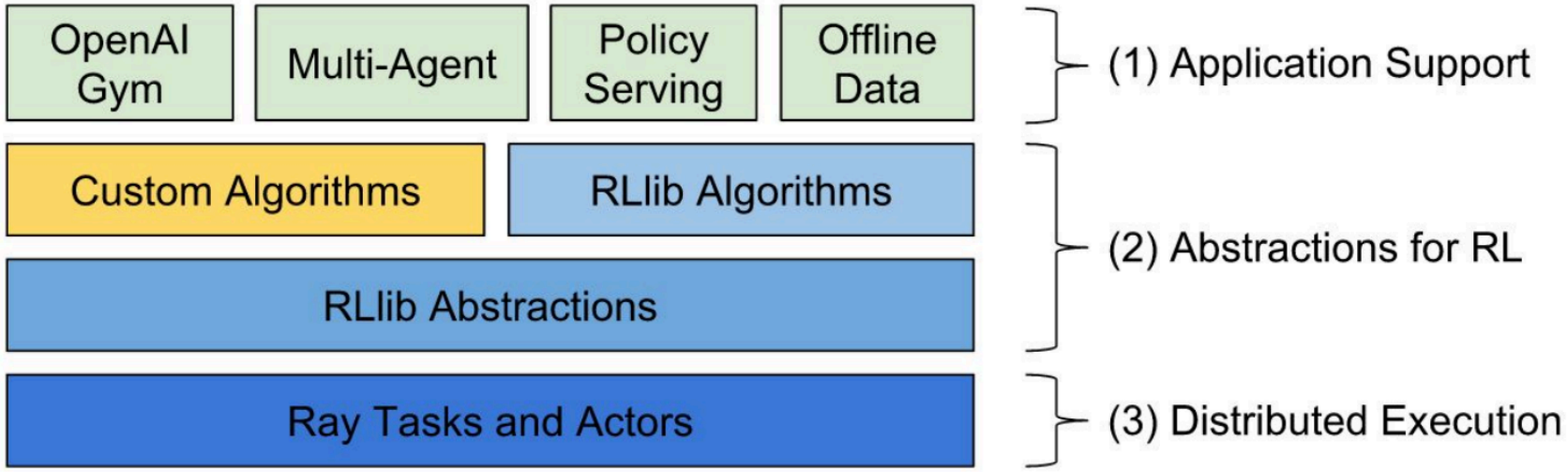
Computational cost

- Transit simulations are usually done with OpenTrack
  - Microscopic
  - Very detailed
  - Take a long time to run
- 300 episodes
  - 15 minute each
  - 75 hours!

Solutions

- Write a simplified, highly optimized simulation
  - In the previous example, they wrote their own simulation in VB

- Distributed training
  - Rllib (https://ray.readthedocs.io/en/latest/rllib.html)



Transportation Research Part C 88 (2018) 1–16

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Reinforcement learning approach for coordinated passenger inflow control of urban rail transit in peak hours

Zhibin Jiang[a], Wei Fan[a,b,*], Wei Liu[c], Bingqin Zhu[d], Jinjing Gu[a]

Real-time passenger flow control

Table 1
Passenger inflow control for URT in major cities in China.

City	The number of operating lines	The number of inflow control lines	The number of inflow control stations	Time Periods for passenger inflow control
Beijing	18	14	75	Morning peak 7:00–9:00 Evening peak 17:00–19:00
Shanghai	14	13	50	Morning peak 7:30–9:00 Evening peak 17:30–19:00
Guangzhou	9	6	38	Morning peak 8:00–9:00 Evening peak 17:30–19:00

