IE 481 & IE 801
Game Theory and Multi-Agent Reinforcement Learning: Intersection of Game Theory and Artificial Intelligence

Jinkyoo Park
SYSTEMS INTELLIGENCE LAB.
KAIST

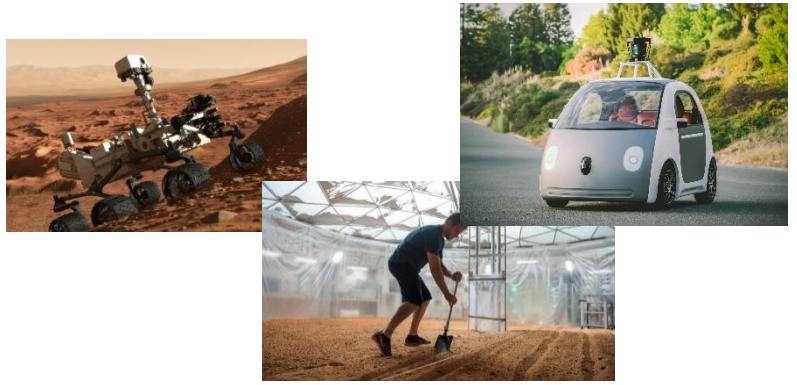
Contents

1. Motivation

Engineering is all about decision makings

- Machine Learning
- **Artificial Intelligence**
- Optimization
- Optimum Control
- Planning
- Markov Decision Process
- Influential Diagram
- Decision Tree
- Dynamic Control
- **Game Theory**
- Search
- Stochastic Programming
- Dynamic programming
- Reinforcement Learning
- Bandit problem

⋮



What are ***the differences*** in these decision-making strategies

What are ***the common aspects*** in these decision-making strategies?

1. Motivation

Main Topics in Decision Makings

What type of decision making framework will be used?

- Single stage or multi stages
- Single decision maker or many decision makers
- Model based or model-free

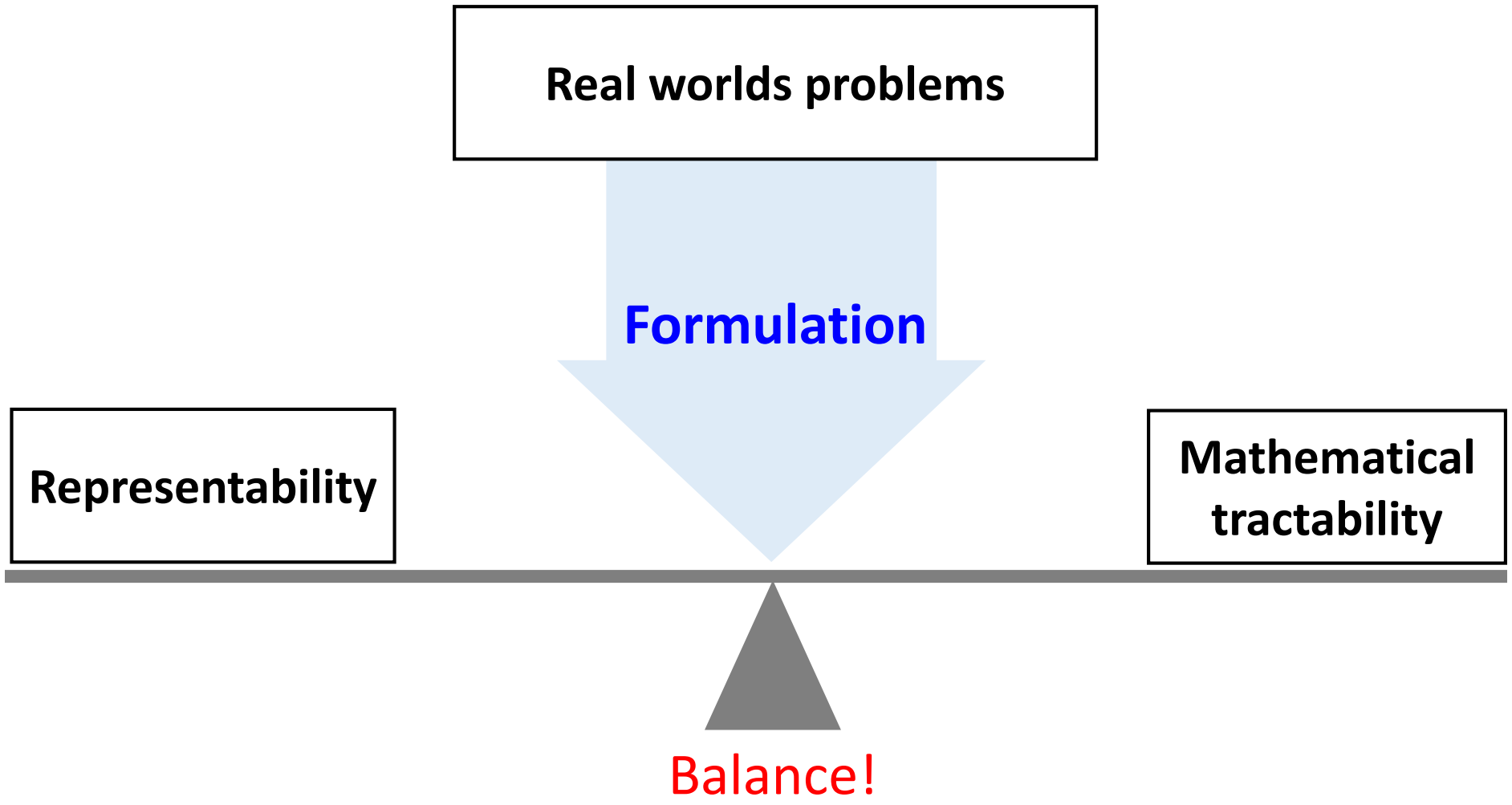
“Decision makings under uncertainties”

How to model uncertainties?

- **Epistemic Uncertainty** (systemic uncertainty) :
  Uncertainty arising through lack of knowledge
  - Model uncertainty
  - State uncertainty
- **Aleatoric uncertainty** (statistical uncertainty):
  Uncertainty arising through an underlying stochastic system

1. Motivation

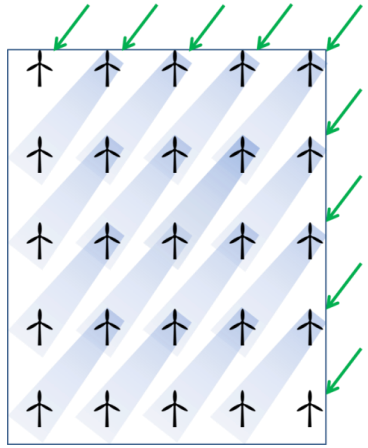
How to Solve Complex Problems?

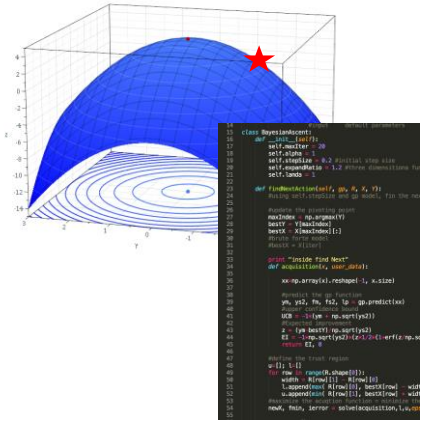


1. Motivation

How to Solve Complex Problems?

Real-world task → **Modeling** → **Formal task (model)** → **Solving** → **Algorithm (program)**





$$\underset{\boldsymbol{x}}{\text{maximize}} \sum_{i=1}^{N} P_i(\boldsymbol{x}; \theta, U)$$

Validation

Are we building the right model?

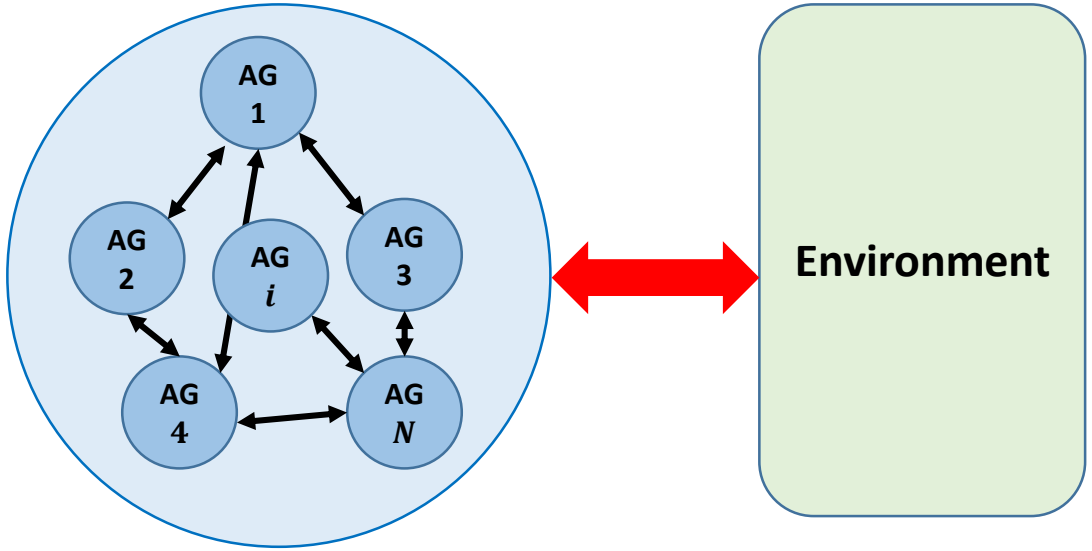
Verification

Does the algorithm capture all the essential aspects of the model?

Is this solution is good for the target system?

Data can help model more realistically and derive more accurate solution!

1. Motivation

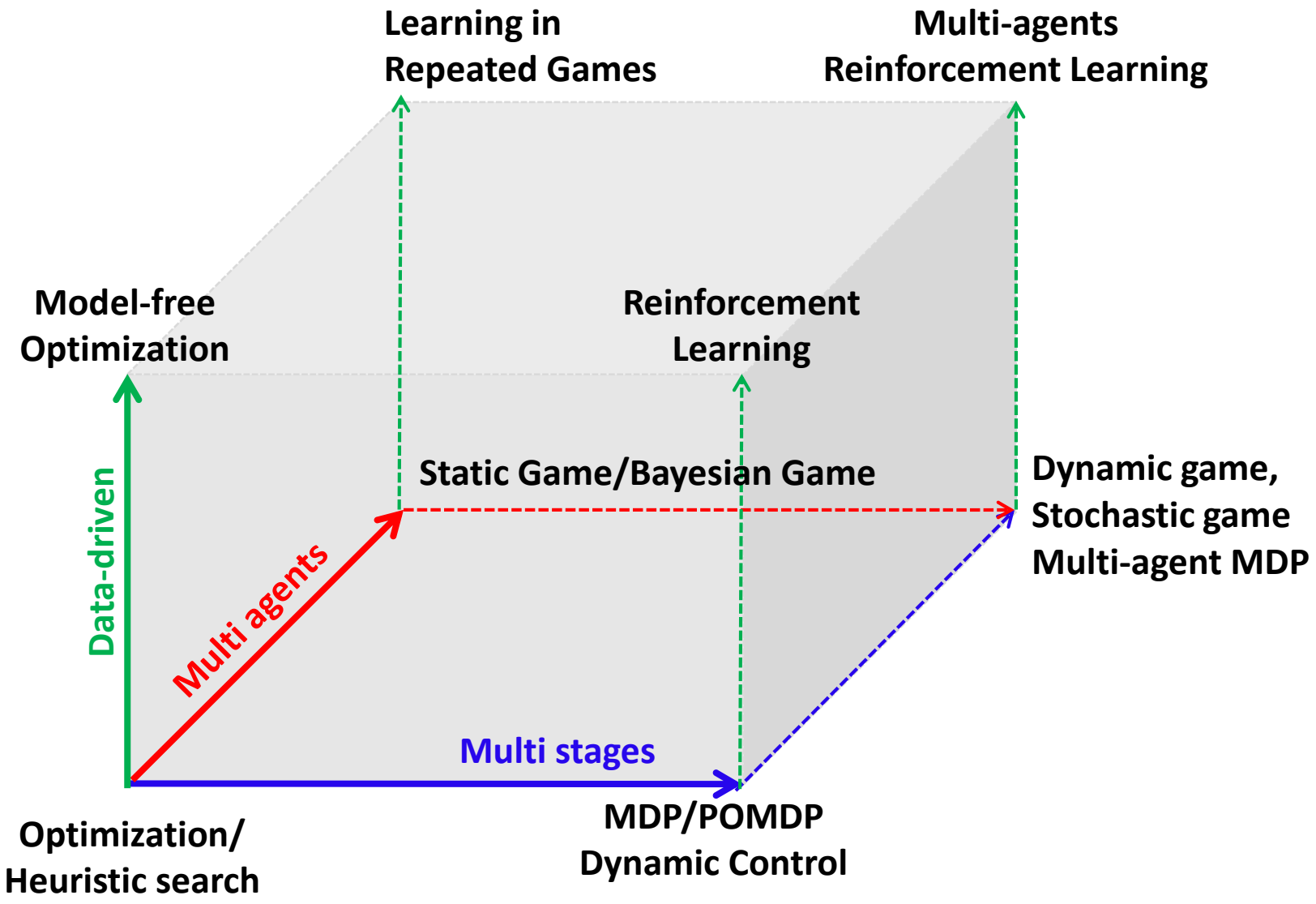


Joint control policy approximated as a decentralized control policy:

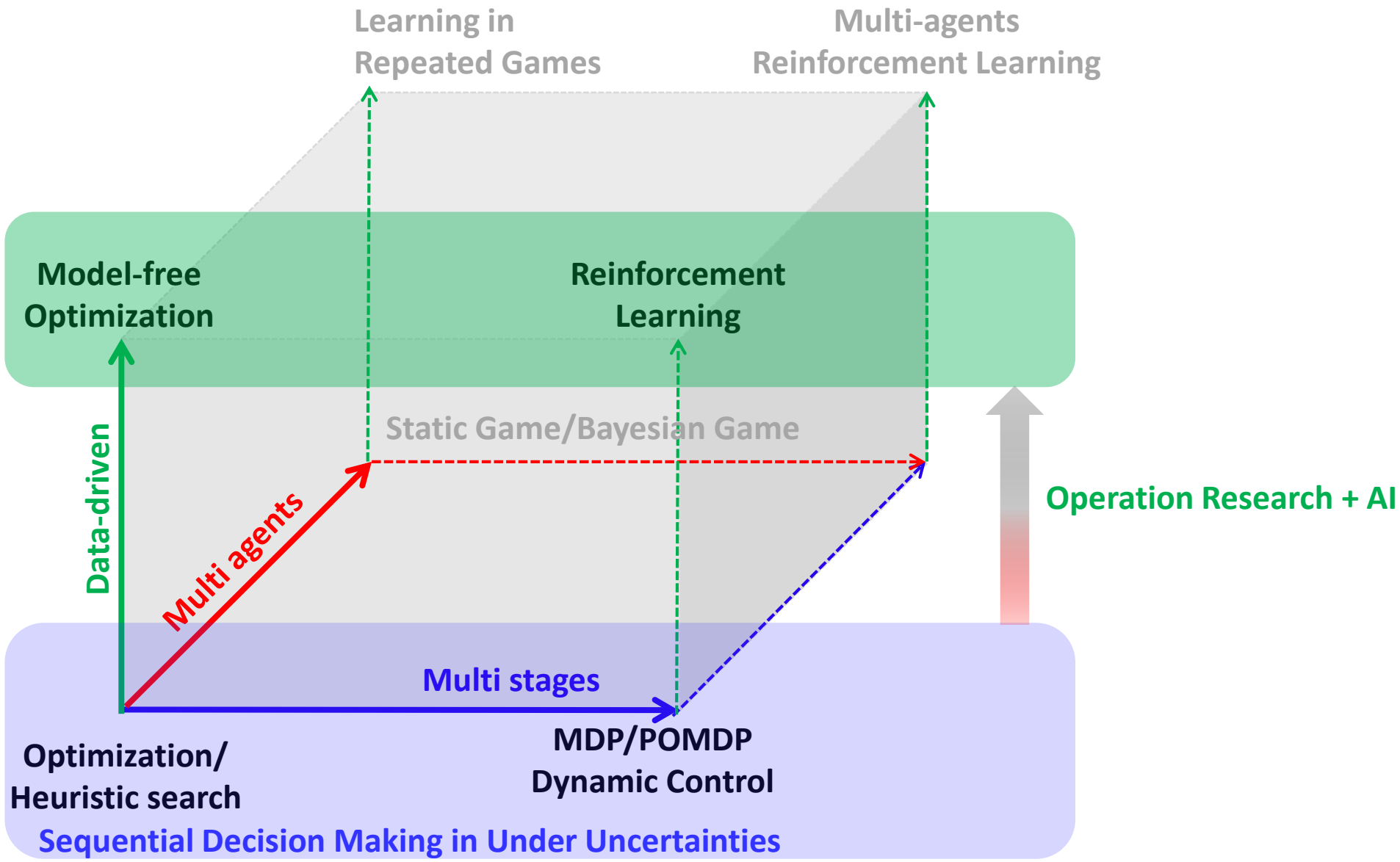
$$\pi(s, a_1, \dots, a_i, \dots, a_n) \approx \prod_{i=1}^{N} \pi_i(s, a_i) \approx \prod_{i=1}^{N} \pi_i(o_i, a_i)$$

- As a system becomes larger and more complex, it become more difficult to understand and control the target systems
- A methodology has been developed to independently model the agents that make up the entire system, to efficiently model the entire system considering their interaction, and to derive distributed control strategies
  - Decentralized MDP, Decentralized-POMDP, Decentralized Cooperative Control, Team Game, Cooperative Game..
  - Analytical solutions to these problems are limited to only special cases
- Recent advanced in Deep learning and Reinforcement learning approach can open up new solution approaches

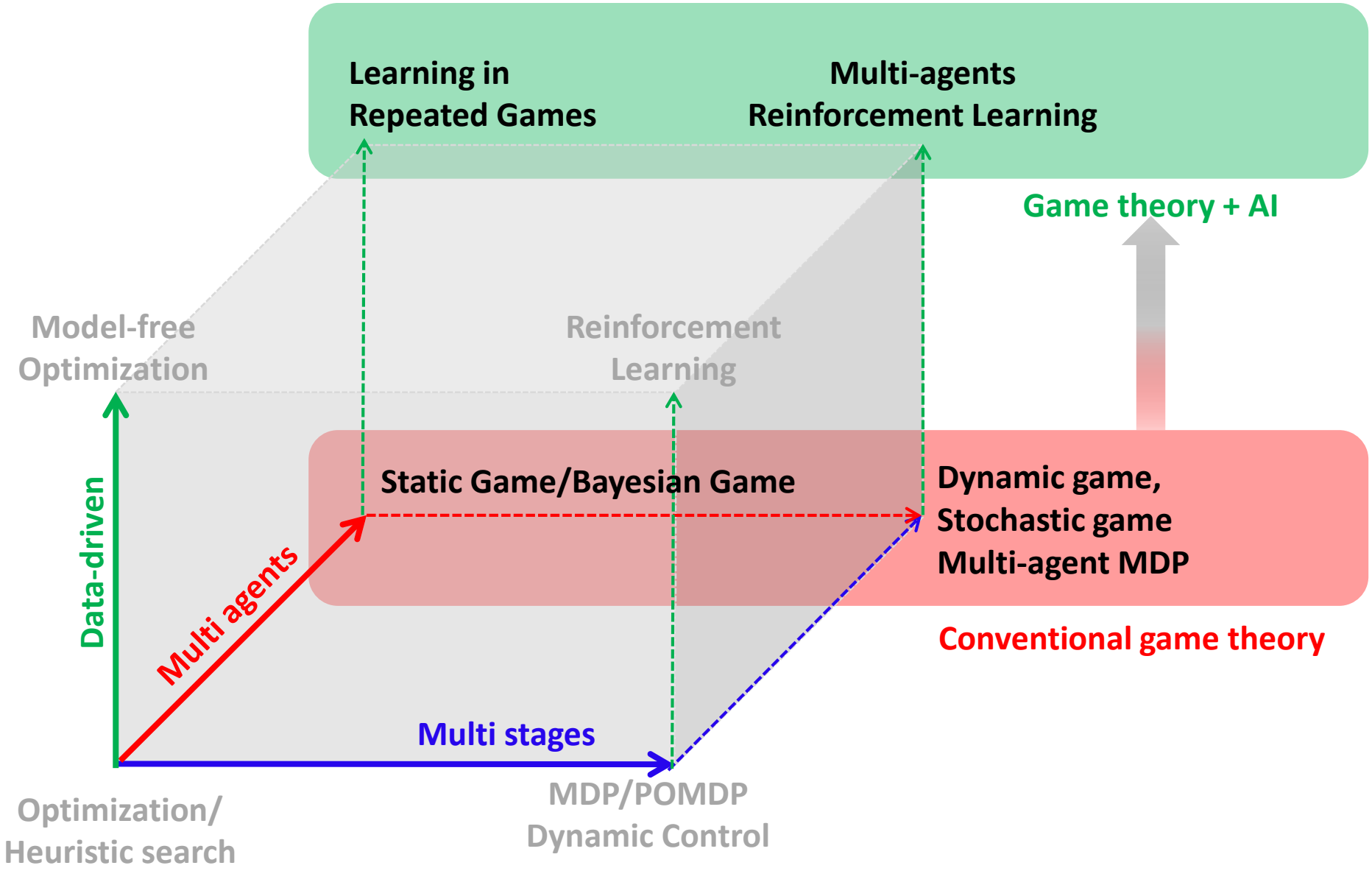
2. OVERVIEW


Learning in
Repeated Games
Multi-agents
Reinforcement Learning
Model-free
Optimization
Reinforcement
Learning
Static Game/Bayesian Game
Dynamic game,
Stochastic game
Multi-agent MDP
Data-driven
Multi agents
Multi stages
Optimization/
Heuristic search
MDP/POMDP
Dynamic Control