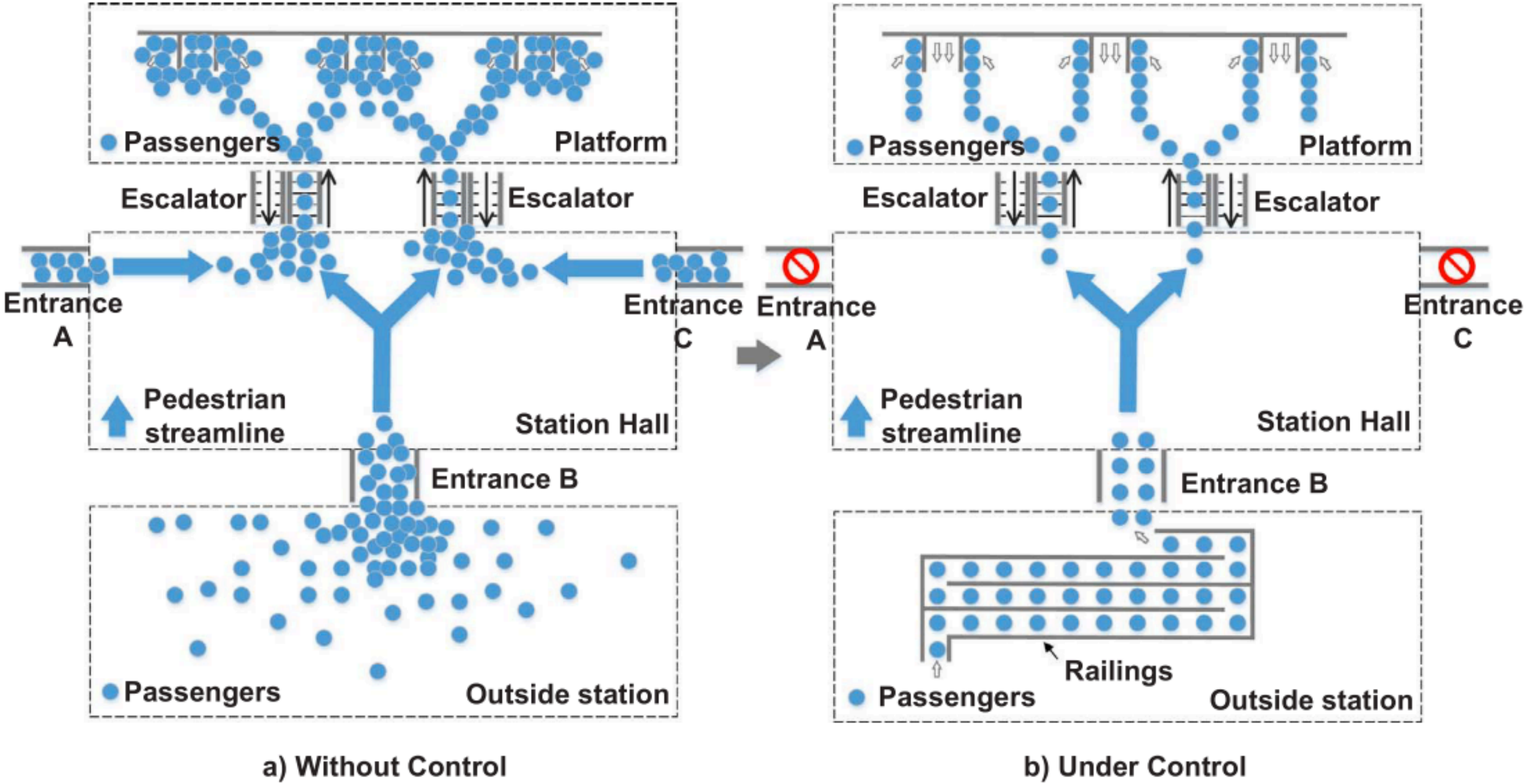
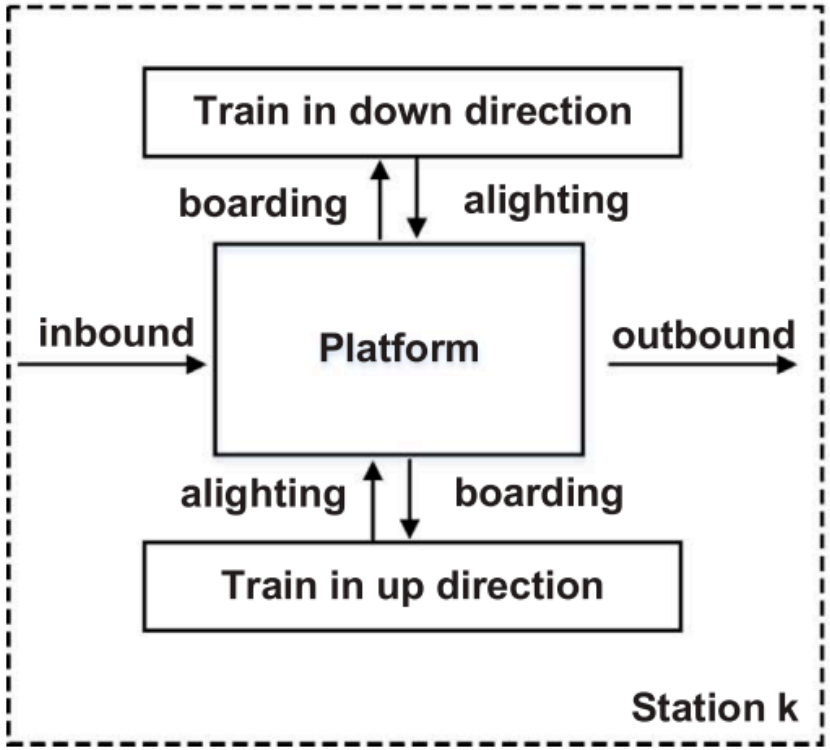
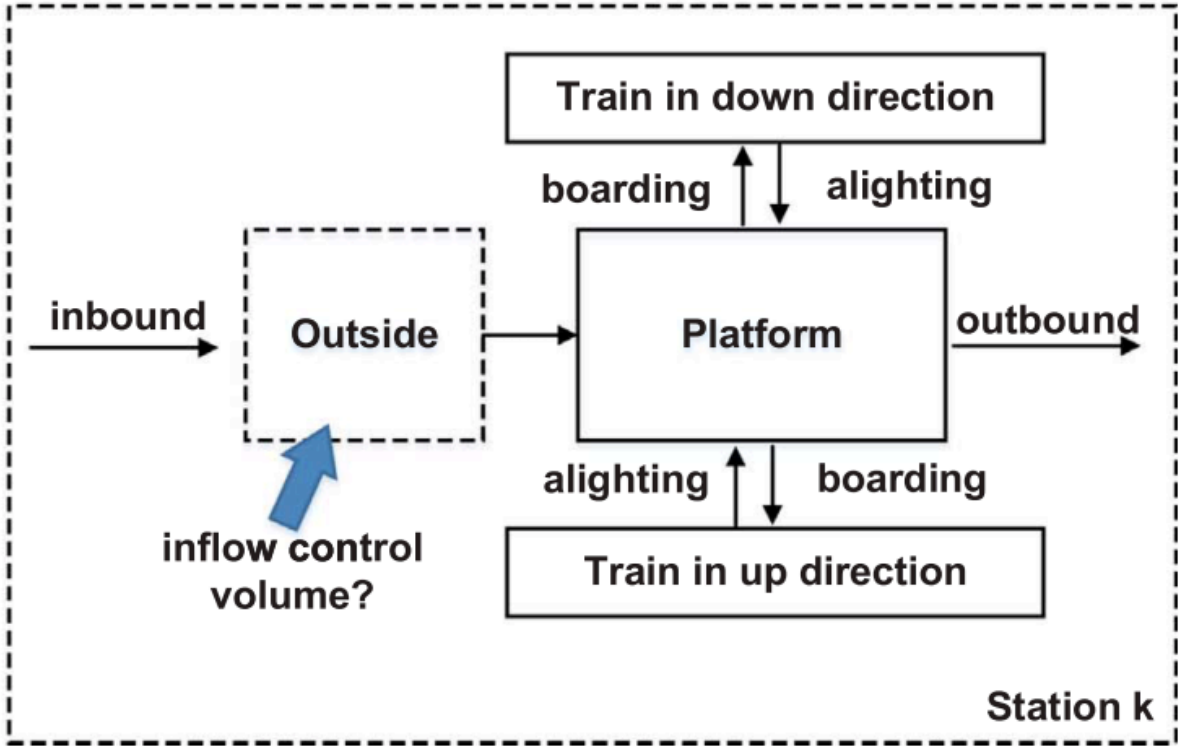