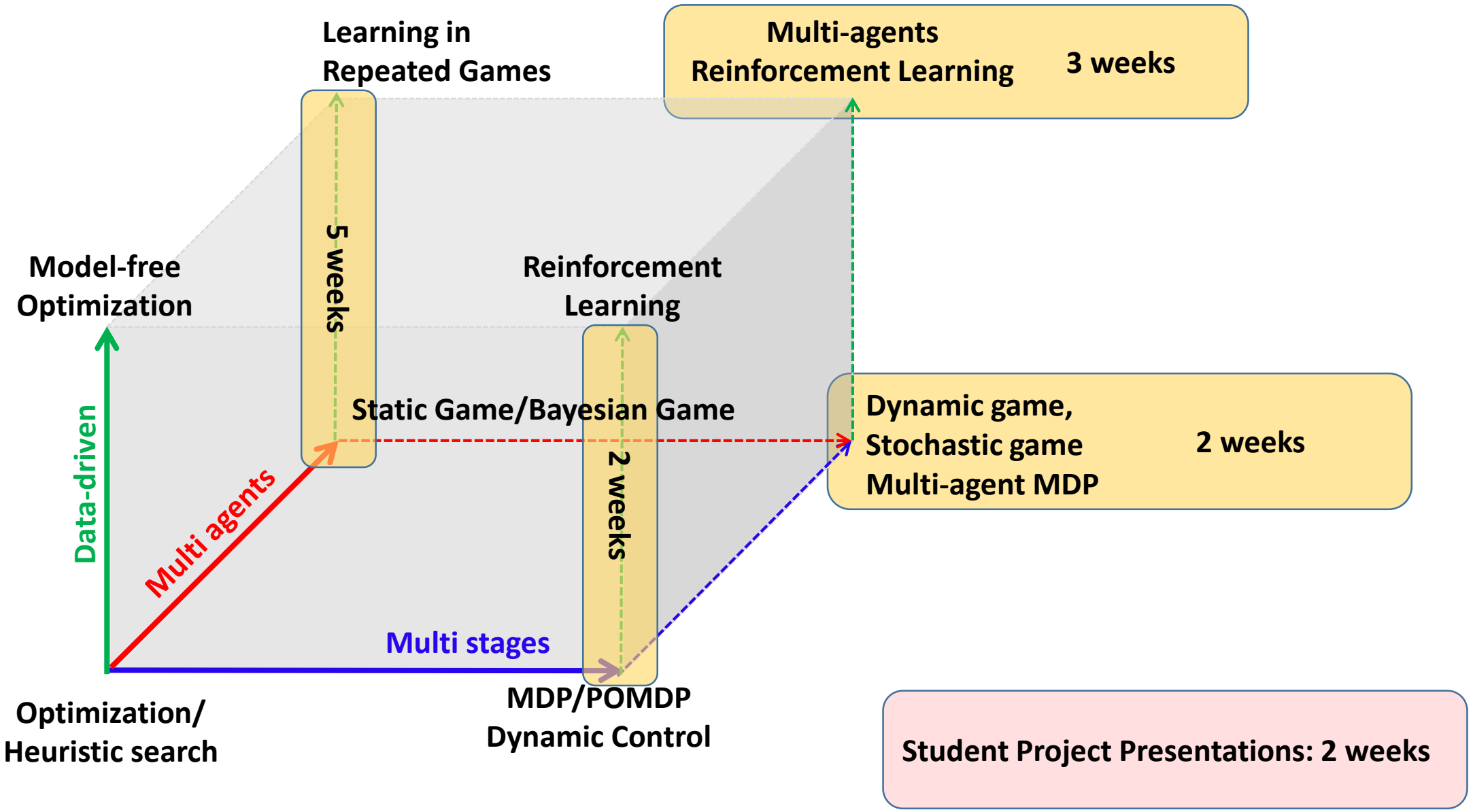
2. OVERVIEW


Learning in
Repeated Games
Multi-agents
Reinforcement Learning
Model-free
Optimization
Reinforcement
Learning
Static Game/Bayesian Game
Data-driven
Multi agents
Operation Research + AI
Multi stages
Optimization/
Heuristic search
MDP/POMDP
Dynamic Control
Sequential Decision Making in Under Uncertainties

2. OVERVIEW

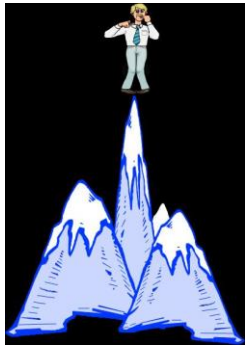

Learning in
Repeated Games
Multi-agents
Reinforcement Learning
Game theory + AI
Model-free
Optimization
Reinforcement
Learning
Static Game/Bayesian Game
Dynamic game,
Stochastic game
Multi-agent MDP
Conventional game theory
Data-driven
Multi agents
Multi stages
Optimization/
Heuristic search
MDP/POMDP
Dynamic Control

2. OVERVIEW


Learning in
Repeated Games
Multi-agents
Reinforcement Learning
3 weeks
5 weeks
Model-free
Optimization
Reinforcement
Learning
Static Game/Bayesian Game
Dynamic game,
Stochastic game
Multi-agent MDP
2 weeks
2 weeks
Data-driven
Multi agents
Multi stages
Optimization/
Heuristic search
MDP/POMDP
Dynamic Control
Student Project Presentations: 2 weeks

2. OVERVIEW : Static Optimization to Static Game

From Optimality to Equilibrium



- **Single agent decision making:**
  - Optimal strategy is one that maximizes the agent's expected utility for a given environment
  - Uncertainties arose from stochastic environment, partially observable states, uncertain rewards, etc., which can be dealt with probability concepts.

$$a^* = \underset{a}{\operatorname{argmax}}\, E_s[u(a, s)]$$

- **Multiagents decision making:**
  - The environment includes other agents, each of which tries to maximize its own utility
  - Thus the notion of an optimal strategy for a given agent is not meaningful because *the best strategy depends on the choices of others*
  - We need to identify certain subsets of outcomes, called solution concepts
  - Two of the most fundamental solution concepts are
    - *Pareto optimality*
    - *Nash equilibrium*

$$u_1(a_1^*, a_2^*) \geq u_1(a_1, a_2^*) \;\; \forall a_1$$
$$u_2(a_1^*, a_2^*) \geq u_2(a_1^*, a_2) \;\; \forall a_2$$

2. OVERVIEW : Static Optimization to Static Game

Nash Equilibrium

Definition (Nash Equilibrium)

A strategy profile $s^* = (s_1^*, \dots, s_n^*)$ is a Nash Equilibrium if, for all agents i and for all strategies s_i, $u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$.

- A strategy profile $s^* = (s_1^*, \dots, s_n^*)$ is a Nash Equilibrium if, for all agents i, s_i^* is a best response to s_{-i}^*, i.e., $s_i^* \in BR(s_{-i}^*)$
- A Nash equilibrium is a stable strategy profile:
  - no agent would want to change his strategy if he knew what strategies the other agents were following

2. OVERVIEW : Static Optimization to Static Game

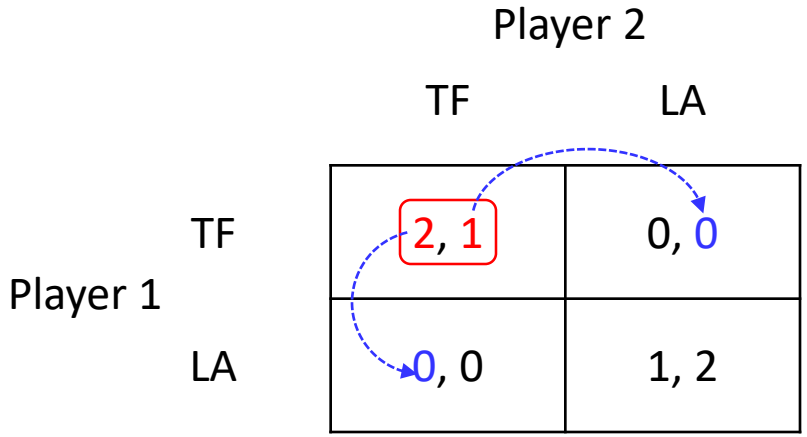
Nash Equilibrium

		Player 2: TF	Player 2: LA
Player 1	TF	2, 1	0, 0
Player 1	LA	0, 0	1, 2

- We immediately see that it has two pure-strategy Nash equilibria

2. OVERVIEW : Static Optimization to Static Game

Nash Equilibrium

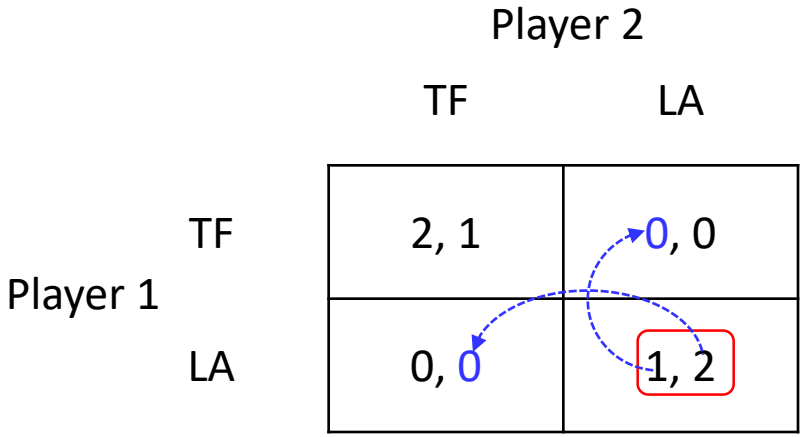


- We can check that these are Nash equilibria by confirming that whenever one of the players play the given (pure) strategy, the other player would only lose by deviating

$$a^* = (\text{TF}, \text{TF}) \qquad \begin{aligned} u_1(\text{TF}, \text{TF}) &> u_1(\text{LA}, \text{TF}) \\ u_2(\text{TF}, \text{TF}) &> u_2(\text{TF}, \text{LA}) \end{aligned}$$

2. OVERVIEW : Static Optimization to Static Game

Nash Equilibrium



- We can check that these are Nash equilibria by confirming that whenever one of the players play the given (pure) strategy, the other player would only lose by deviating

$$a^* = (\text{LA}, \text{LA}) \qquad \begin{aligned} u_1(\text{LA}, \text{LA}) &> u_1(\text{TF}, \text{LA}) \\ u_2(\text{LA}, \text{LA}) &> u_2(\text{LA}, \text{TF}) \end{aligned}$$

2. OVERVIEW : Static Optimization to Static Game

Max-Min Equilibrium

Definition (Maxmin)

The maxmin strategy for player i is $s_i^* = \arg\max_{s_i} \min_{s_{-i}} u_i(s_i, s_{-i})$ and the maxmin value for player i is $\max_{s_i} \min_{s_{-i}} u_i(s_i, s_{-i})$

- The ***maxmin strategy*** of player i in an n-players game is a strategy that maximizes $i's$ worst –case payoff, in the situation where all the others players happen to play the strategies which cause the greatest harm to i
- The ***maxmin strategy*** is a sensible choice for a conservative agent who wants to maximize his expected utility without having to make any assumptions about the other agents
- The ***maxmin value*** (or security level) of the game for player i is that minimum amount of payoff guaranteed by a maxmin strategy
- It is strategy that **defends against** other agents (defensive strategy)

2. OVERVIEW : Static Optimization to Static Game

Max-Min Equilibrium

Definition (Minmax, two-player**)**

In an two-player game, the *minmax strategy* for player i against player $-i$ is $s_i^* = \arg\min_{s_i} \max_{s_{-i}} u_{-i}(s_i, s_{-i})$ and the minmax value is $\min_{s_i} \max_{s_{-i}} u_{-i}(s_i, s_{-i})$

- The *minnmax strategy* of player i in an two-players game is a strategy that keeps the maximum payoff of $-i$ at a minimum
- The *minmax value* of player $-i$ is that minimum
- It is strategy that attack against other agents (offensive strategy)

2. OVERVIEW : Static Optimization to Static Game

Max-Min Equilibrium

In agent $i's$ perspective

$$\max_{s_i} \min_{s_{-i}} u_i(s_i, s_{-i})$$

- Agent always maximizes its payoff
- Defensive strategy (if max is first)

$$\min_{s_i} \max_{s_{-i}} u_{-i}(s_i, s_{-i})$$

- Agent always maximizes its payoff
- offensive strategy (if min is first)

2. OVERVIEW : Static Optimization to Static Game

Correlated Equilibrium

Definition (Correlated equilibrium)

A correlated equilibrium of finite game is a joint probability distribution $\pi \in \Delta(A)$ such that if R is random variable distributed according to π then

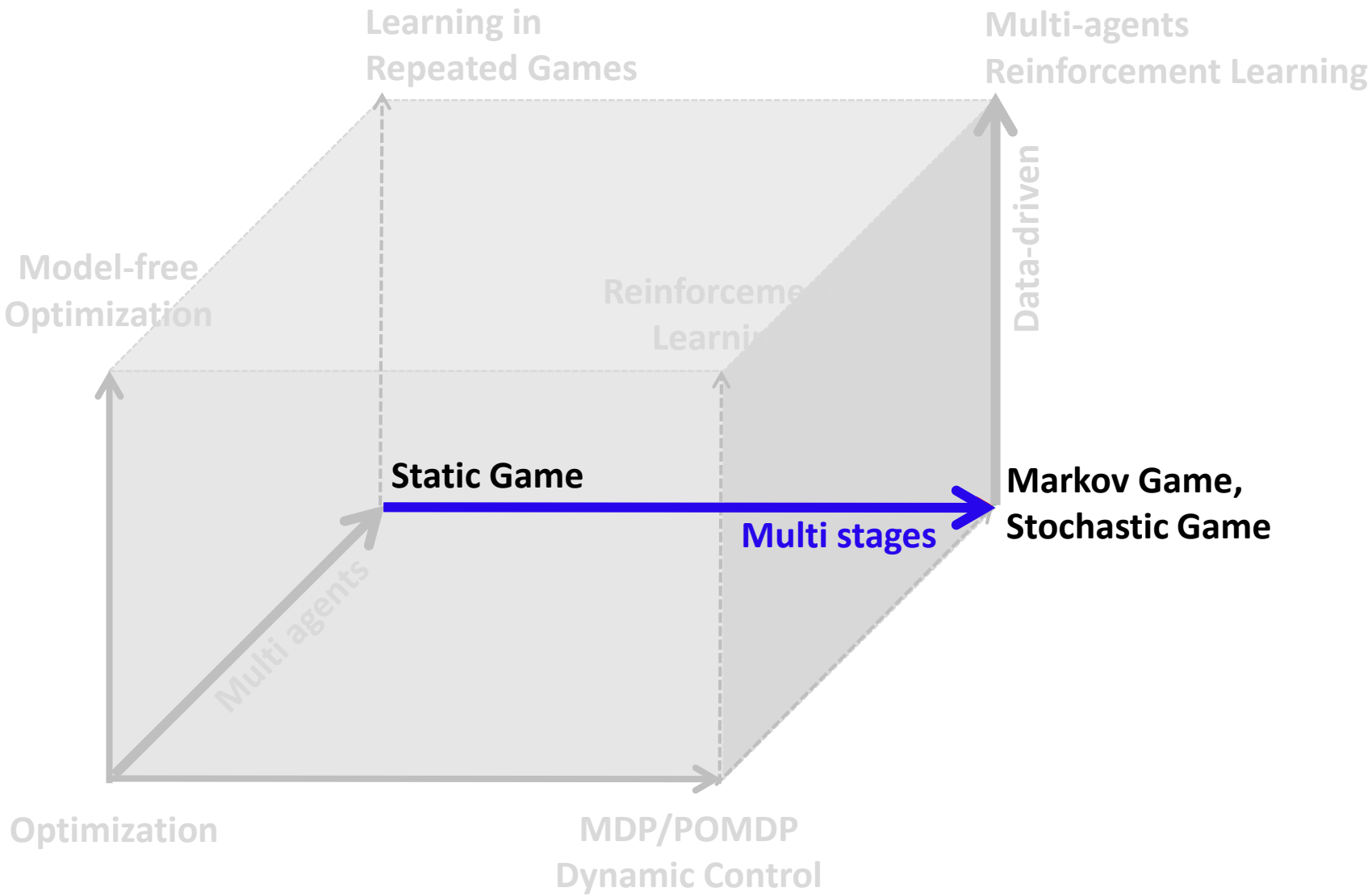
$$\sum_{a_{-i} \in A_{-i}} \text{Prob}(R = a | R_i = a_i) u_i(a_i, a_{-i}) \geq \sum_{a_{-i} \in A_{-i}} \text{Prob}(R = a | R_i = a_i) u_i(a_i', a_{-i})$$

For all players i, all $a_i \in A_i$ such that $\text{Prob}(R_i = a_i) > 0$, and all $a_i' \in A_i$

- A distribution π is defined to be a correlated equilibrium if no player can ever expect to unilaterally gain by deviating from his recommendation, assuming the other players play according to their recommendations.

  - a_i is a recommendation by R drawn from $\pi \in \Delta(A)$
  - a_i' is a deviation from this recommendation

2. OVERVIEW : Static Game to Dynamic Game



2. OVERVIEW : Static Game to Dynamic Game

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	**Dynamic Optimization**	Dynamic Game

Action space

Time space	**Model based**	Finite	Infinite
	Discrete	Discrete time MDP $P(s_{t+1}\|s_t, a_t)$	Discrete-time dynamic system $x_{t+1} = f(x_t, u_t)$
	Continuous	Continuous time MDP $P(s_{t+h}\|s_t, a_t)$	Continuous-time dynamic system $\dot{x}_t = f(x_t, u_t)$

2. OVERVIEW : Static Game to Dynamic Game

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	**Dynamic Optimization**	Dynamic Game

Action space

Time space	**Model free**	Finite	Infinite
	Discrete	Value-based Reinforcement Learning	Policy-based Reinforcement Learning
	Continuous		

2. OVERVIEW : Static Game to Dynamic Game

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	**Dynamic Game**

Action space

Time space	**Model based**	Finite	Infinite
	Discrete	Markov Game (Stochastic Game)	DT Infinite dynamic game (Stochastic Game)
	Continuous	Continuous time Markov Game	CT-time Infinite dynamic game (differential game)

2. OVERVIEW : Static Game to Dynamic Game

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	**Dynamic Game**

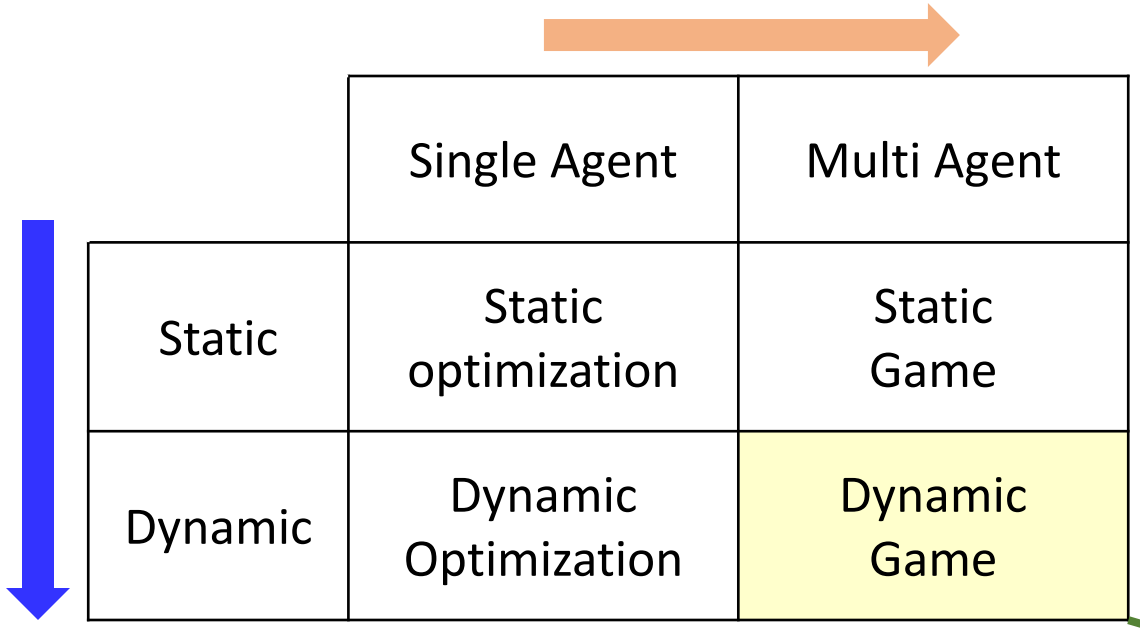
Action space

Time space	**Model free**	Finite	Infinite
	Discrete	Multi-Agent Value-based RL	Multi-Agent Policy-based RL
	Continuous		

2. OVERVIEW : Static Game to Dynamic Game

Equilibrium concept:

-Nash; Zero-sum; Stackelberg; Correlated



	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	Dynamic Game

Dynamic optimization as a static optimization concept:

-Minimum principle (necessary condition)

-Dynamic programming principle (sufficient condition)

-Need to specify information structure

$$
\begin{aligned}
&L^{1*} \triangleq L^1(u^{1*}; u^{2*}; \dots; u^{N*}) \le L^1(u^1; u^{2*}; \dots; u^{N*}),\\
&L^{2*} \triangleq L^2(u^{1*}; u^{2*}; \dots; u^{N*}) \le L^2(u^{1*}; u^2; \dots; u^{N*}),\\
&\dots\\
&L^{N*} \triangleq L^N(u^{1*}; u^{2*}; \dots; u^{N*}) \le L^N(u^{1*}; u^{2*}; \dots; u^{N*})
\end{aligned}
$$

(Think in normal form game setting)

2. OVERVIEW : Static Game to Dynamic Game

Equilibrium concept:

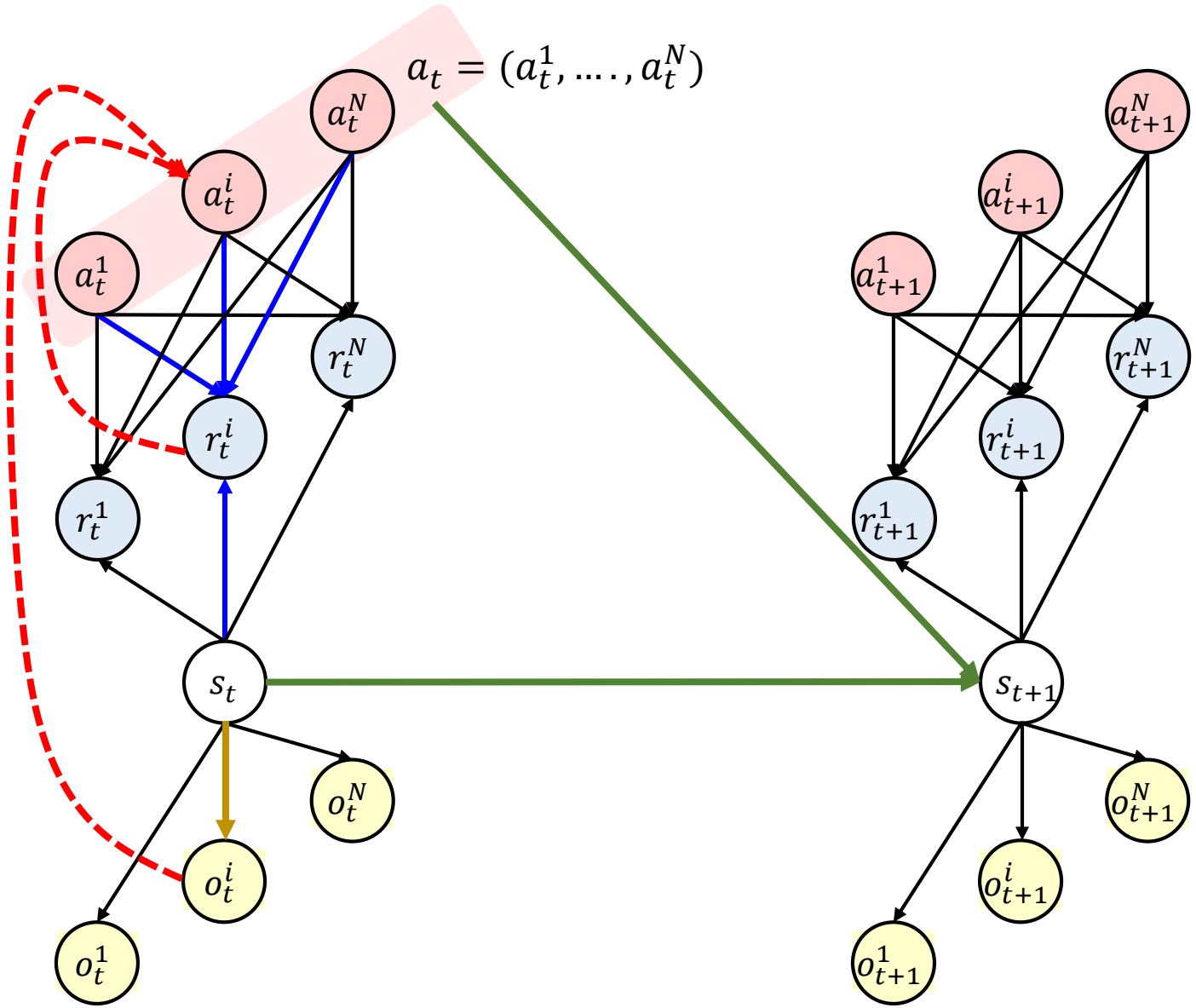
(Dynamic) Information structure	Nash	Zero-sum	Stackelberg
Open-loop (perfect state)	Open-loop Nash-Strategy	Open-loop Zero-sum Strategy	
Feedback (perfect state)	Feedback Nash-Strategy	Feedback Zero-sum Strategy	
⋮			

- We need to specify ***information structure***
  - ✓ Open-loop vs. close-loop (feedback)
  - ✓ Perfect vs. imperfect
- We need to ***equilibrium concept***
  - ✓ Nash, Zero-sum, Stackelberg, Correlated,…

Equilibrium concept + information structure → solution method

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Definition



- ***Transition:***

$$P(s_{t+1}|a_t^1, \ldots, a_t^N, s_t)$$

- ***Reward:***

$$r_i(s_t, a_t^1, \ldots, a_t^N)$$

- ***Observation:***

$$o_t^i = h^i(s_t)$$

- ***Decentralized Policy:***

$$\pi(s_t, a_t^1, \ldots, a_t^N) \approx \prod_{i=1}^{N} \pi_i(s_t, a_t^i)$$

$$\approx \prod_{i=1}^{N} \pi_i(o_t^i, a_t^i)$$

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Definition

Definition (Stochastic game)

A stochastic game is a tuple (N, S, A, R, T), where

- N is a finite set of n players
- S is a finite set of states (stage games),
- $A = A_1 \times \cdots \times A_n$, where A_i is a finite set of actions available to player i,
- $T : S \times A \times S \longmapsto [0,1]$ is the transition probability function; $T(s, a, s')$ is the probability of transitioning from state s to state s' after joint action a,
- $R = r_1 \ldots, r_n$, where $r_i : S \times A \longmapsto \mathbb{R}$ is a real-valued payoff function for player i

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Definition : Transition

- All agents $(1, \dots, n)$ share the joint state s
- The transition equation is similar to the Markov Decision Process decision transition:

MDP : $$\sum_{s'} T(s, a, s') = \sum_{s'} p(s'|a, s) = 1 \,, \forall s \in S, \forall a \in A$$

SG: $$\sum_{s'} T(s, a_1, \dots, a_i, \dots, a_n, s') = \sum_{s'} p(s'|a_1, \dots, a_i, \dots, a_n, s) = 1$$

$$\forall s \in S, \forall a_i \in A_i, i = (1, \dots, n)$$

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Definition : Reward

- Reward function r_i for agent i depends on the current joint state s, the joint action $a = (a_1, \dots, a_n)$, and the next joint future state s'

MDP : $r(s, a, s')$

SG: $r_i(s, a_1, \dots, a_i, \dots, a_n, s')$

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Definition : State Value

- As we did in MDP, we can define value function
- Let π_i be the policy of player $i \in N$. For a given initial state s, the value of state s for player i is defined as

$$V_i(s, \pi_1, \dots, \pi_i, \dots, \pi_n) = \sum_{t=0}^{\infty} \gamma^t E\left[r_{i,t} | \pi_1, \dots, \pi_i, \dots, \pi_n, s_0 = s\right]$$

  - The accumulated rewards depends on the policies of other agents
  - The immediate reward is expressed as expected value, because some policy π_i can be stochastic

- In a *discounted stochastic game*, the objective of each player is to maximize the discounted sum of rewards, with discount factor $\gamma \in [0,1)$.

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Definition : Nash Equilibrium Strategy

Definition (Nash equilibrium policy in Stochastic game)

In a stochastic game $\Gamma = (N, S, A, R, T)$, a Nash equilibrium policy is a tuple of n policies $\pi^* = (\pi_1^*, \ldots, \pi_n^*)$ such that for all $s \in S$ and $i = 1, \ldots n$,

$$V_i(s, \pi_1^*, \ldots, \pi_i^*, \ldots, \pi_n^*) \geq V_i(s, \pi_1^*, \ldots, \pi_i, \ldots, \pi_n^*) \text{ for all } \pi_i \in \Pi_i$$

- A Nash equilibrium is a joint policy where each agent's policy is a best response to the others
- For a stochastic game, each agent's policy is defined over the entire time horizon of the game
- **A Nash equilibrium state value** $V_i(s, \pi_1^*, \ldots, \pi_n^*)$is defined as the sum of discounted rewards when all agents following the Nash equilibrium policies $\pi^* = (\pi_1^*, \ldots, \pi_n^*)$
  - Notations: $V_i^*(s) = V_i^{\pi^*}(s) = V_i(s, \pi_1^*, \ldots, \pi_n^*)$

2. OVERVIEW : Static Game to Dynamic Game

Single Agent Case

Q-values

$Q^{\pi}(s,a)$: The expected utility of taking action a from state s, and then following policy π

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty}\gamma^{k} r_{t+k} \,\middle|\, S_t = s, A_t = a\right)$$

Optimal Q-values

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$$= \max_{\pi} \mathbb{E}[r(s,a,s') + \gamma V^{\pi}(s') | s_t = s, a_t = a]$$

$$= \mathbb{E}\left[r(s,a,s') + \gamma \max_{\pi} V^{\pi}(s') \,|s_t = s, a_t = a\right]$$

$$= \mathbb{E}[r(s,a,s') + \gamma V^{*}(s') | s_t = s, a_t = a] \qquad \because V^{*}(s') \equiv \max_{\pi} V^{\pi}(s')$$

$$= \mathbb{E}\left[r(s,a,s') + \gamma \max_{a'} Q^{*}(s',a') \,|s_t = s, a_t = a\right] \qquad \because V^{*}(s') \equiv \max_{a'} Q^{*}(s',a')$$

Optimization over policy becomes greedy optimization over action!

- Optimal Q-value for a single-agent is the sum of the current reward and future discounted rewards when playing the optimal strategy from the next period onward

2. OVERVIEW : Static Game to Dynamic Game

Multi Agent Case

Q-values for agent i

$Q_i^{\pi}(s, a_1, \dots, a_n)$: The expected utility of taking joint action $(a_1, \dots, a_n)$ from state s, and then following policy π

$$Q_i^{\pi}(s, a_1, \dots, a_n) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{i,t+k} \,\middle|\, S_t = s, A = (a_1, \dots, a_n)\right)$$

Optimal Q-values for agent i

$$Q_i^*(s, a_1, \dots, a_n) = \max_{\pi_1, \dots, \pi_n} Q_i^{\pi}(s, a_1, \dots, a_n)$$

$$= \max_{\pi_1, \dots, \pi_n} \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1, \dots, \pi_n) | s_t = s, a_t = (a_1, \dots, a_n)]$$

$$= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \max_{\pi_1, \dots, \pi_n} V_i(s', \pi_1, \dots, \pi_n) | s_t = s, a_t = (a_1, \dots, a_n)\right]$$

$$= \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = (a_1, \dots, a_n)]$$

$$= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \max_{a_1, \dots, a_n} Q_i^*(s', a_1, \dots, a_n) \,|s_t = s, a_t = (a_1, \dots, a_n)\right]$$

- Optimal Q-value for agent i occurs when all agents are jointly coordinating to maximize agent i's accumulated reward
  - ➢ Rarely occurs! : Optimal Q-values for all agents are not achieved simultaneously

2. OVERVIEW : Static Game to Dynamic Game

Multi Agent Case

Q-values for agent i

$Q_i^{\pi}(s, a_1, \dots, a_n)$: The expected utility of taking joint action $(a_1, \dots, a_n)$ from state s, and then following policy π

$$Q_i^{\pi}(s, a_1, \dots, a_n) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{i,t+k} \,\middle|\, S_t = s, A = (a_1, \dots, a_n)\right)$$

Optimal Q-values for agent i

$$Q_i^*(s, a_1, \dots, a_n) = \max_{\pi_1, \dots, \pi_n} Q_i^{\pi}(s, a_1, \dots, a_n)$$

$$= \max_{\pi_1, \dots, \pi_n} \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1, \dots, \pi_n) | s_t = s, a_t = (a_1, \dots, a_n)]$$

$$= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \max_{\pi_1, \dots, \pi_n} V_i(s', \pi_1, \dots, \pi_n) | s_t = s, a_t = (a_1, \dots, a_n)\right]$$

$$= \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = (a_1, \dots, a_n)]$$

$$= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \max_{a_1, \dots, a_n} Q_i^*(s', a_1, \dots, a_n) | s_t = s, a_t = (a_1, \dots, a_n)\right]$$

- Optimal Q-value for agent i occurs when all agents are jointly coordinating to maximize agent i's accumulated reward
  - ➢ Rarely occurs! : Optimal Q-values for all agents are not achieved simultaneously

2. OVERVIEW : Static Game to Dynamic Game

Multi Agent Case

Q-values for agent i

$Q_i^{\pi}(s, a_1, \dots, a_n)$: The expected utility of taking joint action $(a_1, \dots, a_n)$ from state s, and then following policy π

$$Q_i^{\pi}(s, a_1, \dots, a_n) = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{i,t+k} \,\middle|\, S_t = s, A = (a_1, \dots, a_n)\right)$$

Nash Q-values for agent i

$$Q_i^*(s, a_1, \dots, a_n) = \underset{\pi_1,\dots,\pi_n}{\text{Nash}}\ Q_i^{\pi}(s, a_1, \dots, a_n)$$

$$= \underset{\pi_1,\dots,\pi_n}{\text{Nash}}\ \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1, \dots, \pi_n) | s_t = s, a_t = (a_1, \dots, a_n)]$$

$$= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \underset{\pi_1,\dots,\pi_n}{\text{Nash}}\ V_i(s', \pi_1, \dots, \pi_n) | s_t = s, a_t = (a_1, \dots, a_n)\right]$$

$$= \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = (a_1, \dots, a_n)]$$

$$= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \underset{a_1,\dots,a_n}{\text{Nash}}\ Q_i^*(s', a_1, \dots, a_n) \,|\, s_t = s, a_t = (a_1, \dots, a_n)\right]$$

Equilibrium over policies becomes stage game equilibrium over action!

- A **Nash Q value** $Q_i^*(s, a_1, \dots, a_n)$ is the expected sum of discounted rewards when all agents take the joint action $a = (a_1, \dots, a_n)$ at given state s and follow a Nash equilibrium strategy $\pi^* = (\pi_1^*, \dots, \pi_n^*)$

2. OVERVIEW : Static Game to Dynamic Game

Nash Bellman Equation

For single agent:

$$V^*(s') = \max_a Q^*(s', a)$$

$$Q^*(s, a) = \mathbb{E}[r(s, a, s') + \gamma V^*(s') | s_t = s, a_t = a]$$

$$= \mathbb{E}\left[r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \, | s_t = s, a_t = a\right]$$

For multiple agents:

$$V_i(s', \pi_1^*, \dots, \pi_n^*) = \underset{a_1, \dots, a_n}{\text{Nash}} \, Q_i^*(s', a_1, \dots, a_n)$$

$$Q_i^*(s, a_1, \dots, a_n) = \mathbb{E}[r(s, a, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = a]$$

$$= \mathbb{E}\left[r(s, a, s') + \gamma \underset{a_1, \dots, a_n}{\text{Nash}} \, Q_i^*(s', a_1, \dots, a_n) \, | s_t = s, a_t = (a_1, \dots, a_n)\right]$$

2. OVERVIEW : Static Game to Dynamic Game

Nash Bellman Equation

For multiple agents:

$$V_i(s', \pi_1^*, \dots, \pi_n^*) = \underset{a_1,\dots,a_n}{\text{Nash}}\ Q_i^*(s', a_1, \dots, a_n)$$

$$Q_i^*(s, a_1, \dots, a_n) = \mathbb{E}[r(s, a, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = a]$$

$$= \mathbb{E}\left[r(s, a, s') + \gamma \underset{a_1,\dots,a_n}{\text{Nash}}\ Q_i^*(s', a_1, \dots, a_n) \,|\, s_t = s, a_t = (a_1, \dots, a_n)\right]$$

- Nash **equilibrium** Q value $\underset{a_1,\dots,a_n}{\text{Nash}}\ Q_i^*(s', a_1, \dots, a_n)$ can be computed by computing player ith Nash equilibrium value for the stage game $[Q_i^*(s', a_1, \dots, a_n), \dots, Q_n^*(s', a_1, \dots, a_n)]$

  - for example when $i = 1,2$

	a_2^1	a_2^2
a_1^1	$Q_1^*(s', a_1^1, a_2^1), Q_2(s', a_1^1, a_2^1)$	$Q_1^*(s', a_1^1, a_2^2), Q_2(s', a_1^1, a_2^2)$
a_1^2	$Q_1^*(s', a_1^2, a_2^1), Q_2(s', a_1^2, a_2^1)$	$Q_1^*(s', a_1^2, a_2^2), Q_2(s', a_1^2, a_2^2)$

Nash equilibrium

2. OVERVIEW : Static Game to Dynamic Game

Nash Bellman Equation (simple notation)

$$r_i(s, a_1, \dots, a_n, s') \rightarrow r_i(s, \vec{a}, s')$$

$$V_i(s, \pi_1^*, \dots, \pi_n^*) \rightarrow V_i^*(s)$$

$$Q_i^*(s, a_1, \dots, a_n) \rightarrow Q_i^*(s', \vec{a}\,)$$

$$\begin{aligned} Q_i^*(s, a_1, \dots, a_n) &= \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = (a_1, \dots, a_n)] \\ &= \mathbb{E}\left[r_i(s, a_1, \dots, a_n, s') + \gamma \underset{a_1,\dots,a_n}{\text{Nash}}\, Q_i^*(s', a_1, \dots, a_n) \,|s_t = s, a_t = (a_1, \dots, a_n)\right] \end{aligned}$$

$$\begin{aligned} Q_i^*(s', \vec{a}\,) &= \mathbb{E}[r_i(s, \vec{a}, s') + \gamma V_i^*(s') | s_t = s, a_t = \vec{a}] \\ &= \mathbb{E}[r_i(s, \vec{a}, s') + \gamma\, \text{Nash}\, Q_i^*(s') \,|s_t = s, a_t = \vec{a}] \end{aligned}$$

$$\underset{a_1,\dots,a_n}{\text{Nash}}\, Q_i^*(s', a_1, \dots, a_n) = Q_i^*(s', \vec{a}_{NE}) = \text{Nash}\, Q_i^*(s')$$

2. OVERVIEW : Static Game to Dynamic Game

Nash Bellman Equation (simple notation)

- If we know **Nash equilibrium policy** $\pi^* = (\pi_1^*, \dots, \pi_n^*)$, we can compute the Nash equilibrium state values $V_i(s, \pi_1^*, \dots, \pi_n^*)$ (i.e., policy evaluation)

$$V_i(s, \pi_1^*, \dots, \pi_n^*) = \sum_{t=0}^{\infty} \gamma^t E\left[r_{i,t} | \pi_1^*, \dots, \pi_n^*, s_0 = s\right]$$

- If we know **Nash equilibrium state value** $V_i(s, \pi_1^*, \dots, \pi_n^*)$ and transition models $p(s'|s, a_1, \dots, a_n)$, we can compute **Nash Q-values (i.e., Nash Q-function)** using backward induction (analytical approach)

$$\begin{aligned} Q_i^*(s, a_1, \dots, a_n) &= \mathbb{E}[r_i(s, a_1, \dots, a_n, s') + \gamma V_i(s', \pi_1^*, \dots, \pi_n^*) | s_t = s, a_t = (a_1, \dots, a_n)] \\ &= r_i(s, a_1, \dots, a_n, s') + \sum_{s'} p(s'|s, a_1, \dots, a_n) V_i(s', \pi_1^*, \dots, \pi_n^*) \end{aligned}$$

2. OVERVIEW : Static Game to Dynamic Game

Nash Bellman Equation for State Action Value

Definition (Optimal Q-function)

Optimal Q function is defined as

$$Q^*(s,a) = r(s,a,s') + \gamma \sum_{s' \in S} p(s'|s,a\,) V^*(s')$$

- $V^*(s'\,) = \max_a Q^*(s',a)$
- With **optimum** policy $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$

Definition (Nash Q-function)

Nash-Q function is defined as

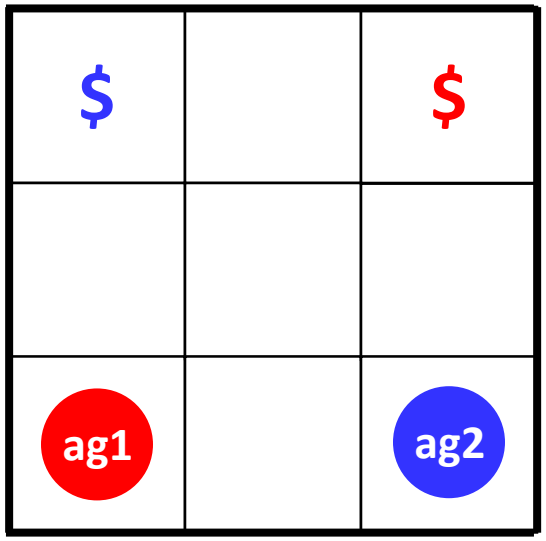
$$Q_i^*(s,\vec{a}) = r_i(s,\vec{a},s') + \gamma \sum_{s' \in S} p(s'|s,\vec{a}) V_i^*(s')$$

- $V_i^*(s') = \text{Nash}\, Q_i^*(s')$ is Nash equilibrium value that can be computed by solving the following state game

$$(Q_1^*(s',\vec{a})\,, \ldots, Q_n^*(s',\vec{a})\,)$$

2. OVERVIEW : Static Game to Dynamic Game

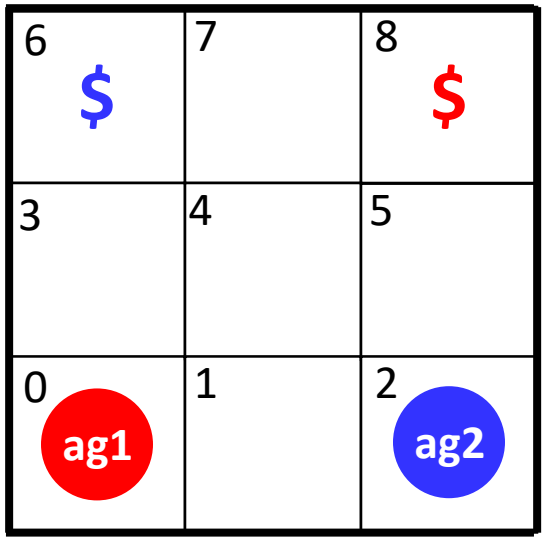
Stochastic Game Example



- Grid game has deterministic moves
- Two agents start from respective lower corners, trying to reach their goal cells in the top row
- Agent can move only one cell a time, and in four possible directions: Left, Right, Up, Down
- If two agents attempt to move into the same cell (excluding a goal cell), they are bounced back to their previous cells
- The game ends as soon as an agent reaches its goal
  - The objective of an agent in this game is therefore to reach its goal with a minimum No. of steps
- Agents do not know
  - the locations of their goals at the beginning of the learning period
  - their own and the other agents' payoff functions
- Agent choose their action simultaneously and observe
  - the previous actions of both agents and the current joint state
  - the immediate rewards after both agents choose their actions

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Example



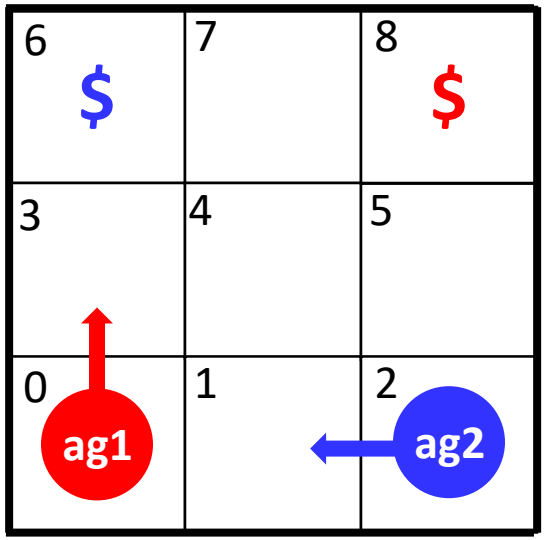
- The action space of agent i, $i = 1,2$, is $A_i = \{Left, Right, Down, Up\}$
- The sate space is $S = \{(0,1), (0,2), \dots, (8,7)\}$
  - $s = (l_1, l_2)$ represents the agents' joint location
  - $l_i \in \{0, 2, \dots, 8\}$ is the indexed location
- The reward function is, for $i = 1, 2$,

$$r_i = \begin{cases} 100 & \text{if } L(l_i, a_i) = Goal_i \\ -1 & \text{if } L(l_1, a_1) = L(l_2, a_2) \text{ and } L(l_i, a_i) \neq Goal_i \text{ for } i = 1,2 \\ 0 & \text{otherwise} \end{cases}$$