Fig. 1. The effect of passenger inflow control on keeping passengers in order at the station.

Real-time passenger flow control



a) Without Control



b) Under Control

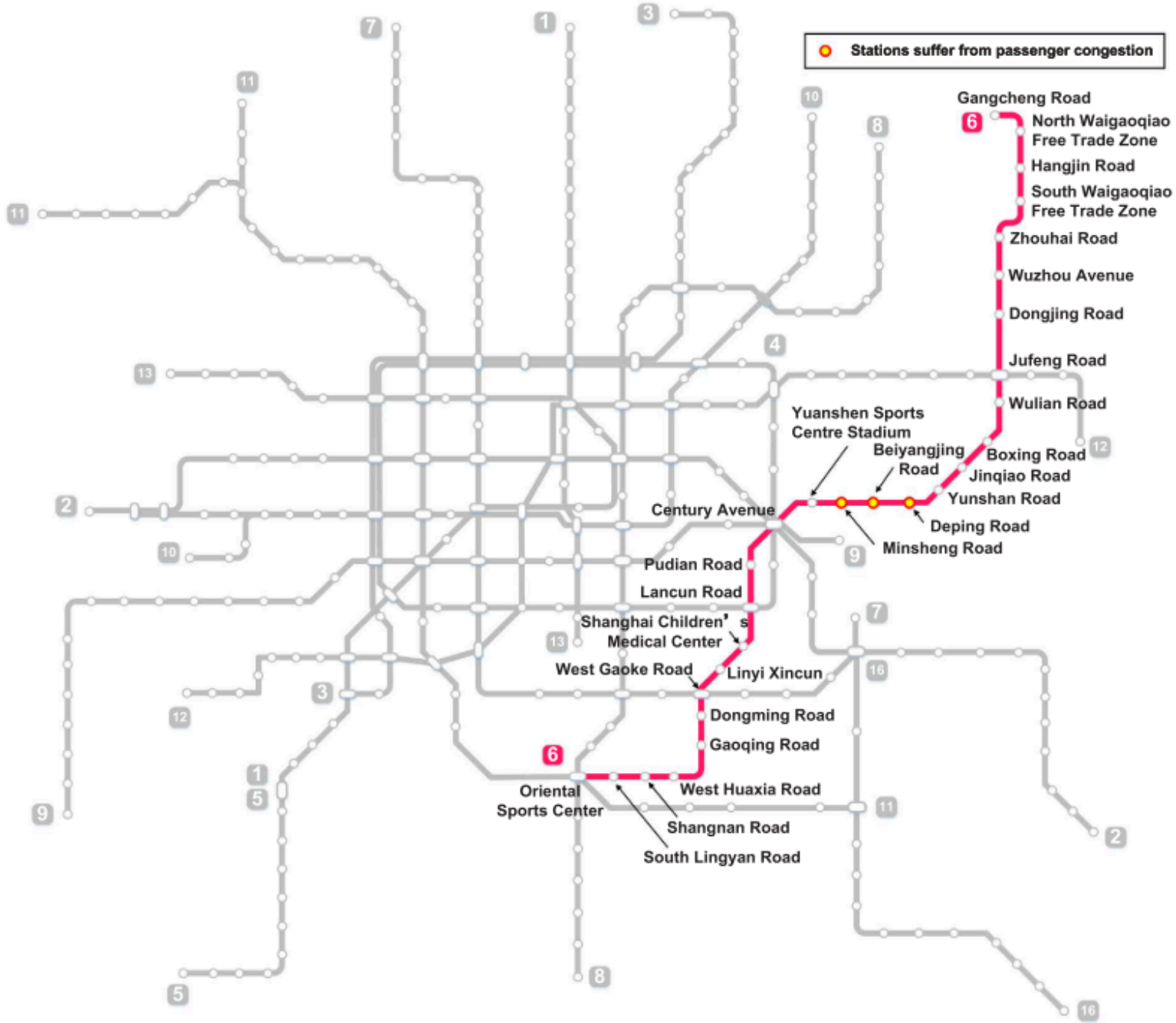


Fig. 7. URT network topology in Shanghai.

Linear programming approach

$$MinZ = \sum_{i \in I} \sum_{j \in J} \beta_j \cdot (T_{j,k}^D - T_{i,j}^G)$$

$$\beta_j = \begin{cases} e^{2d_j}, & 0 \leqslant d_j \leqslant 3, j \in J \\ e^8, & d_j \geqslant 4, j \in J \end{cases}$$

$$d_j = R_{i,(T_{i,j}^G, T_{j,k}^D)} - 1, j \in J, k \in K, i = p_j^O$$

$$T_{i,j}^C = T_{i,j}^B + t^E, j \in J, i = s_j^O / p_j^O \tag{4}$$

$$T_{j,k}^D = T_{i,k}^d, q_{j,k}^4 = 1, j \in J, k \in K, i = p_j^O \tag{5}$$

$$T_{i,j}^E = T_{j,k}^D + r_{k,(p_j^o, p_j^D)}, q_{j,k}^4 = 1, j \in J, k \in K, i = p_j^D \tag{6}$$

$$q_{i,j,t}^1 = \begin{cases} 1, & t \in [T_{i,j}^A, T_{i,j}^B), j \in J, i = s_j^O \\ 0, & otherwise \end{cases} \tag{7}$$

$$q_{i,j,t}^2 = \begin{cases} 1, & t \in [T_{i,j}^C, T_{j,k}^D), j \in J, k \in K, i = p_j^O \\ 0, & otherwise \end{cases} \tag{8}$$

$$q_{j,k,t}^3 = \begin{cases} 1, & t \in [T_{j,k}^D, T_{i,j}^E), q_{j,k}^4 = 1, i = p_j^O / p_j^D, j \in J, k \in K \\ 0, & otherwise \end{cases} \tag{9}$$