$l_i' = L(l_i, a_i)$ is the next location when executing a_i at l_i

2. OVERVIEW : Static Game to Dynamic Game

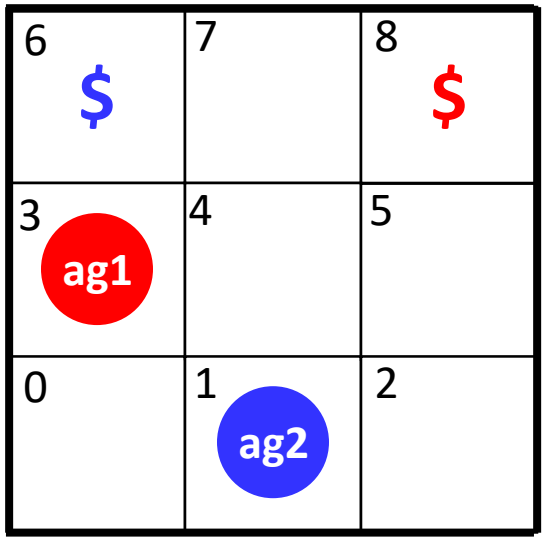
Stochastic Game Example



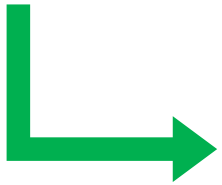
- $s = (l_1, l_2) = (0,2)$
- $a = (a_1, a_2) = (Up, Left)$

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Example

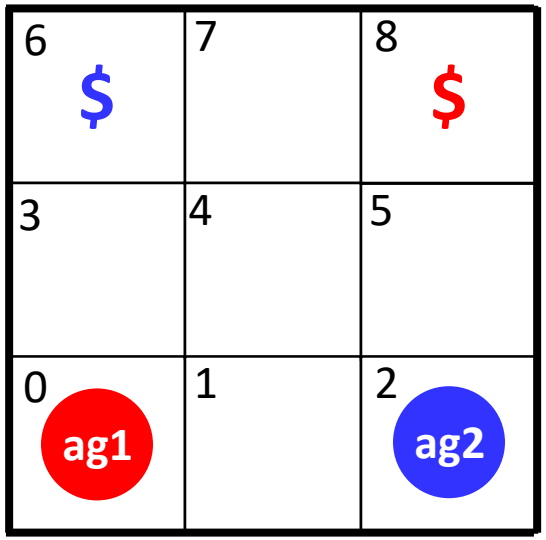


- $s = (l_1, l_2) = (0,2)$
- $a = (a_1, a_2) = (Up, Left)$



- $s' = \big(L(l_1, a_1), L(l_2, a_2)\big) = (3, 1)$
- $r_1 = 0$
- $r_2 = 0$

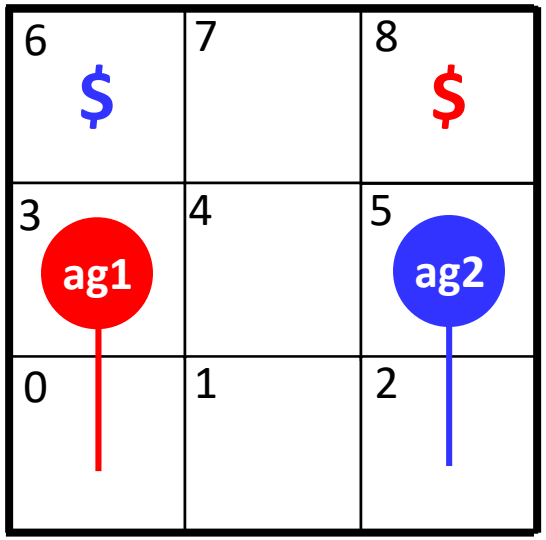
Grid Game 1 represented as stochastic game



Nash Equilibrium strategies

2. OVERVIEW : Static Game to Dynamic Game

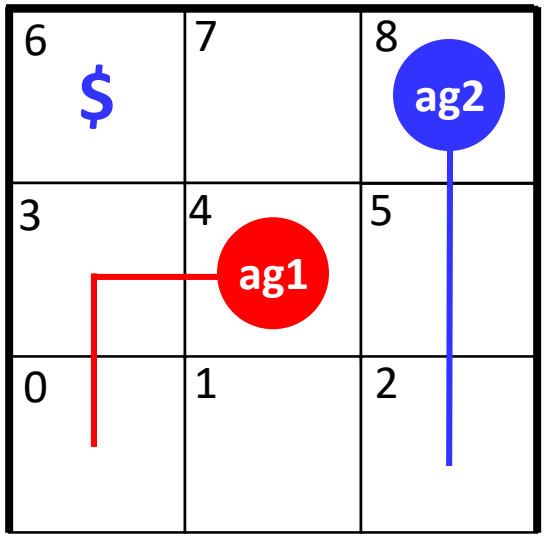
Stochastic Game Example



Nash Equilibrium strategies

2. OVERVIEW : Static Game to Dynamic Game

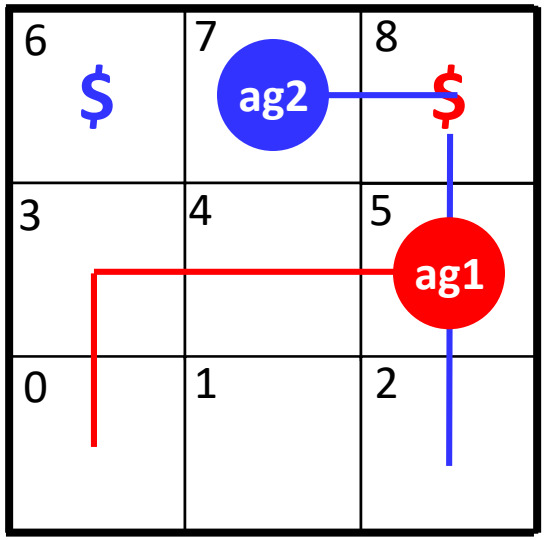
Stochastic Game Example



Nash Equilibrium policies

2. OVERVIEW : Static Game to Dynamic Game

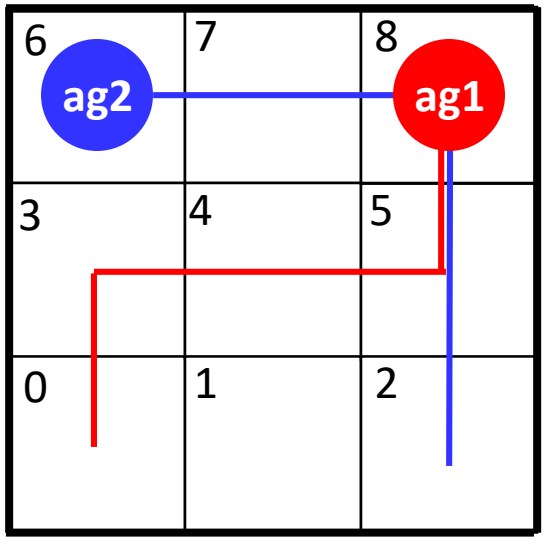
Stochastic Game Example



Nash Equilibrium policies

2. OVERVIEW : Static Game to Dynamic Game

Stochastic Game Example



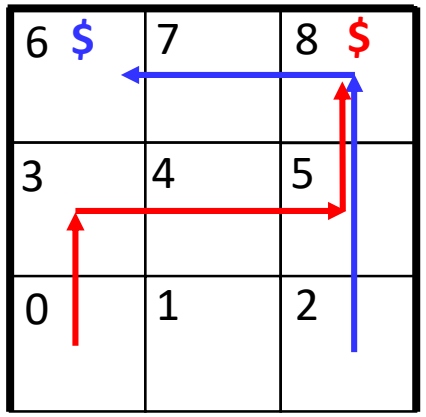
Nash Equilibrium policies

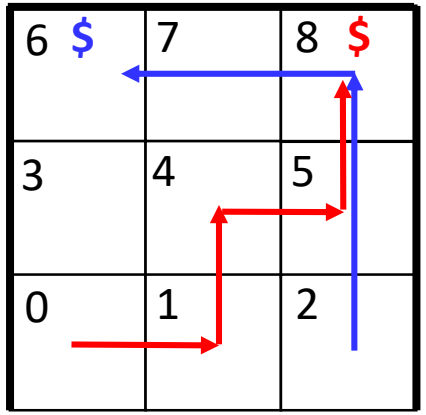
State s	$\pi_1(s)$
$(0, any)$	U
$(3, any)$	$Right$
$(4, any)$	$Right$
$(5, any)$	Up

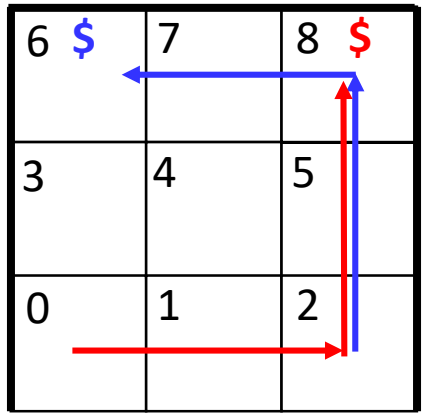
Nash strategy for agent 1

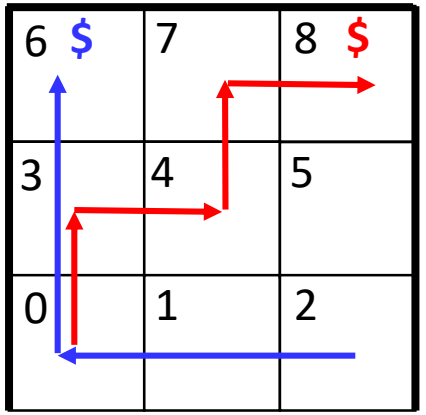
2. OVERVIEW : Static Game to Dynamic Game

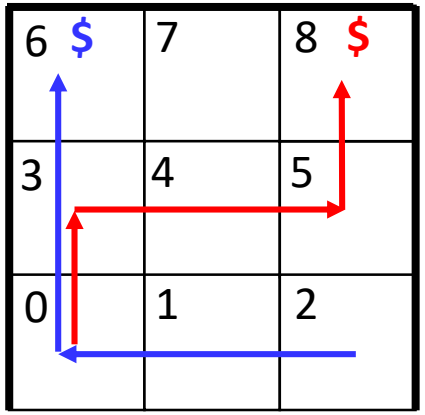
Stochastic Game Example



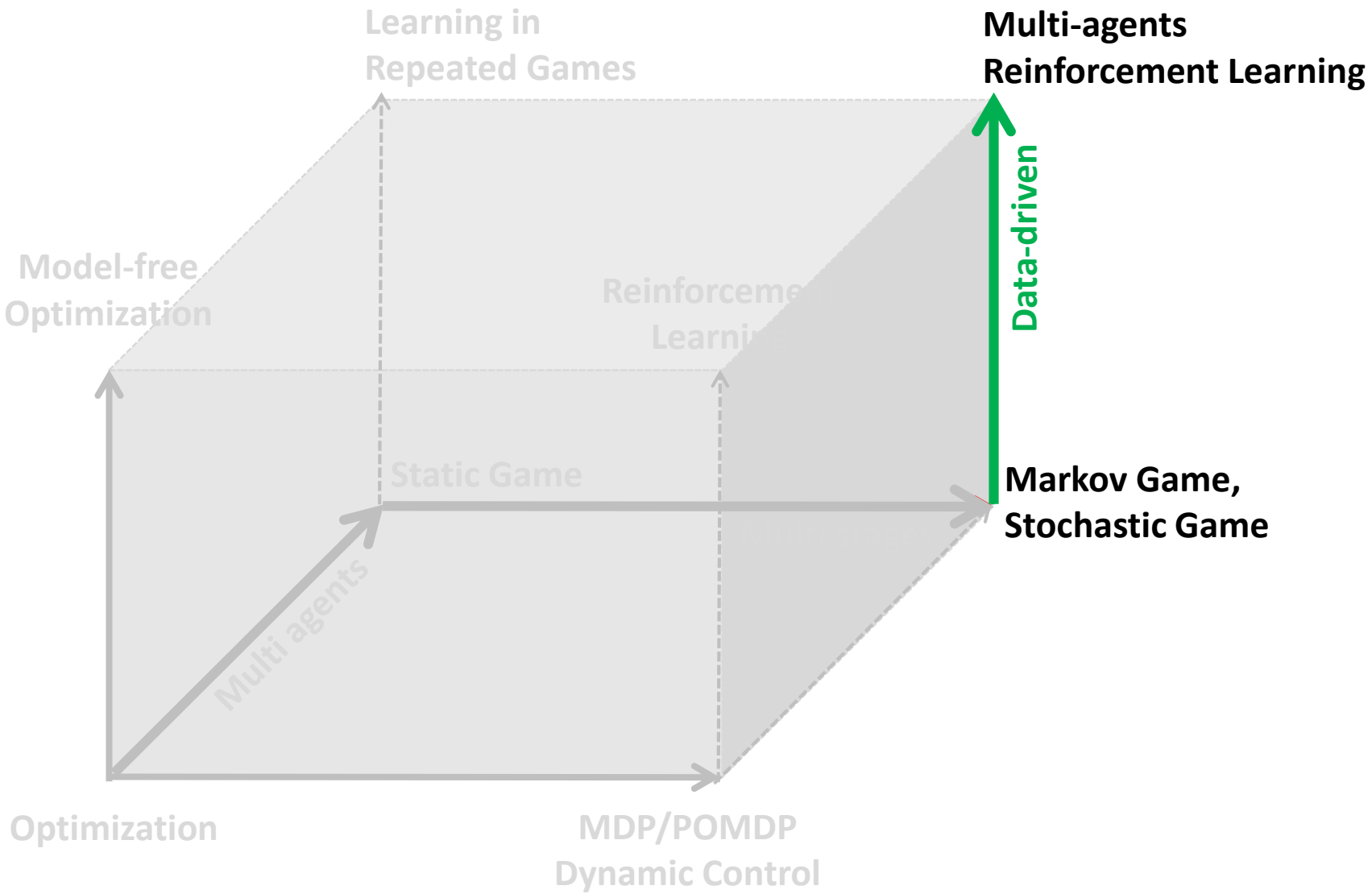








2. OVERVIEW : Dynamic Game to Multi-Agent Reinforcement Learning



2. OVERVIEW : Dynamic Game to Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Q Learning Template

MultiQ(StochastiGame, f, γ, α, T)

Inputs	equilibrium selection function f
	discounting factor γ
	learning rate α
	total training time T
Outputs	state − value functions V_i^*
	action − value functions Q_i^*
Initialize	$s, a_1, \dots, a_n$ and $Q_1, \dots, Q_n$

for $t = 1{:}T$

1. simulate actions $\vec{a} = (a_1, \dots, a_n)$in state s
2. observe rewards $r_1, \dots, r_n$ and next state s'
3. for $i = 1$ to n (for each agent)
   (a) $V_i(s') = f_i(Q_1(s', \vec{a}), \dots, Q_n(s', \vec{a}))$
   (b) $Q_i(s, \vec{a}) = (1 - \alpha_i)Q_i(s, \vec{a}) + \alpha_i[r_i + \gamma V_i(s')]$
4. agent choose actions $a_1', \dots, a_n'$
5. $s = s', a_1 = a_1', \dots, a_n = a_n'$
6. adjust learning rate $\alpha = (\alpha_1, \dots, \alpha_n)$

2. OVERVIEW : Dynamic Game to Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning Overview

Equilibrium selection function f : $V_i(s') = f_i(Q_1(s', \vec{a}), \ldots, Q_n(s', \vec{a}))$

- We going to study the following equilibrium concept:
  - Value function based (Bellman function based)
    - Single agent Q-learning
    - Independent Q learning by multiple agents
    - Nash-Q learning (Hu and Wellman 1998)
    - Minmax-Q learning (Littman 1994)
    - Friend-or-Foe Q learning (Littman 2001)
    - Correlated Q learning (Greenwald and Hall 2003)
  - Policy gradient methods (direct search for policy)
    - Wind-or-Learn-Fast Policy Hill Climbing (WOLF-PHC) (Policy gradient method)
  - Deep learning based MARL (limited to a team game or cooperative game)
    - Learning to cooperate
    - Learning to communicate

2. OVERVIEW : Dynamic Game to Multi-Agent Reinforcement Learning

Nash Q-Learning

- Q-learning directly find optimal Q-function (Q table) instead of optimum finding policy π^*

- Single agent Q-learning:
  - Iteratively find optimal Q values $Q^*(s,a)$ (table)

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r(s,a) + \gamma V(s')]$$
$$= (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \max_a Q(s',a)\right]$$

- Nah Q-learning:
  - Iteratively find Nash-Q values Nash $Q_i^*(s,a_1,\dots,a_n)$ (table for each agent)

$$Q_i(s,\vec{a}) = (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma V_i(s')]$$
$$= (1-\alpha)Q_i(s,\vec{a}) + \alpha[r_i + \gamma \,\text{Nash}\, Q_i(s')]$$

$Q_i(s',\vec{a})$: Nash-Q values (state-action values)
Nash $Q_i(s')$: **Nash equilibrium value** of Nash-Q values

- the learning agent updates its Nash Q-value depending on the joint strategy of all the players and not only its own expected payoff.

2. OVERVIEW : Dynamic Game to Multi-Agent Reinforcement Learning

MinMax Q-Learning

For agent $i = 1{:}2$

$$\text{(a) } V_i(s') = f_i(Q_1(s', a_i, a_{-i}), Q_2(s', a_i, a_{-i}),) = \max_{\pi_i(s',\cdot)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i(s', a_i, a_{-i}) \pi_i(s', a_i)$$

$$= \text{Maxmin}\, Q_i(s')$$

$$\text{(b) } Q_i(s, a_i, a_{-i}) = (1-\alpha) Q_i(s, a_i, a_{-i}) + \alpha[r_i + \gamma V_i(s')]$$

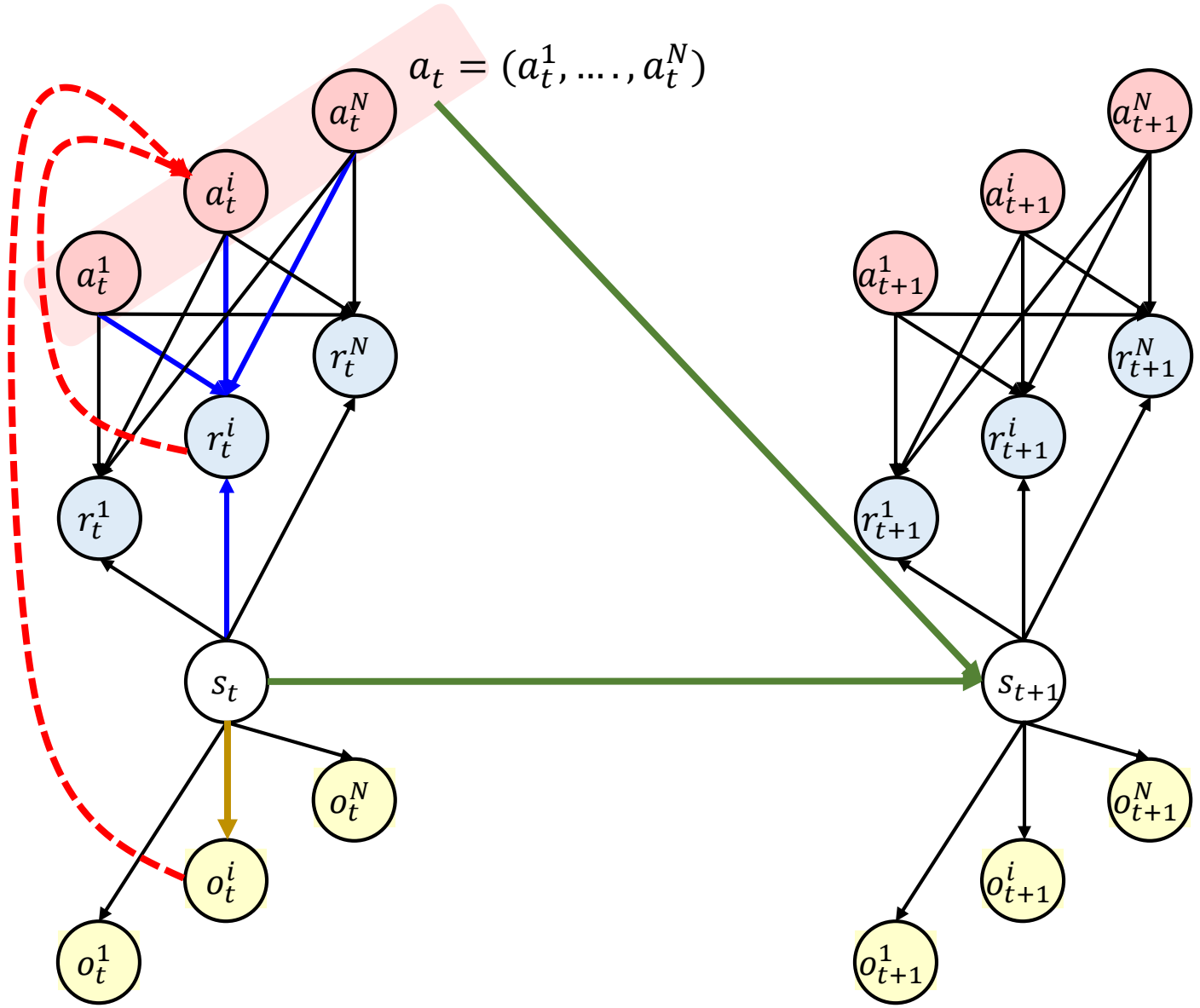
$$= (1-\alpha) Q_i(s, a_i, a_{-i}) + \alpha[r_i + \gamma\, \text{Maxmin}\, Q_i(s')]$$

Action selection strategy:

$$\pi_i(s',\cdot) = \underset{\pi_i(s',\cdot)}{\text{argmax}} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i(s', a_i, a_{-i}) \pi_i(s, a_i)$$

- Note that when computing the maxmin value, each agent can consider only its own action value function $Q_i(s', a_i, a_{-i})$
- But, still each agent need to track the action taken by the other agent for updating