$$\sum_{k \in K} q_{j,k}^4 = 1, j \in J \tag{10}$$

$$P_{i,t}^p = \sum_{j \in J} q_{i,j,t}^2, i \in I, t \in M \tag{11}$$

$$P_{i,t}^p \leqslant A_i \cdot \theta, i \in I, t \in M \tag{12}$$

$$P_{k,t}^r = \sum_{j \in J} q_{j,k,t}^3, k \in K, t \in M \tag{13}$$

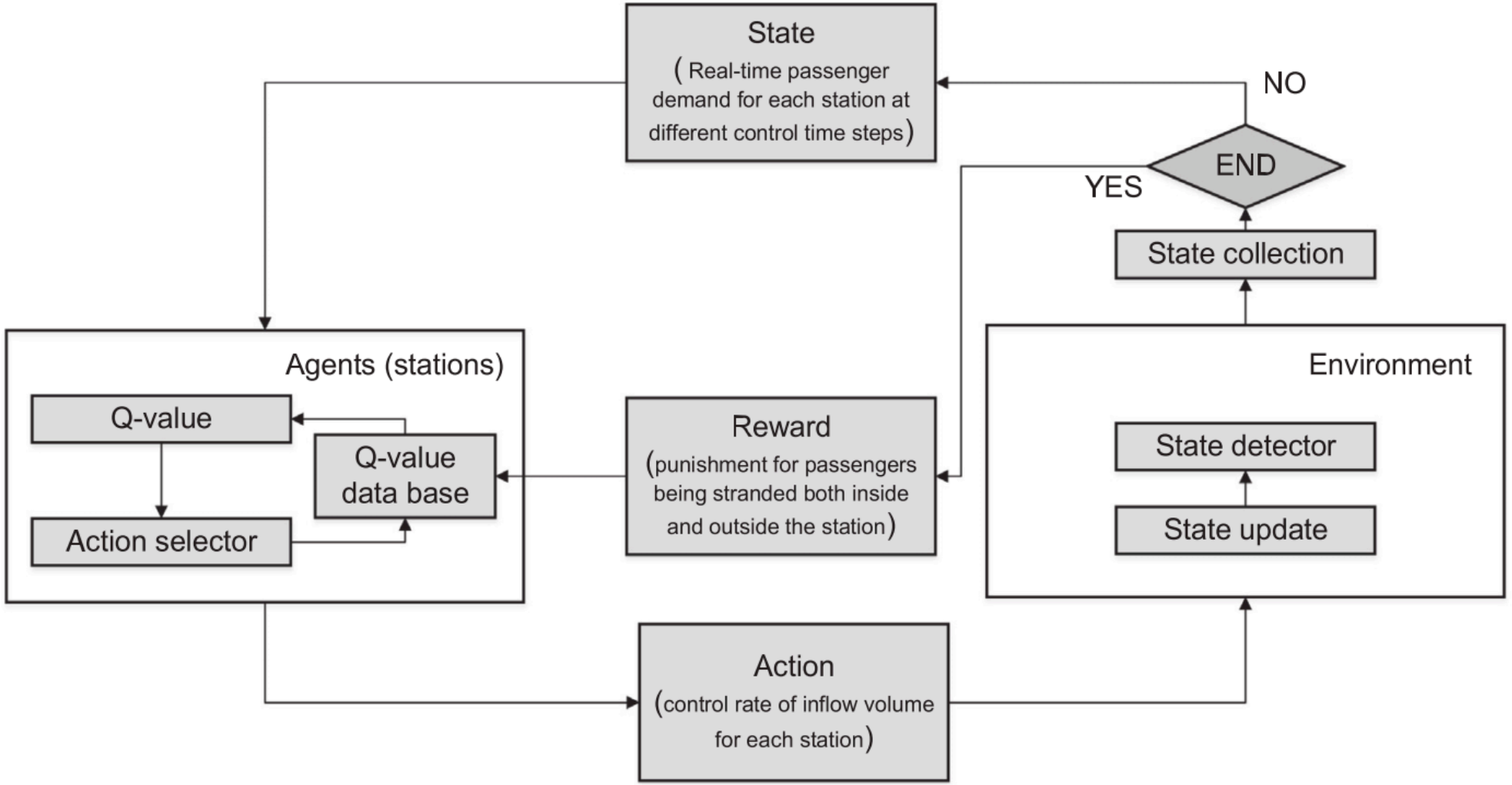
$$P_{k,t}^r \leqslant C \cdot \varphi, k \in K, t \in M \tag{14}$$

$$P_{i,n}^e = \sum_{j \in J} \sum_{t = T_0 + (n-1) \cdot t^c}^{T_0 + n \cdot t^c} q_{i,j,t}, i \in s_j^O, n \in N \tag{15}$$

$$P_{i,n}^a = \sum_{j \in J} \sum_{t = T_0 + (n-1) \cdot t^c}^{T_0 + n \cdot t^c} q_{i,j,t}^1, i \in s_j^O, n \in N \tag{16}$$

$$a_{i,n} = \frac{P_{i,n}^a - P_{i,n}^e}{P_{i,n}^a}, i \in I, n \in N \tag{17}$$

Real-time passenger flow control



MDP formulation

- State

$$\frac{total\ inflow\ at\ time\ t}{total\ inflow\ at\ time\ t + waiting\ outside\ since\ (t-1)}$$

- Action
  - Percentage of pax denied entrance; {0%, 20%, 40%, 60%, 80%, 100%}

- Reward
  - Total waiting time, with a penalty for pax stranded outside

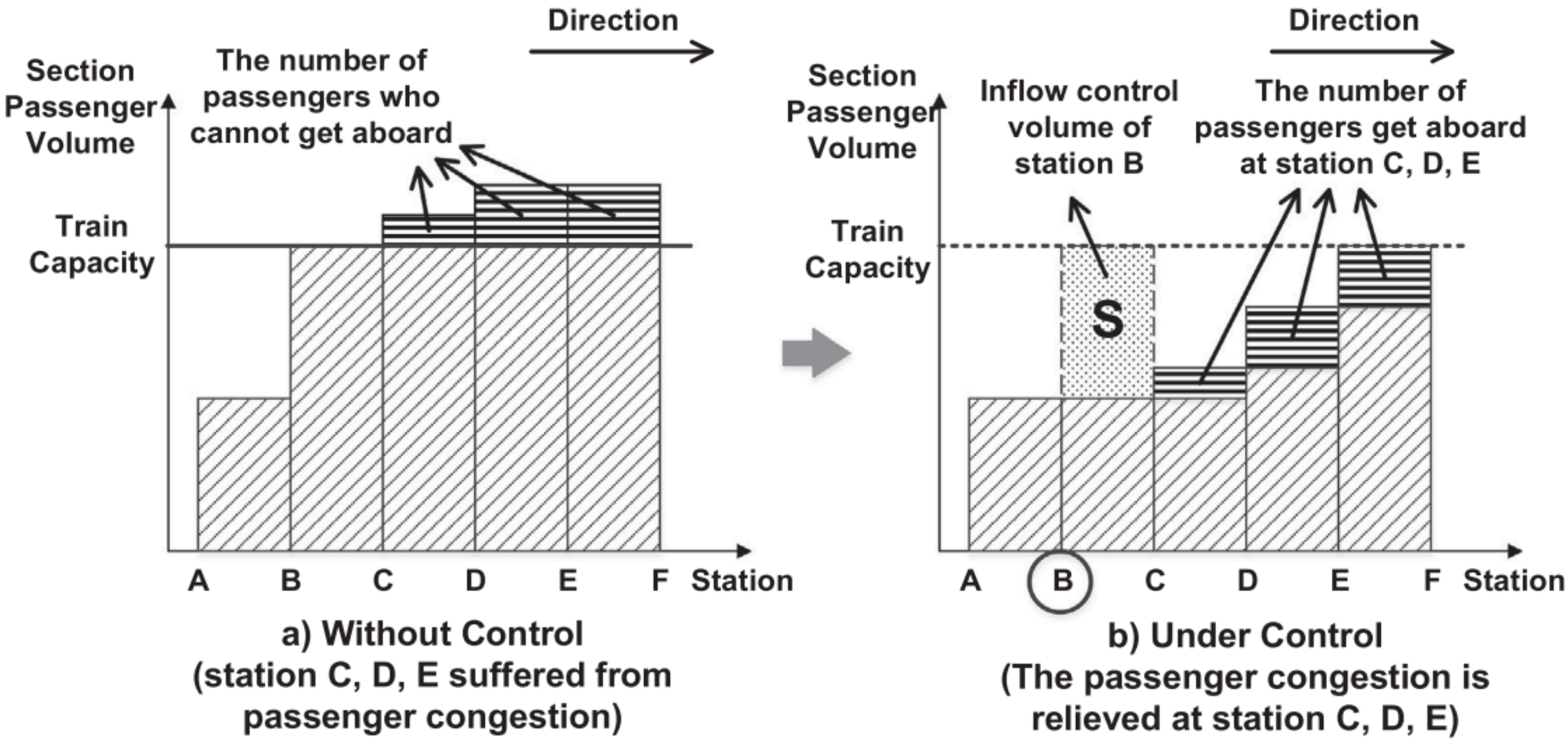


Fig. 2. The effect of coordinated inflow control on balancing the utilization of the train capacity.

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Analysing the impact of travel information for minimising the regret of route choice☆

Gabriel de O. Ramos*, Ana L.C. Bazzan, Bruno C. da Silva

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

Problem description

- Self-interested drivers aim at choosing routes that minimize travel costs between their origins and destinations
- Regret is how much worse an agent performs compared to the best fixed action in hindsight
- Regret cannot be computed by agents because its calculation requires observing the costs of all available routes (including non-taken ones)

$$\mathcal{R}_i^T = \max_{a_i^t \in A_i} \frac{1}{T} \sum_{t=1}^{T} r(a_i^t) - \frac{1}{T} \sum_{t=1}^{T} r(\dot{a}_i^t)$$

Problem description

- The app is responsible for providing travel information to the agents
- It has access to the travel time of all routes within the network
- It keeps track of the average travel time of each route
- The recommendation is only used to compute the agents' regret

Problem formulation

- Each driver is an agent

- Each agent has a local estimate of regret
  - Built up from its own experience

$$H_i = \{r(a_i^t) | a_i^t \in A_i, t \in [1, T]\}$$

- The app provides a global estimate of regret
  - Averaged over all drivers

$$\hat{r}(a^t) = \frac{1}{t} \sum_{u=1}^{t} r(a^u)$$

- Each agent chooses an action based based on its local estimate, but afterwards updates it based on the experienced reward and app info