3. Deep MARL: Stochastic Game



- ***Transition:***

$$P(s_{t+1}|a_t^1, \ldots, a_t^N, s_t)$$

- ***Reward:***

$$r_t^i(s_t, a_t^1, \ldots, a_t^N)$$

- ***Observation:***

$$o_t^i = h^i(s_t)$$

- ***Decentralized Policy:***

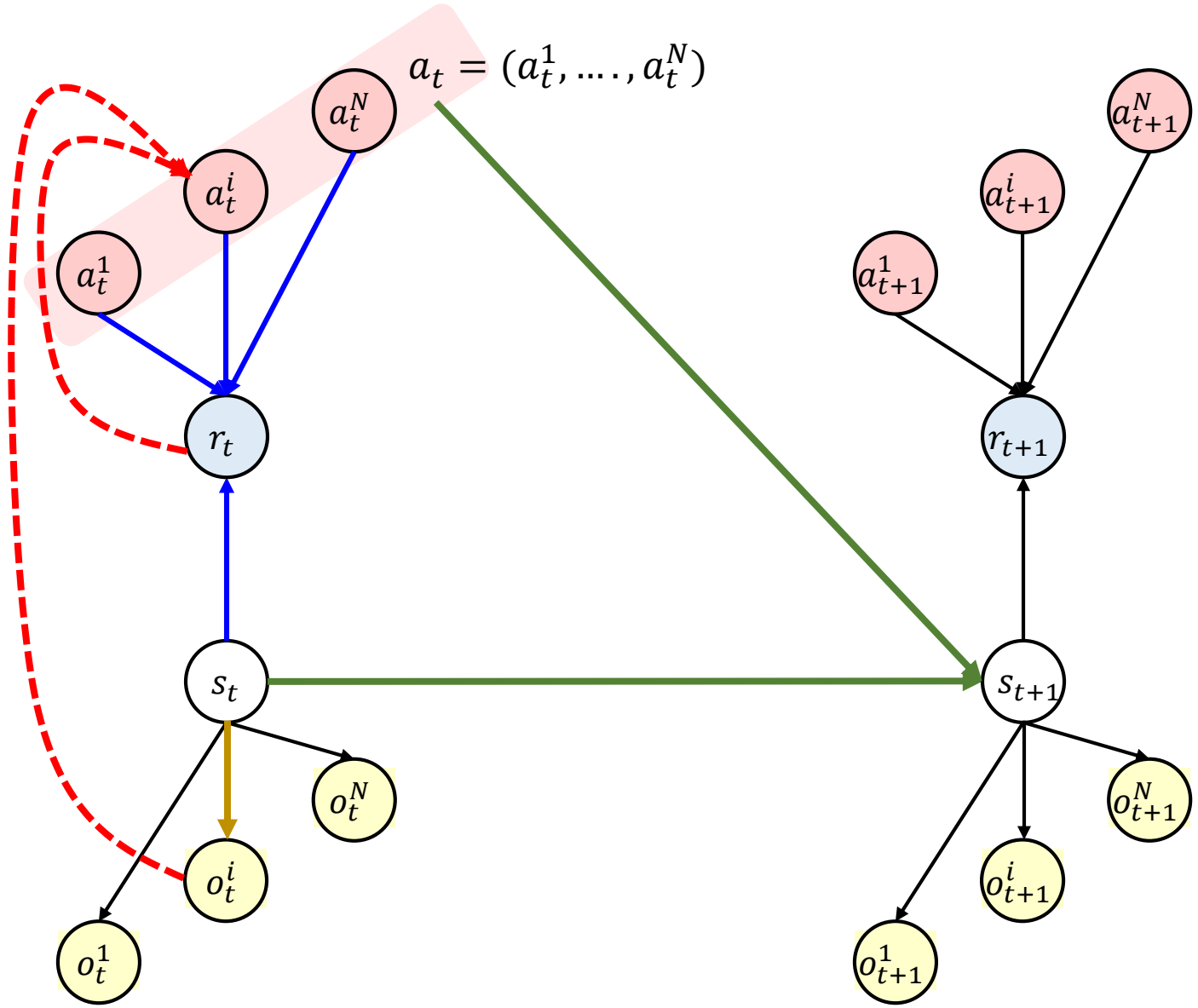
$$\pi(s_t, a_t^1, \ldots, a_t^N) \approx \prod_{i=1}^{N} \pi_i(s_t, a_t^i)$$

$$\approx \prod_{i=1}^{N} \pi_i(o_t^i, a_t^i)$$

- ***Objective of agent i:***

$$\max_{\pi_i} E\left[\sum_{t=1}^{T} r_t^i(s_t, a_t^1, \ldots, a_t^N)\right]$$

3. Deep MARL: Team Game



- ***Transition:***

$$P(s_{t+1}|a_t^1, \ldots, a_t^N, s_t)$$

- ***Reward:***

$$r_t(s_t, a_t^1, \ldots, a_t^N)$$

- ***Observation:***

$$o_t^i = h^i(s_t)$$

- ***Decentralized Policy:***

$$\pi(s_t, a_t^1, \ldots, a_t^N) \approx \prod_{i=1}^{N} \pi_i(s_t, a_t^i)$$

$$\approx \prod_{i=1}^{N} \pi_i(o_t^i, a_t^i)$$

- ***Objective of agent i:***

$$\max_{\pi_i} E\left[\sum_{t=1}^{T} r_t(s_t, a_t^1, \ldots, a_t^N)\right]$$

3. Deep MARL: MARL approach to Team Game

		Training	
		Centralized Training	Decentralized Training
Execution	Centralized Execution	MDP	
	Decentralized Execution	Dec-(PO)MDP	?

3. Deep MARL: MARL approach to Team Game

		Training	
		Centralized Training	**Decentralized Training**
Execution	Centralized Execution	MDP	
	Decentralized Execution	Dec-(PO)MDP	?

- **Centralized Training** vs **Decentralized Training**
  - In centralized training, we assume a central learner who can access all the global information $s = (s_1, \dots, s_N)$ and $a = (a_1, \dots, a_N)$ for modeling mutual interactions among agents
  - In decentralized training, each agent has a limited information about the global state and the joint action

3. Deep MARL: MARL approach to Team Game

		Training	
		Centralized Training	Decentralized Training
Execution	**Centralized Execution**	MDP	
	Decentralized Execution	Dec-(PO)MDP	?

- Centralized Training vs Decentralized Training
  - In centralized training, we assume a central learner who can access all the global information $s = (s_1, \dots, s_N)$ and $a = (a_1, \dots, a_N)$ for modeling mutual interactions among agents
  - In decentralized training, each agent has a limited information about the global state and the joint action

- **Centralized Exestuation** vs **Decentralized Execution**
  - In centralized exestuation, a joint action $a = (a_1, \dots, a_N)$ is selected by a central agent
  - In centralized exestuation, each agent selects individual action to compose a joint action:

$$\pi(s, a) \approx \prod_{i=1}^{N} \pi_i(\text{Information of agent } i, a_i)$$

3. Deep MARL: MARL approach to Team Game

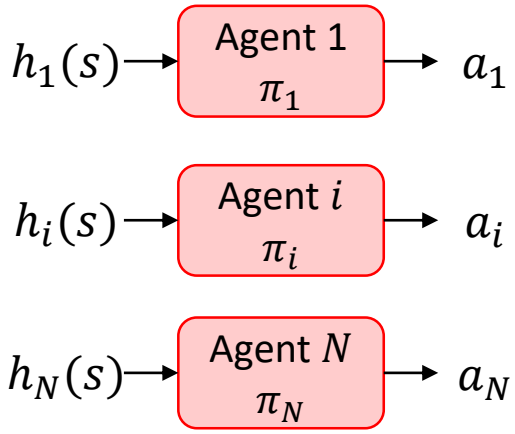
		Training	
		Centralized Training	Decentralized Training
Execution	Centralized Execution	MDP	
	Decentralized Execution	Dec-(PO)MDP (CTDE)	?

$s = (s_1, \dots, s_N)$
$a = (a_1, \dots, a_N)$
$r = f(s, a)$

$Q(s, a)$

$\pi_1, \dots, \pi_i, \dots, \pi_N$

Centralized Training



Decentralized Execution

- **Centralized Training** and **Decentralized Execution** (CTDE) is a widely adopted to overcome the non-stationarity problem when training multi-agent systems (Oliehoek et al., 2008; Kraemer & Banerjee, 2016; Jorge et al., 2016; Foerster et al., 2018)
- CTDE enables to leverage the observations of each agent, as well as their actions, to better model the interactions among agents during training.
- Depending on the information structure $h_i(s)$ of each agent has in execution, there are various approaches in CTDE
  - Local observations: each agent can access only to its local (state) observation
  - Global observations: each agent can access to the global state (observation)

3. Deep MARL: CTDE framework for Team Game

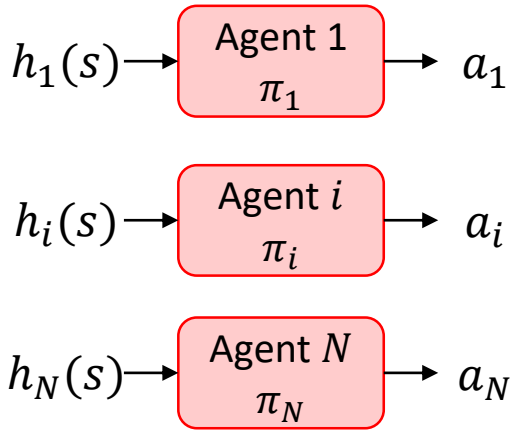
		Training
		Centralized Training
Execution	Centralized Execution	MDP
	Decentralized Execution	Dec-(PO)MDP (CTDE)

$s = (s_1, \dots, s_N)$
$a = (a_1, \dots, a_N)$
$r = f(s, a)$

$Q(s, a)$

$\pi_1, \dots, \pi_i, \dots, \pi_N$

Centralized Training



Decentralized Execution

- **Depending on the information that each agent can use when making a decision (information structure)**, CTDE has different approaches.
  - Local observation: $h_i(s) = s_i(\text{or } o_i)$: (when there is partial observability and/or communication constraints)
  - Global observation: $h_i(s) = s(\text{or } o)$: (global observation or communication is allowed)

3. Deep MARL: Information Structure in CTDE framework

Local observation: $h_i(s) = s_i(\text{or } o_i)$

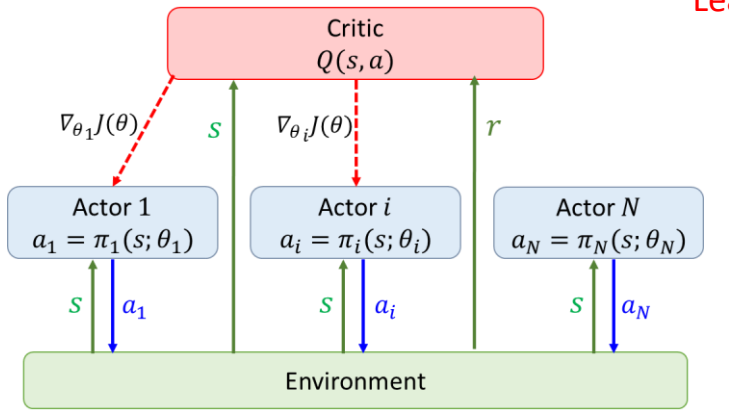
$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s_i, a_i)$$

Do not know other agent's state and action

$$\approx \prod_{i=1}^{N} \pi_i(s_i, a_i; \theta_i)$$:Function approximation

Consensus through central $Q(s,a;\phi)$ modeling

Learning to cooperate



Global observation: $h_i(s) = s(\text{or } o)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s, a_i)$$

Do not know other agent's action

$$\approx \prod_{i=1}^{N} \pi_i(s, a_i; \theta_i)$$:Function approx.:

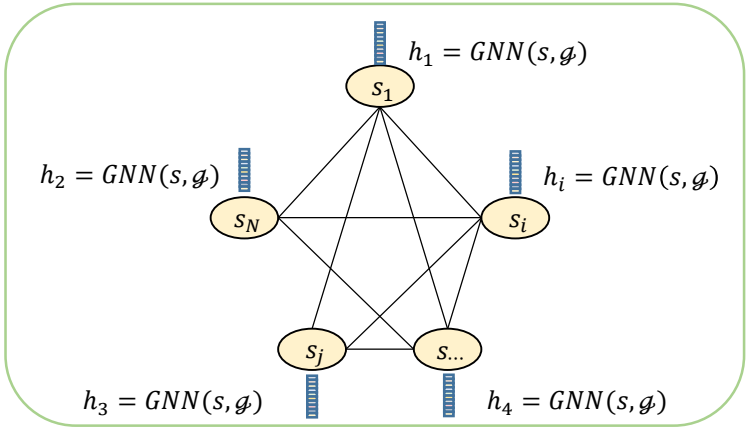
→**difficult** to process s (high dim)

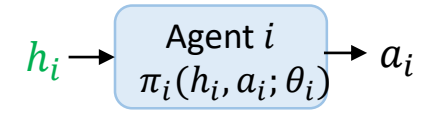
$$\approx \prod_{i=1}^{N} \pi_i(h_i = GNN(s;\phi_i), a_i; \theta_i)$$

(State representation with inductive biases)

Consensus through Communication & GNN

Learning to communicate





3. Deep MARL: Information Structure in CTDE framework

Local observation: $h_i(s) = s_i (\text{or } o_i)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s_i, a_i)$$

Do not know other agent's state and action

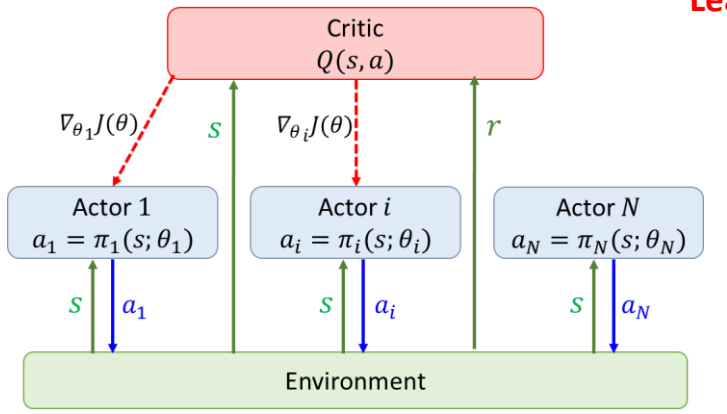
$$\approx \prod_{i=1}^{N} \pi_i(s_i, a_i; \theta_i)$$:Function approximation

$$= \prod_{i=1}^{N} \pi(s_i, a_i; \theta)$$

Parameter sharing among **homogeneous** agents

Consensus through central $Q(s,a;\phi)$ modeling

Learning to cooperate



Global observation: $h_i(s) = s (\text{or } o)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s, a_i)$$

Do not know other agent's action

$$\approx \prod_{i=1}^{N} \pi_i(s, a_i; \theta_i)$$:Function approx.:

→**difficult** to process s (high dim)

$$\approx \prod_{i=1}^{N} \pi_i(h_i = GNN(s;\phi_i), a_i; \theta_i)$$

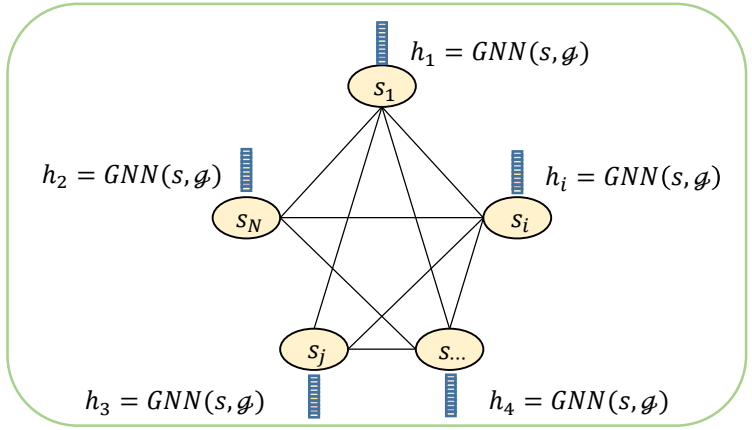
(State representation with inductive biases)

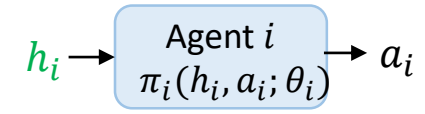
$$\approx \prod_{i=1}^{N} \pi(h_i = GNN(s;\phi), a_i; \theta)$$

Parameter sharing among **homogeneous** agents

Consensus through Communication & GNN

Learning to communicate





3. Deep MARL: Information Structure in CTDE framework

Local observation: $h_i(s) = s_i (\text{or } o_i)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s_i, a_i)$$

Do not know other agent's state and action

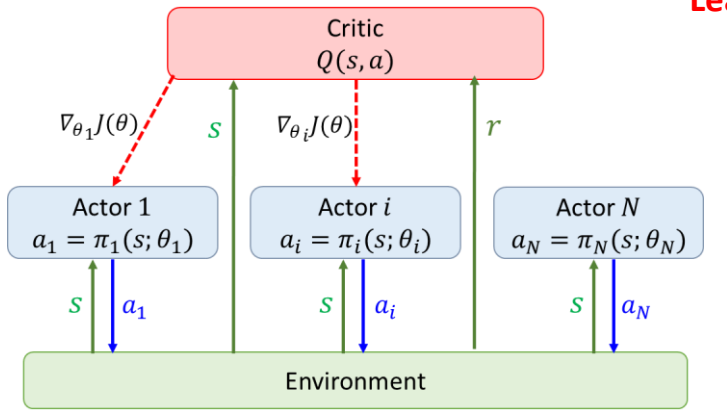
$$\approx \prod_{i=1}^{N} \pi_i(s_i, a_i; \theta_i)$$:Function approximation

$$= \prod_{i=1}^{N} \pi(s_i, a_i; \theta_{g(i)}) \quad g(i) \in \{1, \dots, M\}$$

Parameter sharing among **a group of agents**

Consensus through central $Q(s,a;\phi)$ modeling

Learning to cooperate



Global observation: $h_i(s) = s (\text{or } o)$

$$\pi(s,a) \approx \prod_{i=1}^{N} \pi_i(s, a_i)$$

Do not know other agent's action

$$\approx \prod_{i=1}^{N} \pi_i(s, a_i; \theta_i)$$:Function approx.:

→**difficult** to process s (high dim)

$$\approx \prod_{i=1}^{N} \pi_i(h_i = GNN(s;\phi_i), a_i; \theta_i)$$

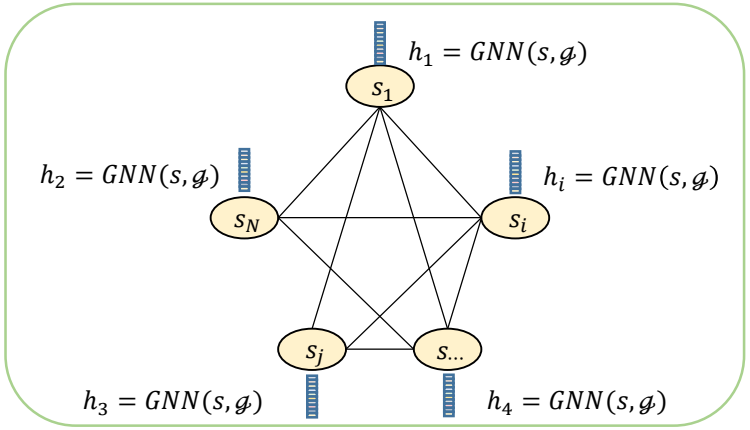
(State representation with inductive biases)

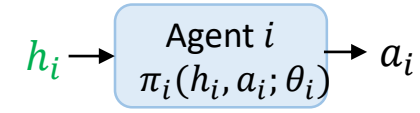
$$\approx \prod_{i=1}^{N} \pi(h_i = GNN(s;\phi), a_i; \theta_{g(i)}) \quad g(i) \in \{1, \dots, M\}$$