Problem formulation

- Agent's estimate of regret for a route

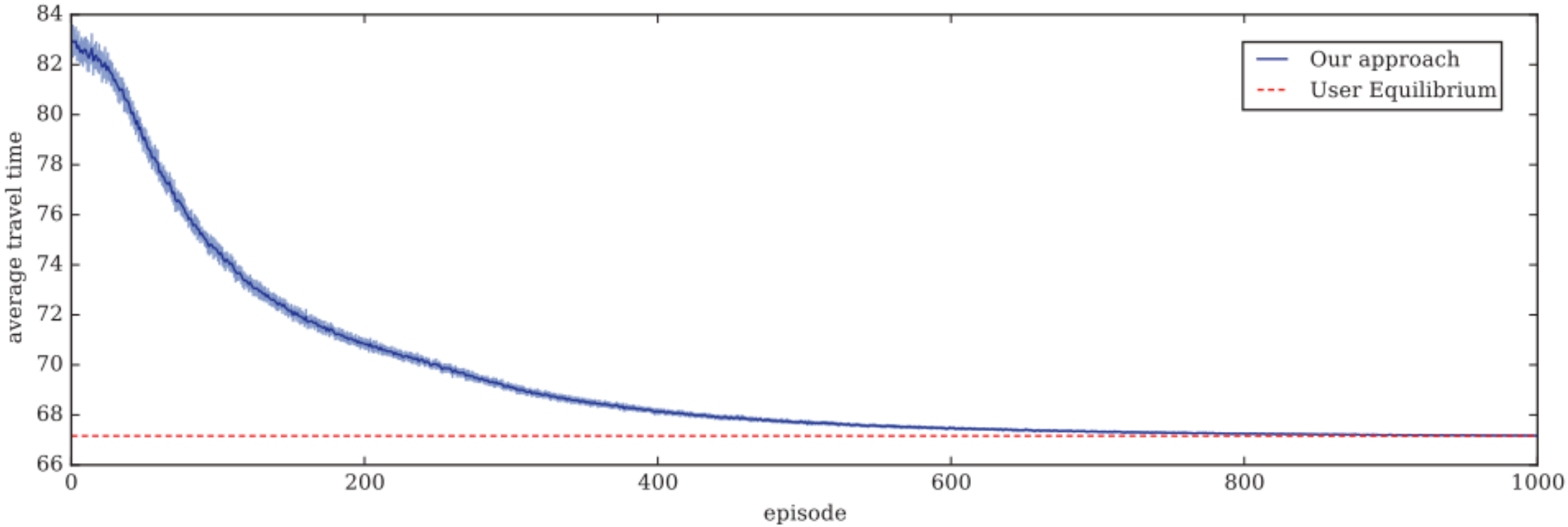
$$\tilde{r}(a_i^t) = \begin{cases} r(a_i^t) & \text{if } a_i^t = \dot{a}_i^t \\ \tilde{r}(a_i^{t-1}) & \text{otherwise} \end{cases}$$

- Agent's estimated action regret

$$\tilde{\mathcal{R}}_{i,a}^T = \max_{b_i^t \in A_i} \frac{1}{2} \left[\hat{r}(b_i^t) + \frac{1}{T} \sum_{t=1}^{T} \tilde{r}(b_i^t) \right] - \frac{1}{T} \sum_{t=1}^{T} \tilde{r}(a_i^t)$$

Algorithm 1. Algorithm overview (for agent i).

initialise Q-table: $Q(a_i) \leftarrow 0 \quad \forall\, a_i \in A_i$;
initialise history of estimates: $H_i \leftarrow \varnothing$;
for $t \in T$ **do**
    receive app recommendations $\{\hat{r}(a_i^t) | a_i \in A_i\}$;
    choose action $\dot{a}_i^t \in A_i$ using ϵ-greedy;
    take action $\dot{a}_i^t$ and observe reward $r(\dot{a}_i^t)$;
    for $a_i^t \in A_i$ **do**
        update estimate $\widetilde{r}(a_i^t) \in H_i$ using Eq. (3);
    end
    update regret $\widetilde{\mathcal{R}}_{i,\dot{a}_i^t}^t$ of action $\dot{a}_i^t$ using Eq. (4);
    update Q-value of $\dot{a}_i^t$ using Eq. (6):
$Q(\dot{a}_i^t) \leftarrow (1-\alpha) Q(\dot{a}_i^t) + \alpha \widetilde{\mathcal{R}}_{i,\dot{a}_i^t}^t$;
end


84
82
80
78
76
74
72
70
68
66
average travel time
0
200
400
600
800
1000
episode
Our approach
User Equilibrium

Q-Learning for Flexible Learning of Daily Activity Plans

David Charypar and Kai Nagel

Motivation

- Activity-based demand generation

- For each member of a synthetic population, a daily activity plan stating a sequence of activities (e.g., home-work-shop-home), including locations and times, needs to be found. Activities at different locations generate demand for transportation.

MDP formulation

- State: type of activity, starting time of activity, time already spent at activity

- Action: The possible actions are either to stay at a given activity or to move to another activity.

- Rewards are given as “utility per time slice,” which corresponds to a coarse version of marginal utility

TABLE 5 Optimal Day Plan for More Realistic Example

Resolution	Work	Shop	Leisure	Home
60 min	08:00–17:00	18:00–18:00 (skipped)	19:00–23:00	00:00–07:00
30 min	08:00–17:30	18:00–18:30	19:00–23:30	00:00–07:00
15 min	08:00–17:30	17:45–18:15	18:30–23:45	00:15–07:15

TABLE 3 Travel Times for More Realistic Example

		t_{travel} for . . .		
From	To	. . . t_{res} = 15 min	. . . t_{res} = 30 min	. . . t_{res} = 60 min
Home	Work	45 min	60 min	60 min
Work	Shop	15 min	30 min	60 min
Shop	Leisure	15 min	30 min	60 min
Leisure	Home	30 min	30 min	60 min

NOTE: Travel times are assumed to be constant throughout the day.