Parameter sharing among **a group of agents**

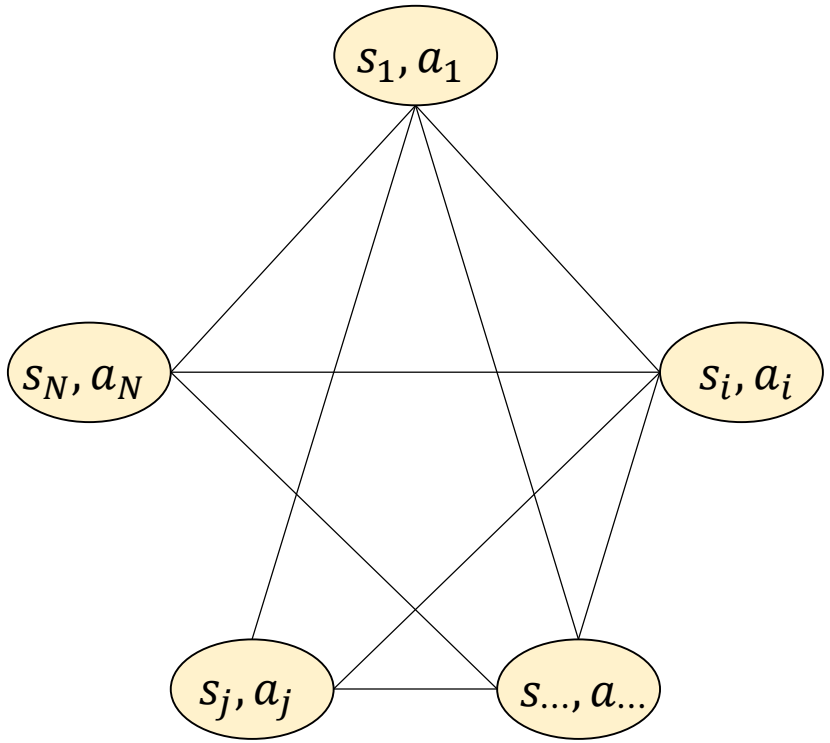
Consensus through Communication & GNN

Learning to communicate





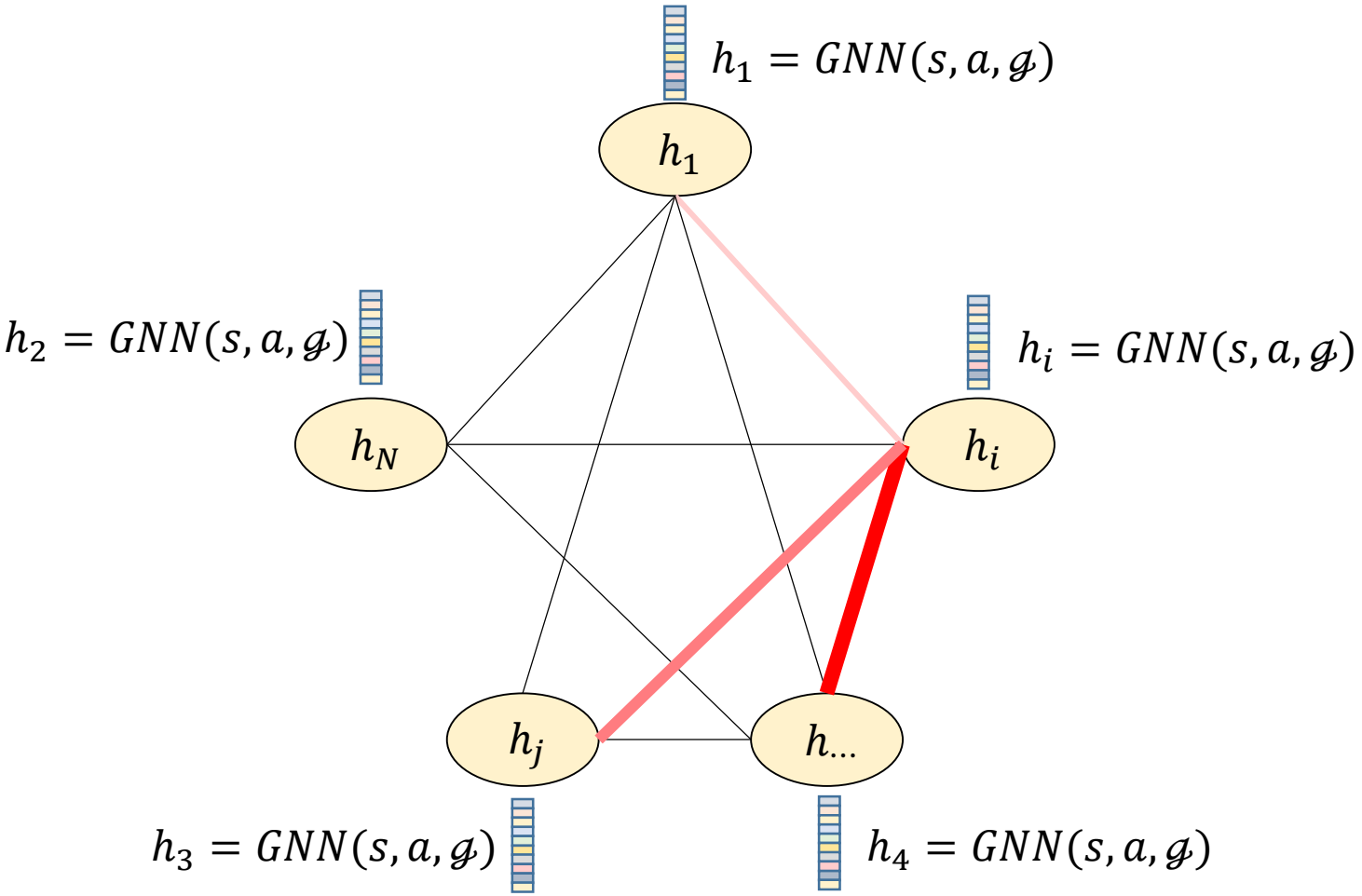
3. Deep MARL: Central Critic approach + Communication Approach



Objective:

Find the decentralized policy $\pi_\theta(s,a) \approx \prod_{i=1}^{N} \pi_{\theta_i}(s_i, a_i)$

3. Deep MARL: Central Critic approach + Communication Approach



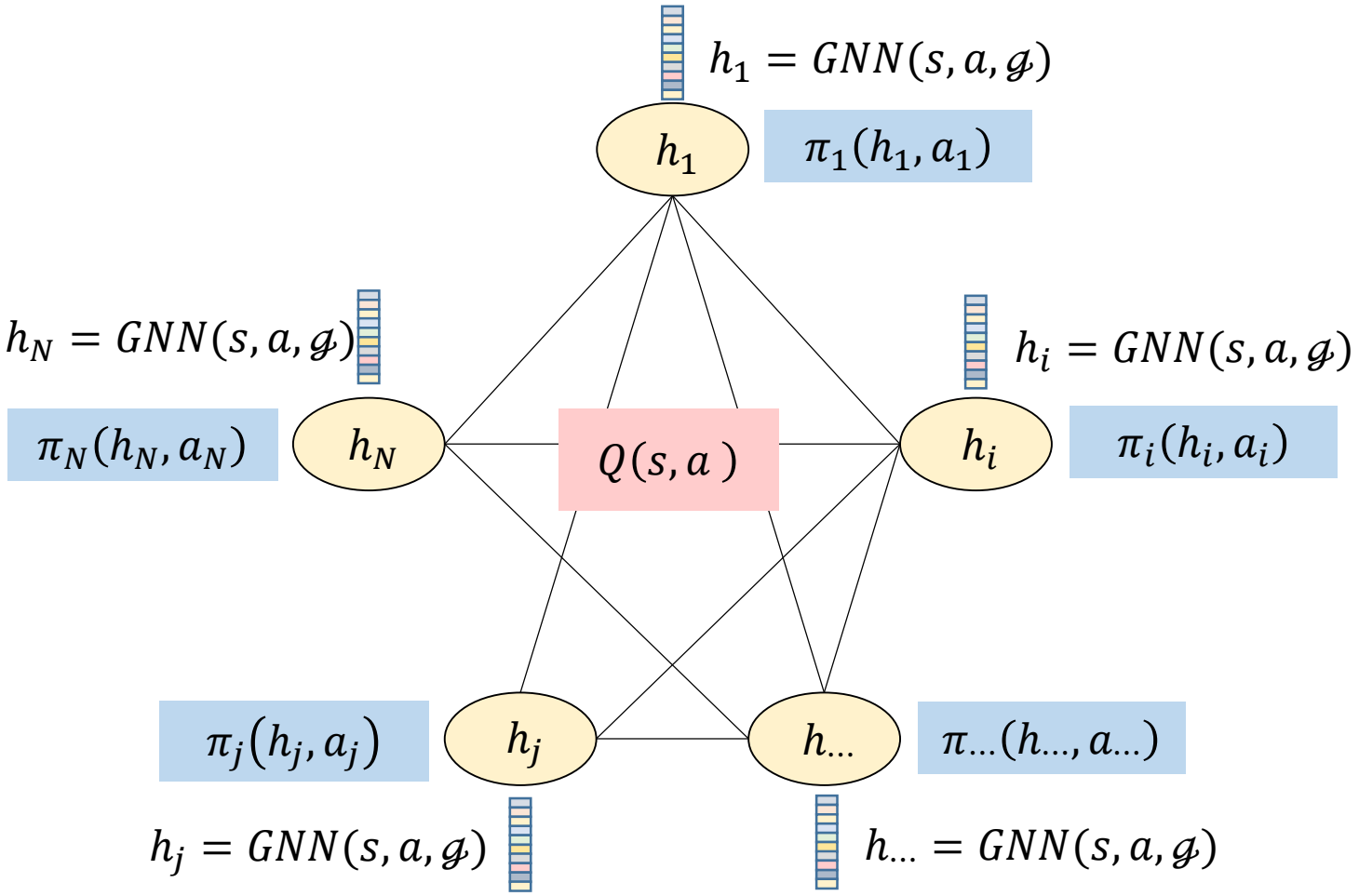
Message computation

- $\bar{h}_i = \sum_{j \in \mathcal{N}(i)} m(h_i, h_j)$ *or*
- $\bar{h}_i = \sum_{j \in \mathcal{N}(i)} a_{ij} m(h_j)$ *or*
- $h_i = \sum_{j \in \mathcal{N}(i)} m(h_j)$

Node update

- $h_i' = g(h_i, \bar{h}_i)$

3. Deep MARL: Central Critic approach + Communication Approach



GNN for approximating global Q

Centralized Critic:

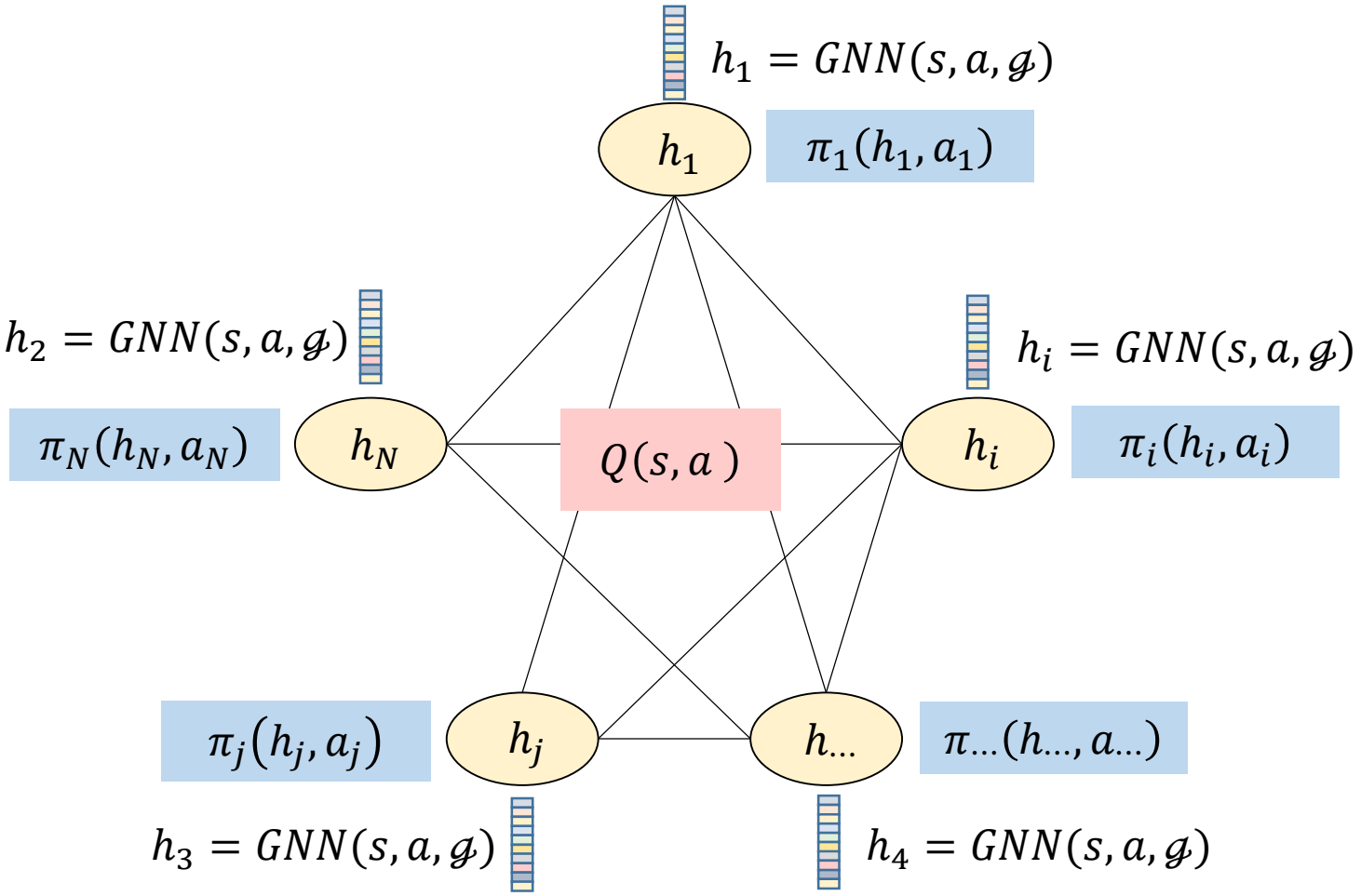
$$Q(s,a) \approx NLP\left(\sum_{i=1}^{N} h_i^{(T)}; \phi\right)$$

$\nabla_{\theta_1} J(\pi)$ $\quad$ $\nabla_{\theta_N} J(\pi)$

$\pi_1(s_i, a_i) \approx NLP(s_i; \theta_1)$ $\quad$ $\pi_N(s_N, a_N) \approx NLP(s_N; \theta_2)$

Decentralized actor

3. Deep MARL: Central Critic approach + Communication Approach



GNN for approximating global Q

Centralized Critic:

$$Q(s, a) \approx NLP\left(\sum_{i=1}^{N} h_i^{(T)}; \phi\right)$$

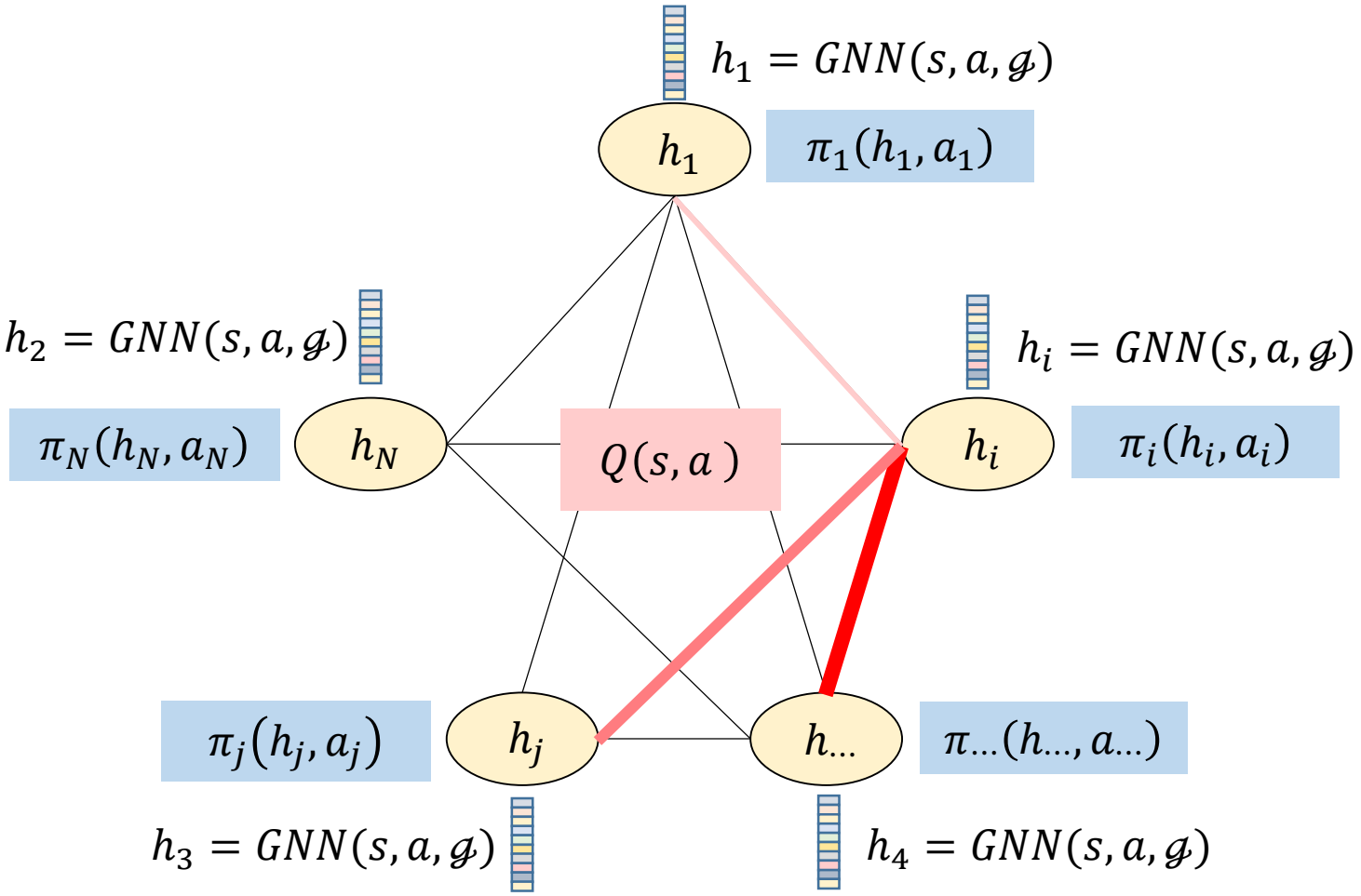
$\nabla_{\theta_1} J(\pi)$ $\qquad$ $\nabla_{\theta_N} J(\pi)$

$\pi_1(h_i, a_i) \approx NLP(h_i; \theta_1)$ $\qquad$ $\pi_N(h_N, a_N) \approx NLP(h_N; \theta_2)$

Decentralized actor

GNN for approximating Individual Policy

3. Deep MARL: Central Critic approach + Communication Approach



GNN for approximating global Q

Centralized Critic:

$$Q(s, a) \approx NLP\left(\sum_{i=1}^{N} h_i^{(T)}; \phi\right)$$

$\nabla_{\theta_1} J(\pi)$ $\quad$ $\nabla_{\theta_N} J(\pi)$

$\pi_1(h_i, a_i) \approx NLP(h_i; \theta_1)$ $\quad$ $\pi_N(h_N, a_N) \approx NLP(h_N; \theta_2)$

Decentralized actor

GNN for approximating Individual Policy

3. Deep MARL: Centralized Training Decentralized Execution

Summary

- Widely adopted to overcome the non-stationarity problem when training multi-agent systems
- CTDE enables to leverage the observations of each agent, as well as their actions, to better estimate the action-value functions during training.
- As the policy of each agent only depends on its own private observations during training, the agents are able to decide which actions to take in a decentralized manner during execution

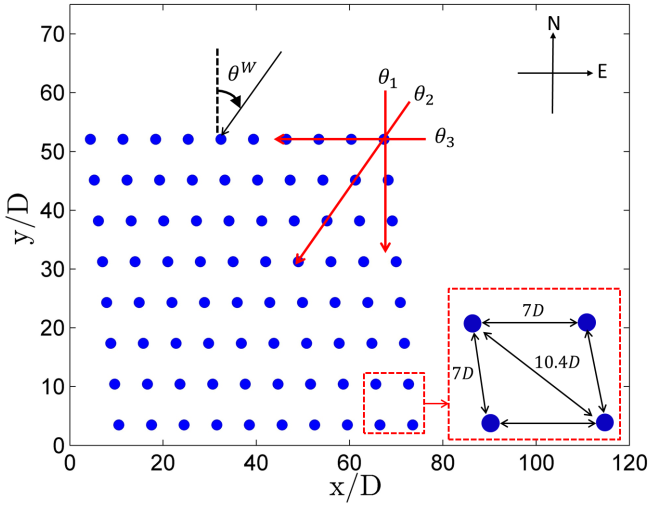
4. APPLICATIONS

Cooperative Wind Farm Control

https://wattsupwiththat.com/2015/08/31/study-wind-farms-offer-diminishing-returns-as-they-grow-more-widespread/

4. Applications : Cooperative Wind Farm Control

Motivation



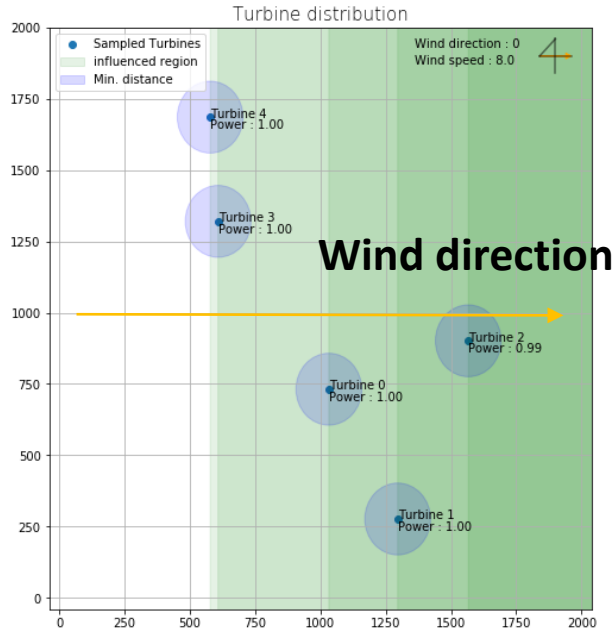
What is the optimum control inputs at different wind conditions (θ, U) ?

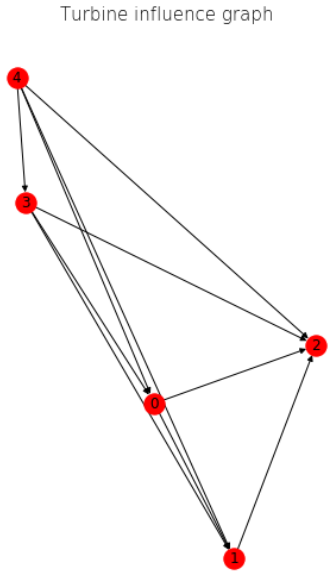
$$\boldsymbol{x}^* = \underset{x}{\operatorname{argmax}}\, f(\boldsymbol{x}; U, \theta) \triangleq \sum_{i=1}^{N} P_i(\boldsymbol{x}; \theta, U)$$

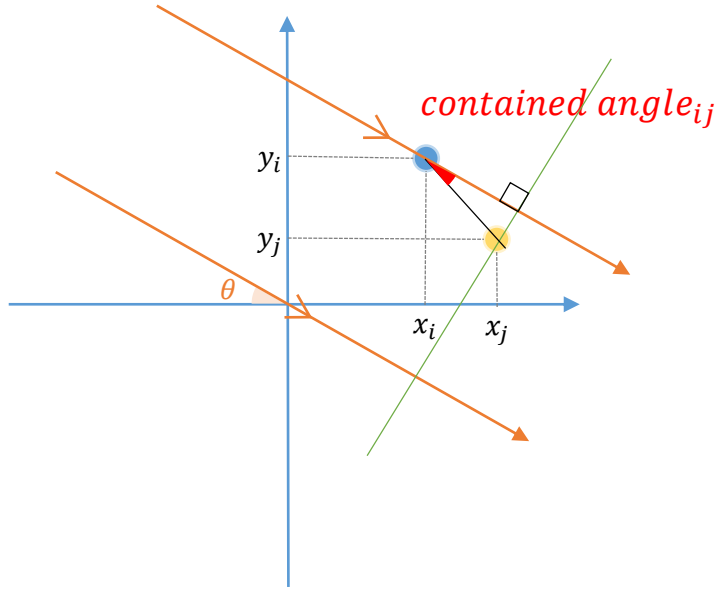
$$\boldsymbol{x}^* = \pi(U, \theta)$$ **:Construct the optimum control policy**

4. Applications : Cooperative Wind Farm Control

GNN Based MARL







Edge features

$$\mathcal{G}_1 = (N_1, E_1, g_1)$$

Node features

Global features

Node features N = $\{free\ flow\ wind\ speed\}_i$

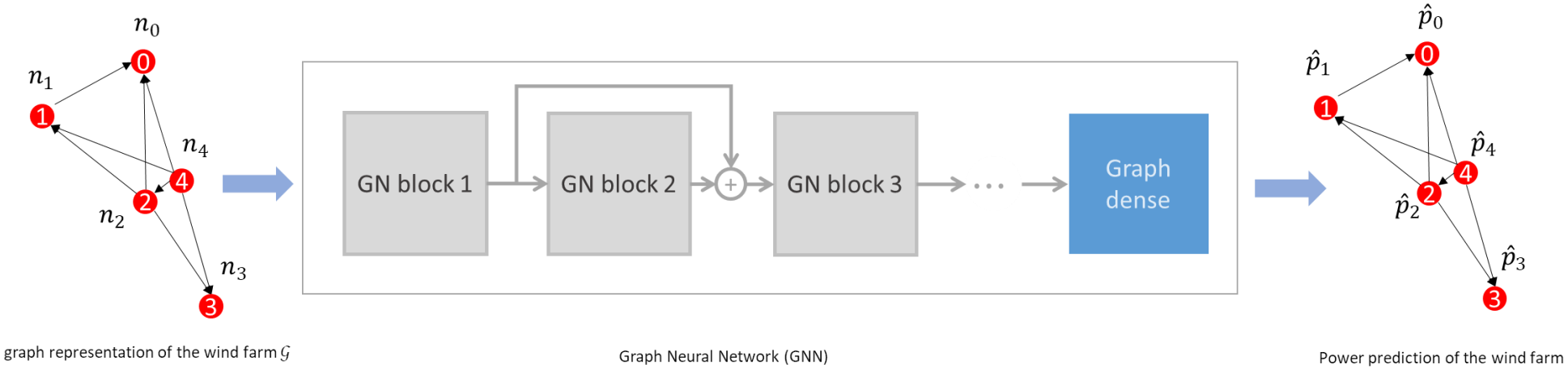
Edge features E = $\{Euclidean\ distance\ between\ turbine_{ij}, contained\ angle_{ij}\}_{i,j}$

Global features g = $\{free\ flow\ wind\ speed\}$ $\quad i, j\ are\ turbine\ index$

4. Applications : Cooperative Wind Farm Control

GNN Based MARL

$$\hat{\mathcal{G}}_y = GraphNeuralNetwork\ (\mathcal{G}_x; \theta)$$



graph representation of the wind farm $\mathcal{G}$

Graph Neural Network (GNN)

Power prediction of the wind farm

Graph Neural Network (GNN) is a special type of neural network that takes graphs as inputs.
One Important property of GNN is that GNN learn message propagation protocols among edges, nodes, global attributes.

In this research, we used Graph Network (GN) blocks and Graph Dense layer for learning message propagation protocols and predicting wind farm power generations.

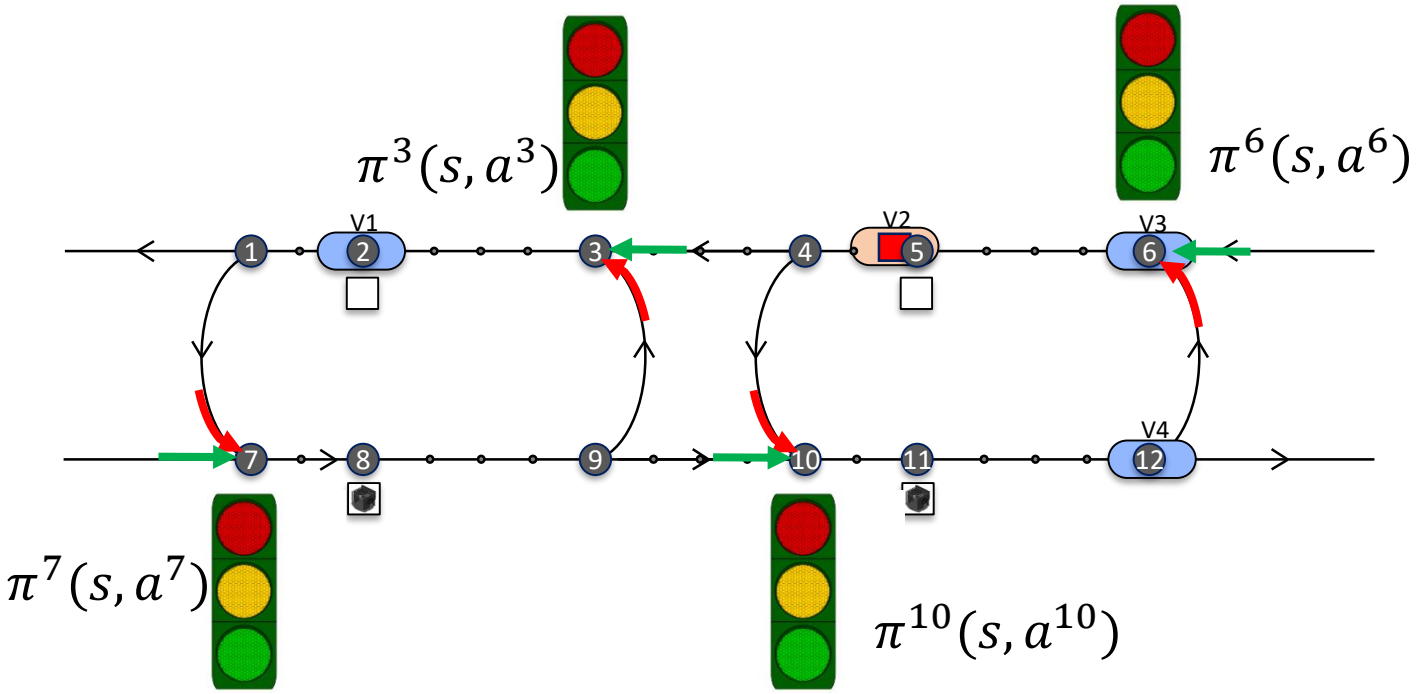
4. APPLICATIONS

Overhead Hoist Transport (OHT) Control in Semiconductor Fab.

https://wccftech.com/globalfoundries-puts-7nm-on-hold-indefinitely-spins-off-asic-division-as-subsidiary/

4. Applications : Congestion Management of OHTs in Semiconductor Fab

Motivation

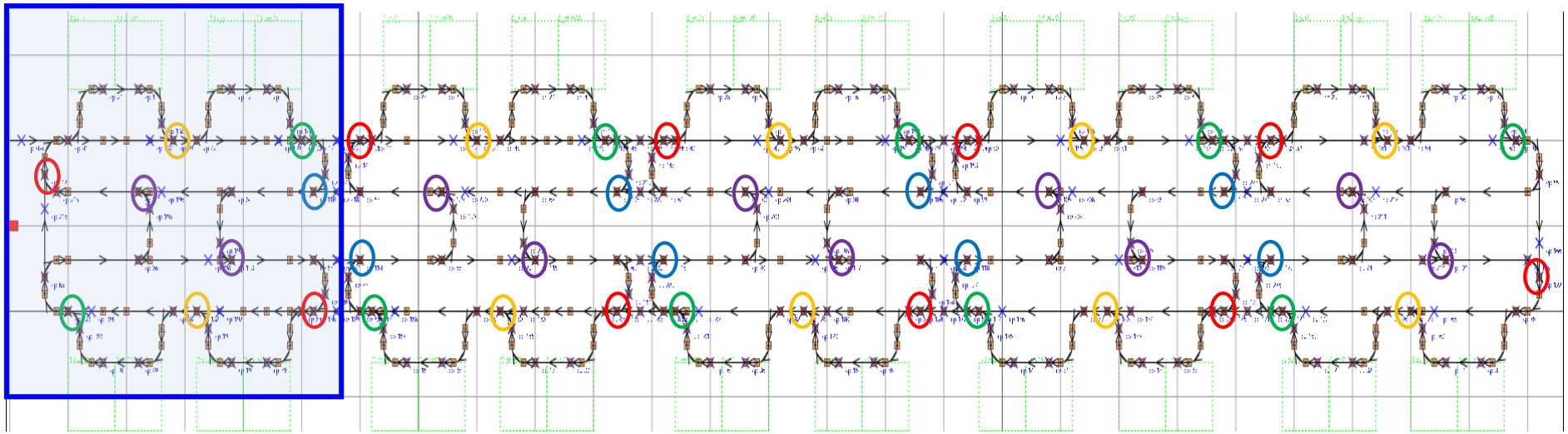


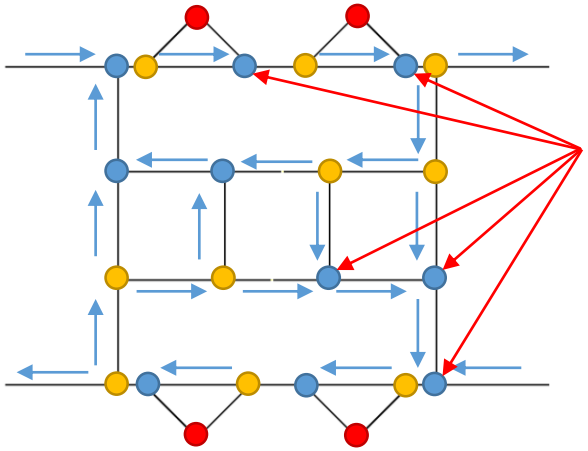
- 전체 시스템 성능을 향상시키는 joint policy $\pi(s, a)$를 개별적인 local policy $\pi(s, a^i)$ 로 표현하도록 하는 Decentralized Actor-critic algorithm을 적용한다.

$$\pi(s, a) = \prod_{i=1}^{N} \pi(s, a^i)$$

4. Applications : Congestion Management of OHTs in Semiconductor Fab

Graph Construction



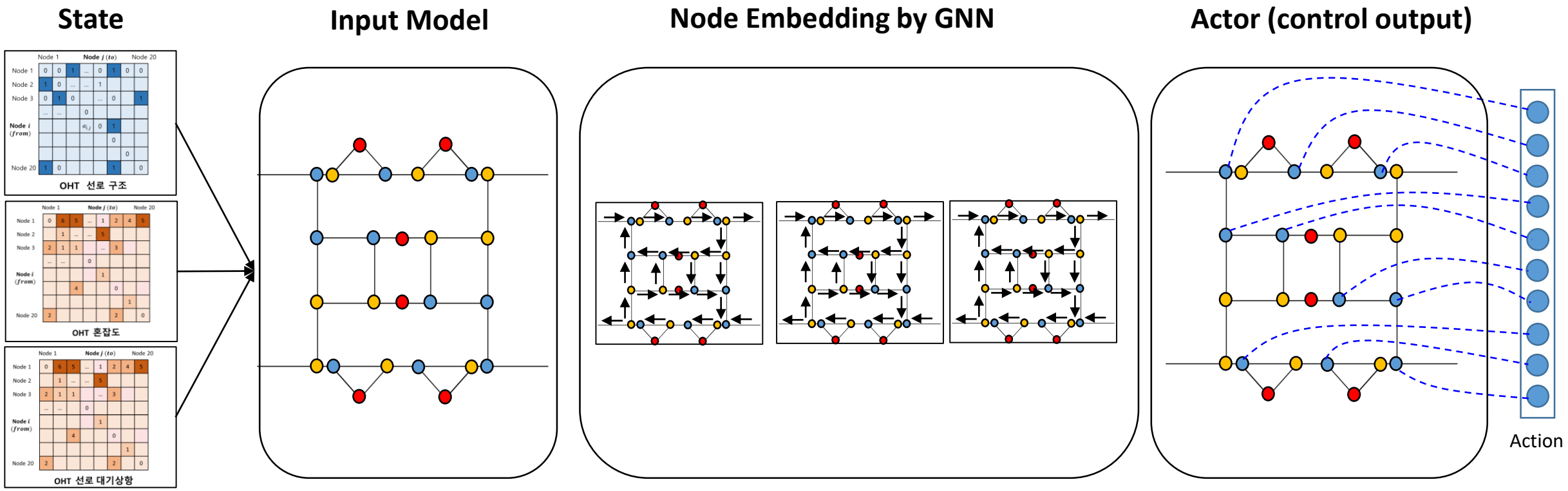


control nodes
(Intersection)

- Node
  - Tool node
  - Intersection node => **Control**
  - Distributed node
- Directed edge : path

4. Applications : Congestion Management of OHTs in Semiconductor Fab

State Definition & Graph Embedding



4. Applications : Congestion Management of OHTs in Semiconductor Fab

State Definition & Graph Embedding

Decentralized MDP Formulation (Team Game)

- State
  - ✓ Graph Structure : $G = (V, E)$
  - ✓ State vector of nodes : x_v, for all $v \in V$
    - Node type : $V^T \cup V^I \cup V^D = V$
  - ✓ State vector of edges : $e_{(u,v)}$, for all $(u, v) \in E$

- Action (only defined for V^I)
  - ✓ For each time step t,

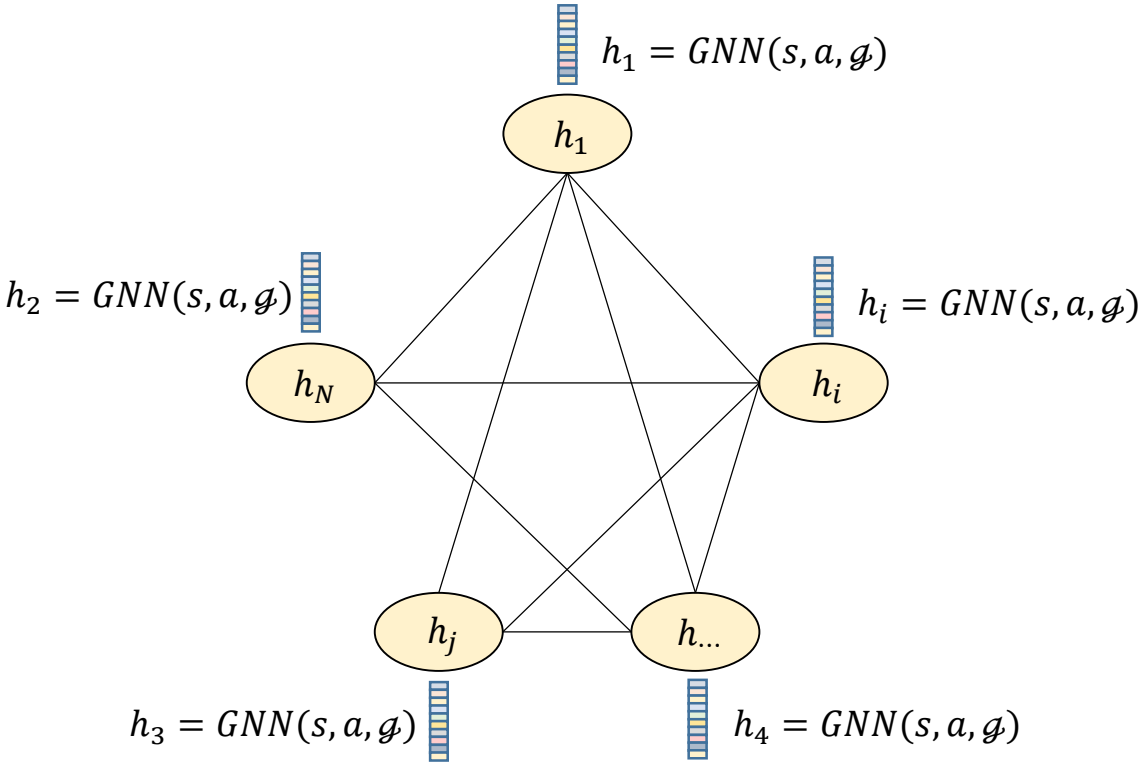
$$a^t_{v \in V^I} \in \{left, right\}$$

- Reward
  - ✓ r^t = total distance of all vehicles

- Objective
  - ✓ **Maximize accumulated total distance ⇔ Maximize mean velocity of vehicles in OHT system**

4. Applications : Congestion Management of OHTs in Semiconductor Fab

Reinforcement Learning Formulation: Actor-Critic Algorithm



Centralized Critic: $V(s) \approx NLP\left(\sum_{i=1}^{N} h_i^{(T)}; \phi\right)$

$\nabla_\theta J(\pi_\theta)$

Decentralized actor: $\pi_i(a_i | h_i) \approx NLP(h_i; \phi)$

Parameter sharing

4. Applications : Congestion Management of OHTs in Semiconductor Fab

Reinforcement Learning Formulation: Actor-Critic Algorithm

- **Critic Network**
  - Centralized State value: $V(s) = \text{NLP}\left(\sum_{i=1}^{N} h_i^{(T)}\right)$ or $\text{NLP}\left(\left[h_1^{(T)}, \ldots, h_N^{(T)}\right]\right)$

- **Actor Network:**
  - Policy
    - ✓ $\pi(a|s;\theta) = \prod_{i \in N} \pi(a_i|s;\theta)$
    - ✓ $\pi(a_i|h_i;\theta) \Rightarrow \pi(a_i|GNN(s);\theta)$
    - ✓ All action nodes select action individually by using its node embedding value
  - Action is selected by soft-max function (left of right)

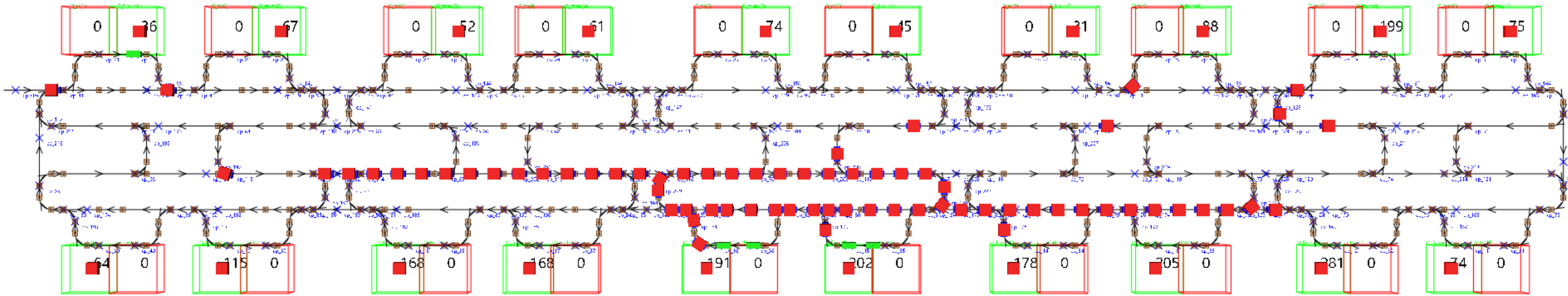
4. Applications : Congestion Management of OHTs in Semiconductor Fab

Learning & Simulation Results

24hr warm-up/12hr execution

Action Interval: 3 sec
Without Dijkstra Algorithm

	GNN	Greedy	Random	FIFO
Waiting Average Time	86	167	232	883
Delivery Time (tbl)	22046	19880	23858	23322
Delivering Trips Made	319	340	287	265
Delivering Average Time	121.25	111.51	136.54	147.75
Retrieving Average Time	14.22	15.83	14.03	15.65



Vehicle Number = 35, 45, 55, 65, 75

Acceleration/Deceleration rate

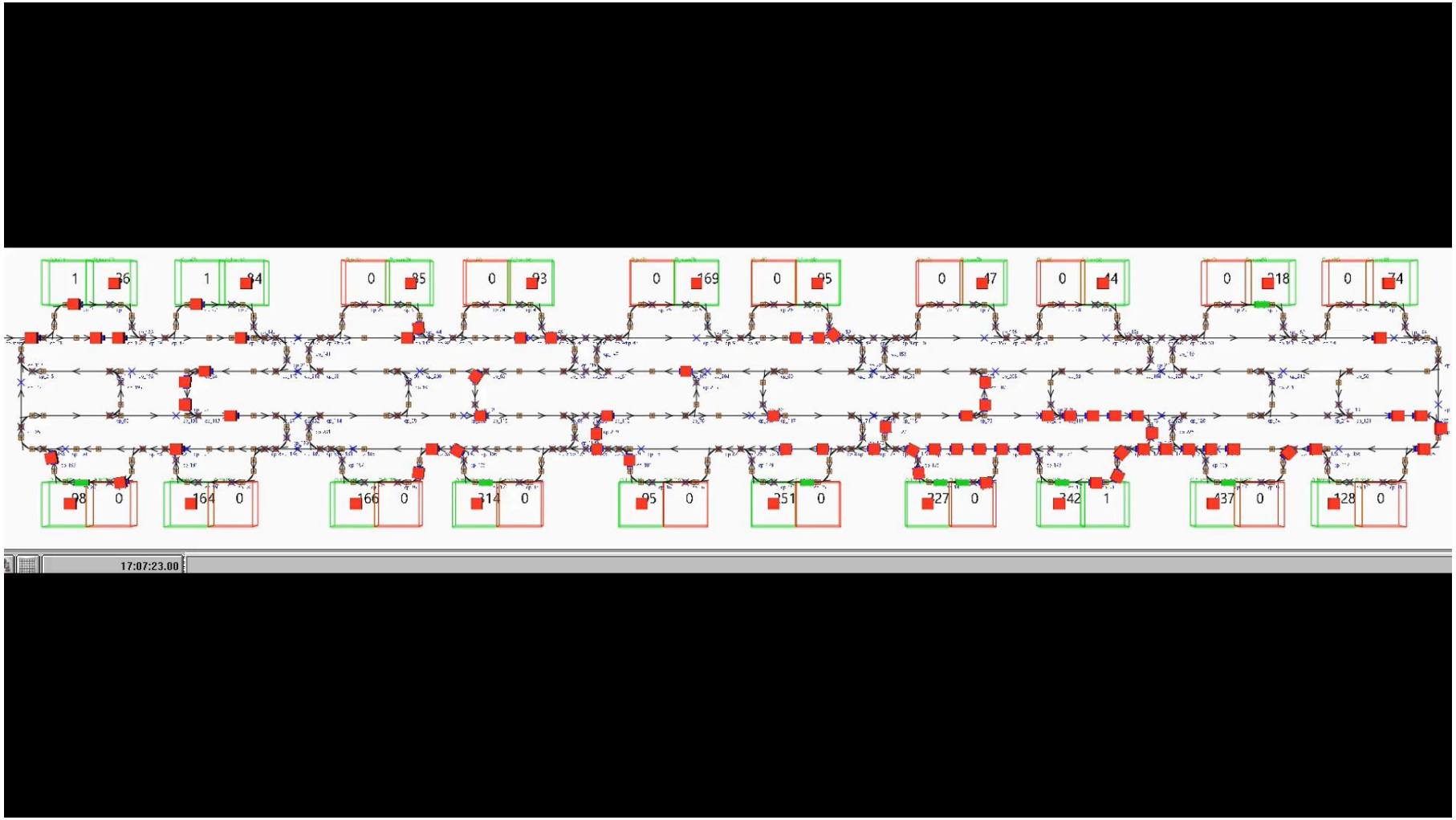
N-step TD (N=1,4,10)