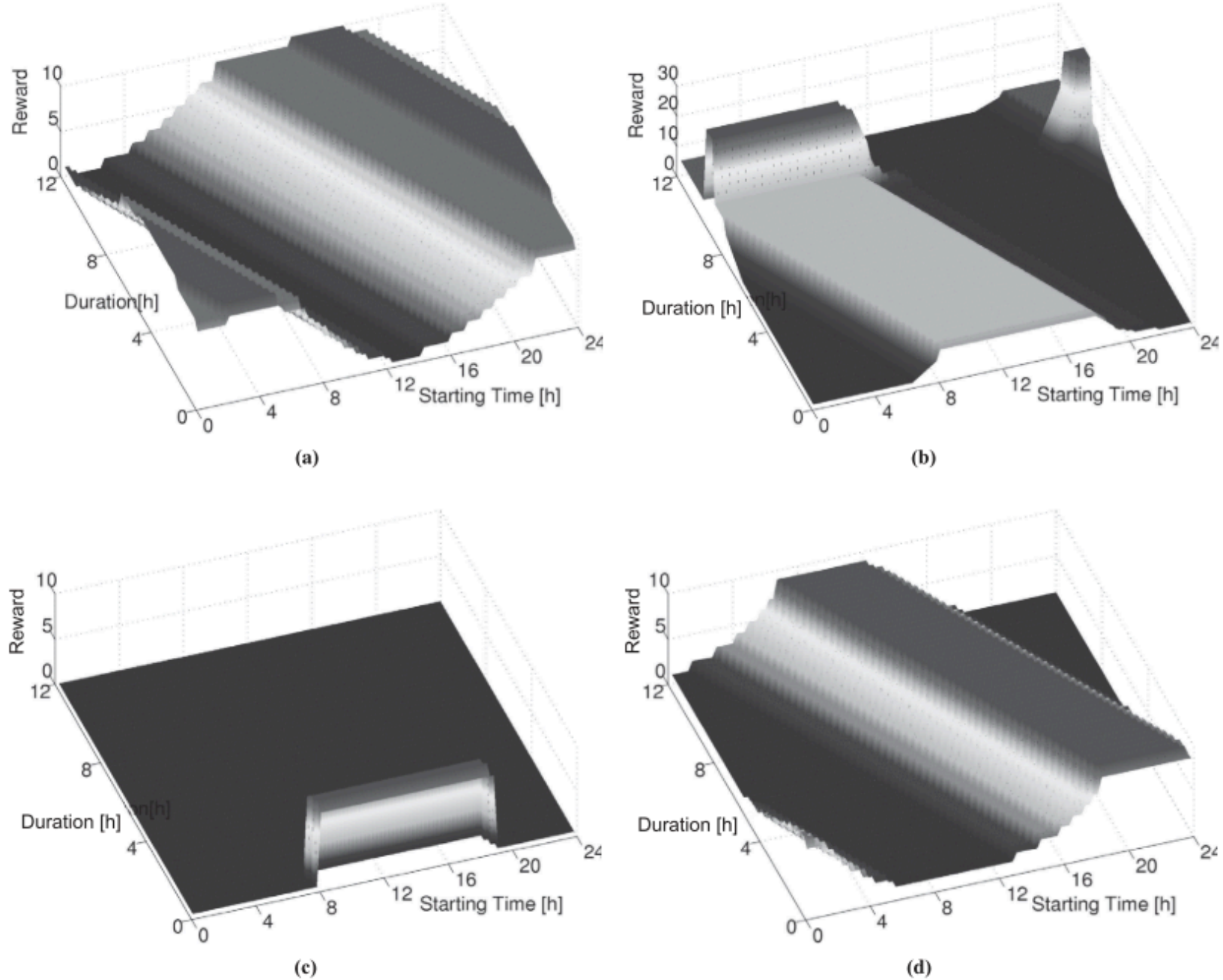


FIGURE 1 Plot of rewards per 15 min for different activities in more realistic example. Rewards depend on starting time and duration of activity: (*a*) home activity, (*b*) work activity, (*c*) shop activity, and (*d*) leisure activity.

TABLE 6 Computational Performance with More Realistic Example Using High Initialization

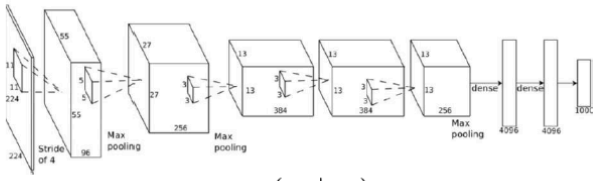
Resolution $t_{\text{resolution}}$	Number of Iterations	Running Time
60 min	50k	143 ms
30 min	500k	545 ms
15 min	2M	1.98 s

NOTE: Number of iterations needed to converge to optimal solution is indicated with probability substantially higher than 50%.

Approximate solutions

- We are often in interested in problems where the state-action combinations are combinatorial
  - State includes
    - Images
    - Continuous variables
    - Large number of features
  - Actions might also be continuous

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

- In such situations, we cannot use a table to hold every possible state-action combination and its value
- Store parameterized state (or state-action) value functions!