Algorithm: DQN, REINFORCE

4. Applications : Congestion Management of OHTs in Semiconductor Fab

Results

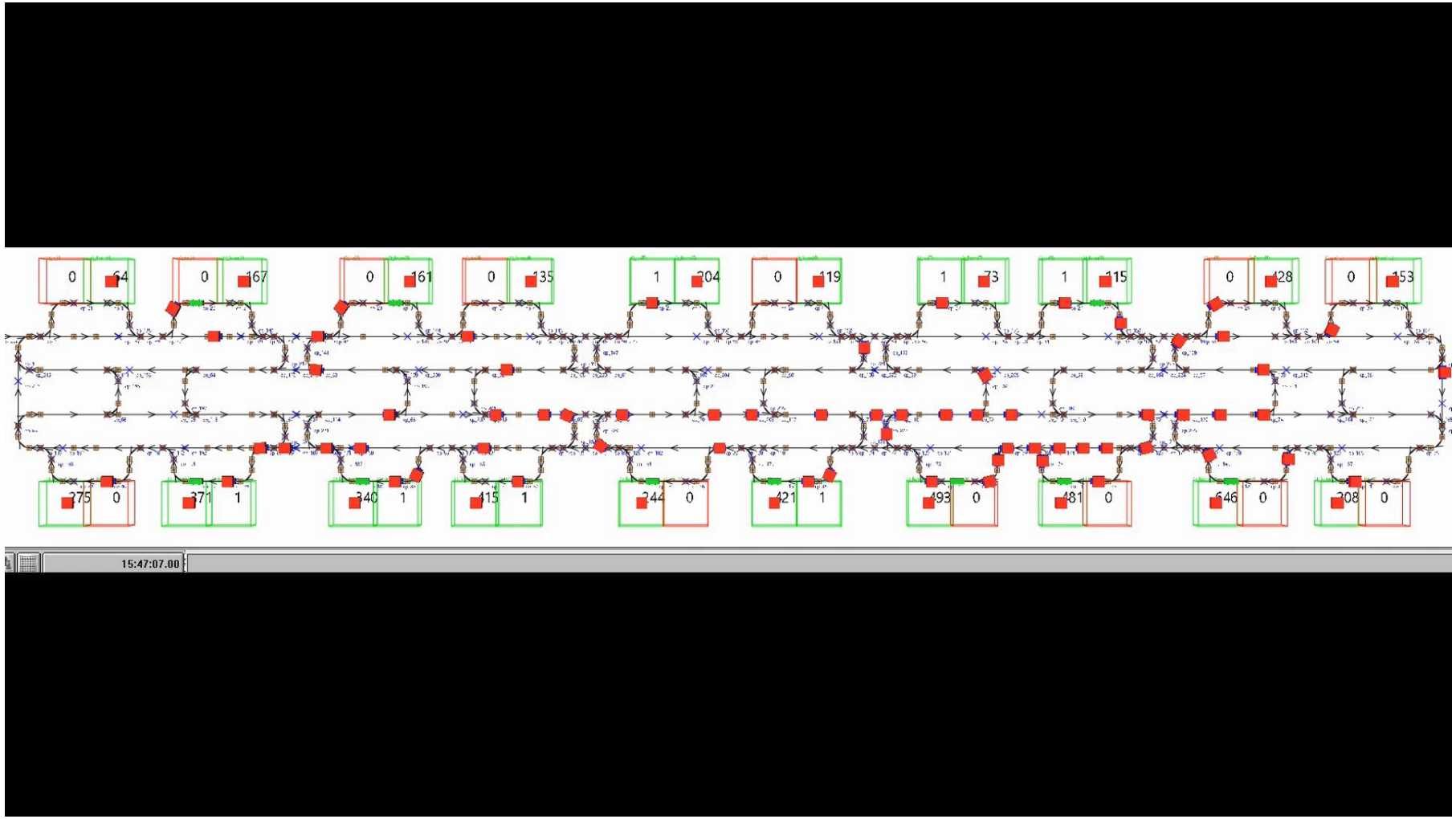
GNN : deadlock 탈출 시 4~5분 소요



4. Applications : Congestion Management of OHTs in Semiconductor Fab

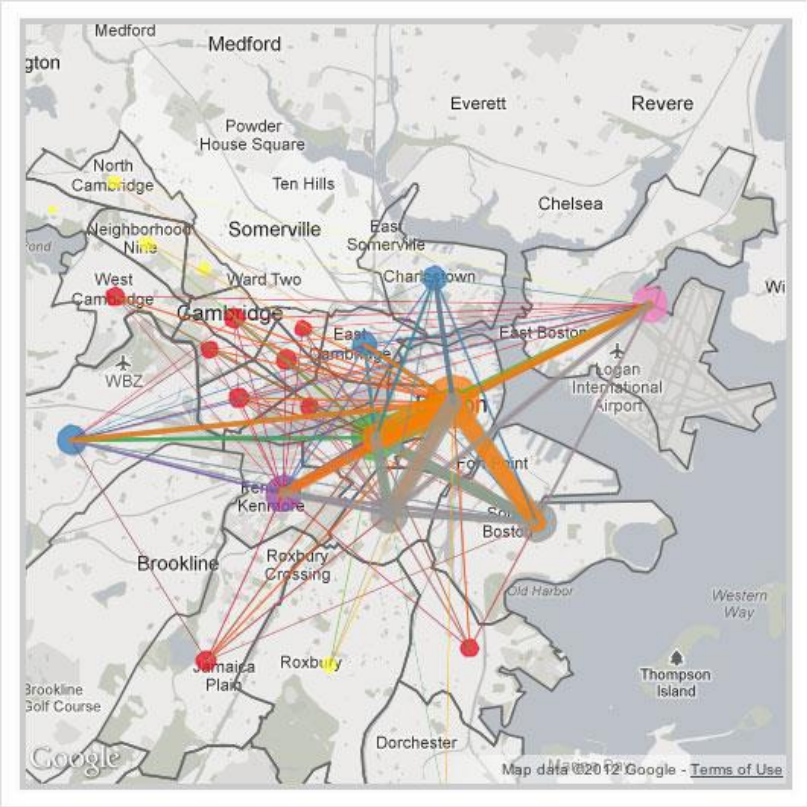
Results

FIFO : deadlock 탈출 시 30~90분 소요

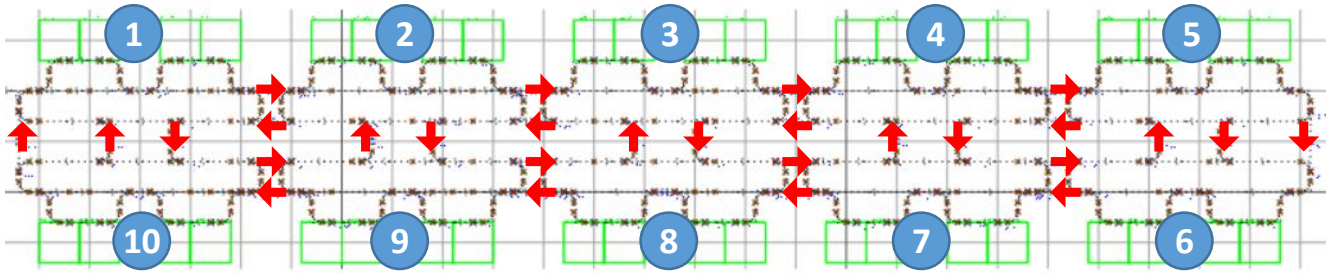


4. Applications : Rebalancing of OHTs in Semiconductor Fab.

Motivation



Tool based Zone



- Which area in a city an empty taxi should go to make more money?
- Which zone in a fab an idle OHT should go to deliver more parts?

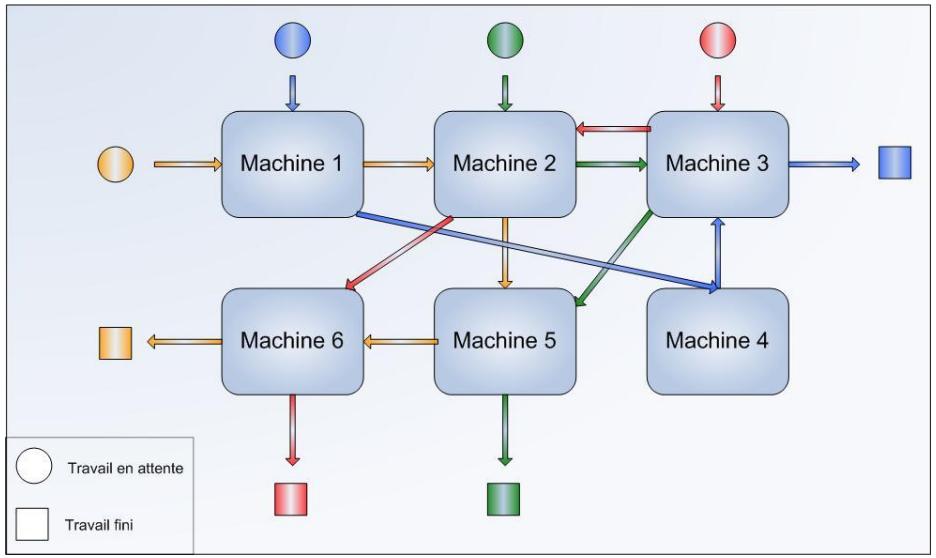
4. APPLICATIONS

Decentralized Job Shop Scheduling

https://en.letsgodigital.org/semiconductor-industry/samsung-semiconductor-business/

4. Applications : Job Shops Scheduling

Motivation



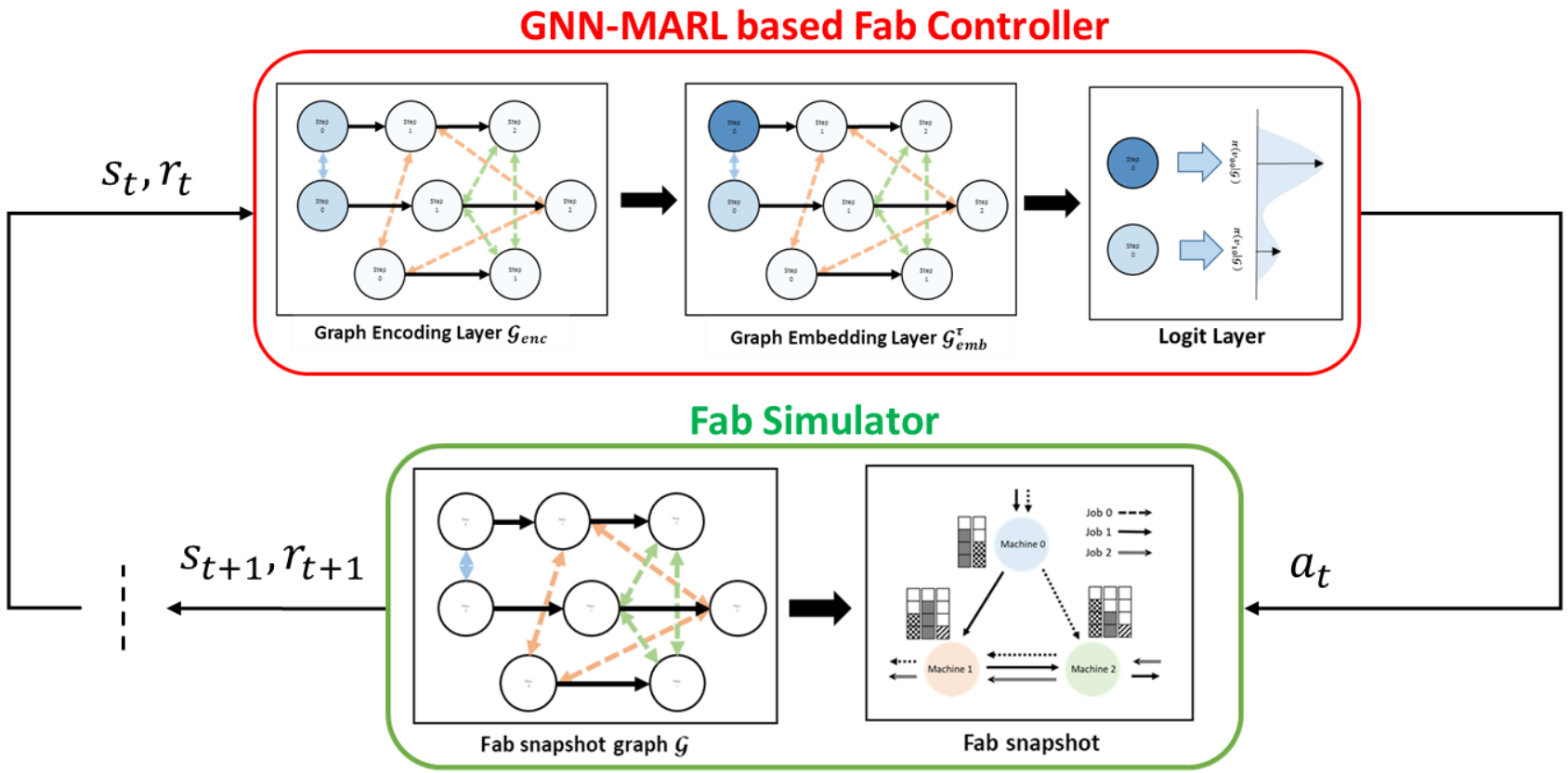
- Job-shop is a manufacturing system.
- **Multiple jobs** (products) are processed on **several machines**.
- Each job consists of a sequence of **operations**, which must be performed in a **given order**.
- Each operation must be processed on a **specific machine.**
- The objective of this problem is **scheduling operations** on the machines to **minimize total completion time (make span)** of all jobs.

- Job-shop scheduling problem is a NP-Hard Problem (combinatorial optimization)

  ✓ **Exact Algorithm** (Mixed Integer Programming)
  ✓ **Approximation Algorithm** (Dual approximation scheme)
  ✓ **Meta-Heuristic Algorithm** (Evolutionary algorithms)

4. Applications : Job Shops Scheduling

GNN-MARL based Job-Shop Controller training overview



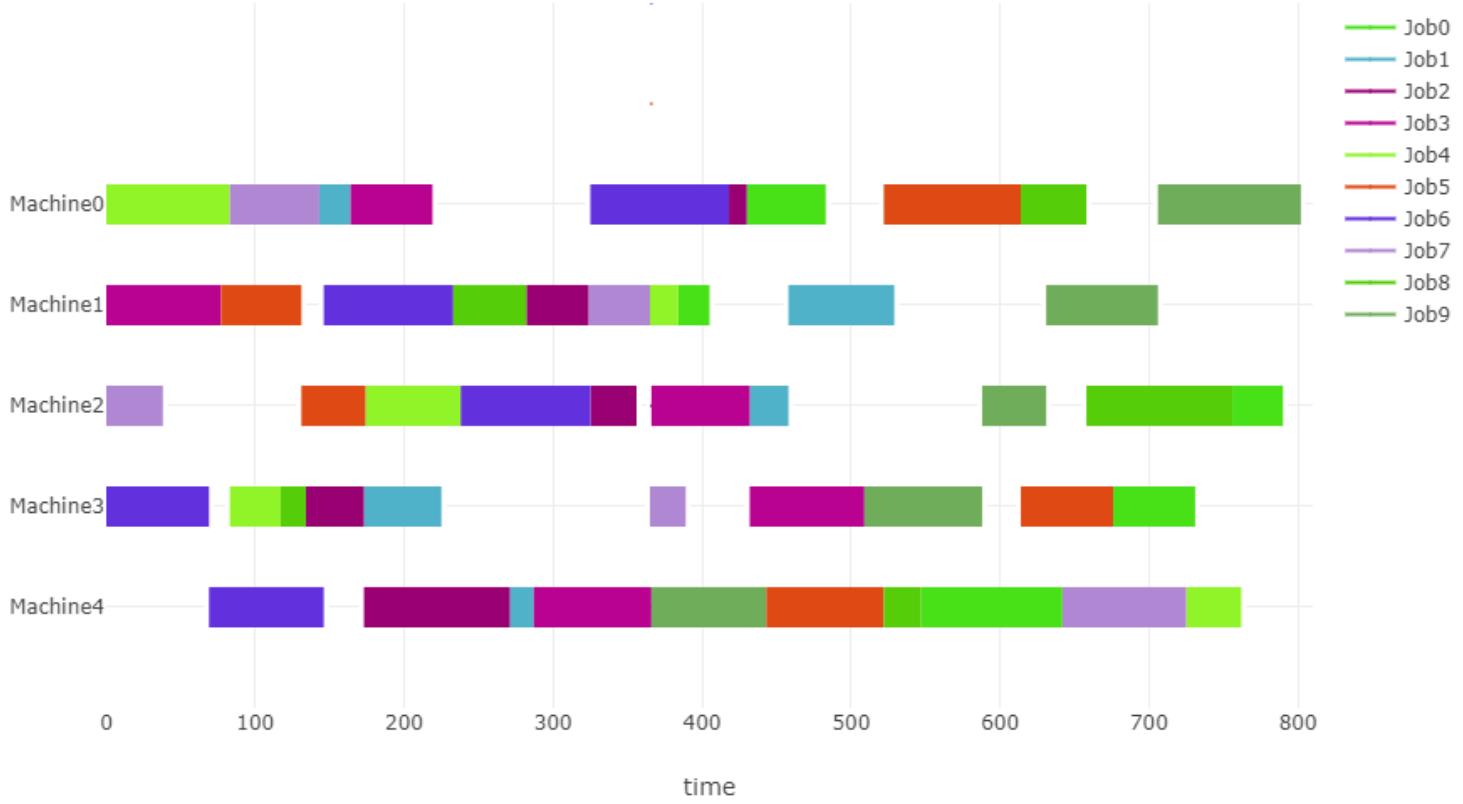
$\boldsymbol{i}$ *is an episode index*

Simulation time $\boldsymbol{t = 0, \ldots, T_i}$

We trained GNN_{Θ} and Job-shop controller θ in **end-to-end fashion** with reinforcement learning framework

4. Applications : Job Shops Scheduling

GNN-MARL based Job-Shop Controller training overview (Episode)



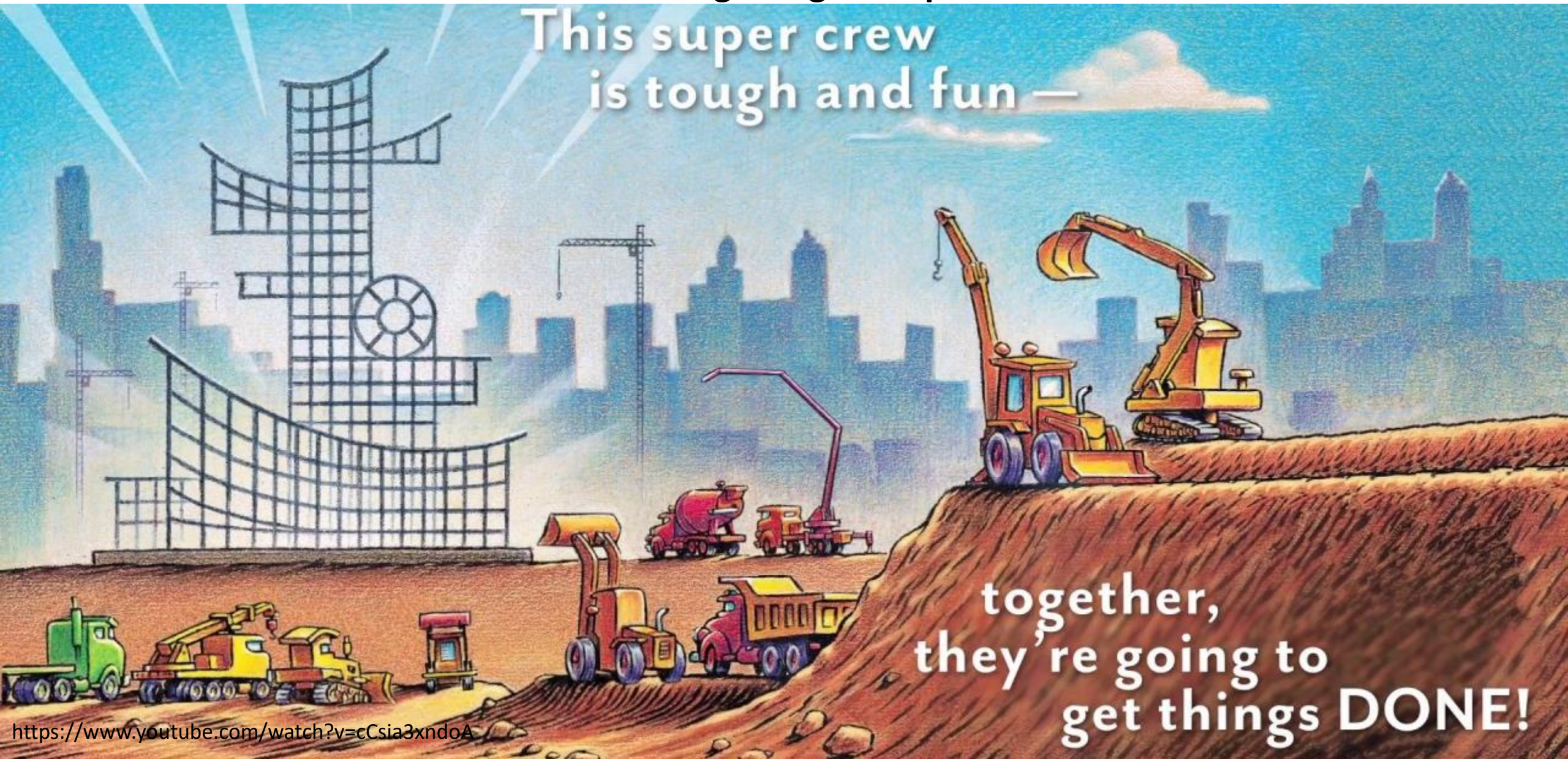
Complete JSP schedule

When an JSP instance is given, the agent (GNN based Job-Shop Controller) can derive a complete JSP schedule

We refer an episode to a complete JSP schedule

4. APPLICATIONS

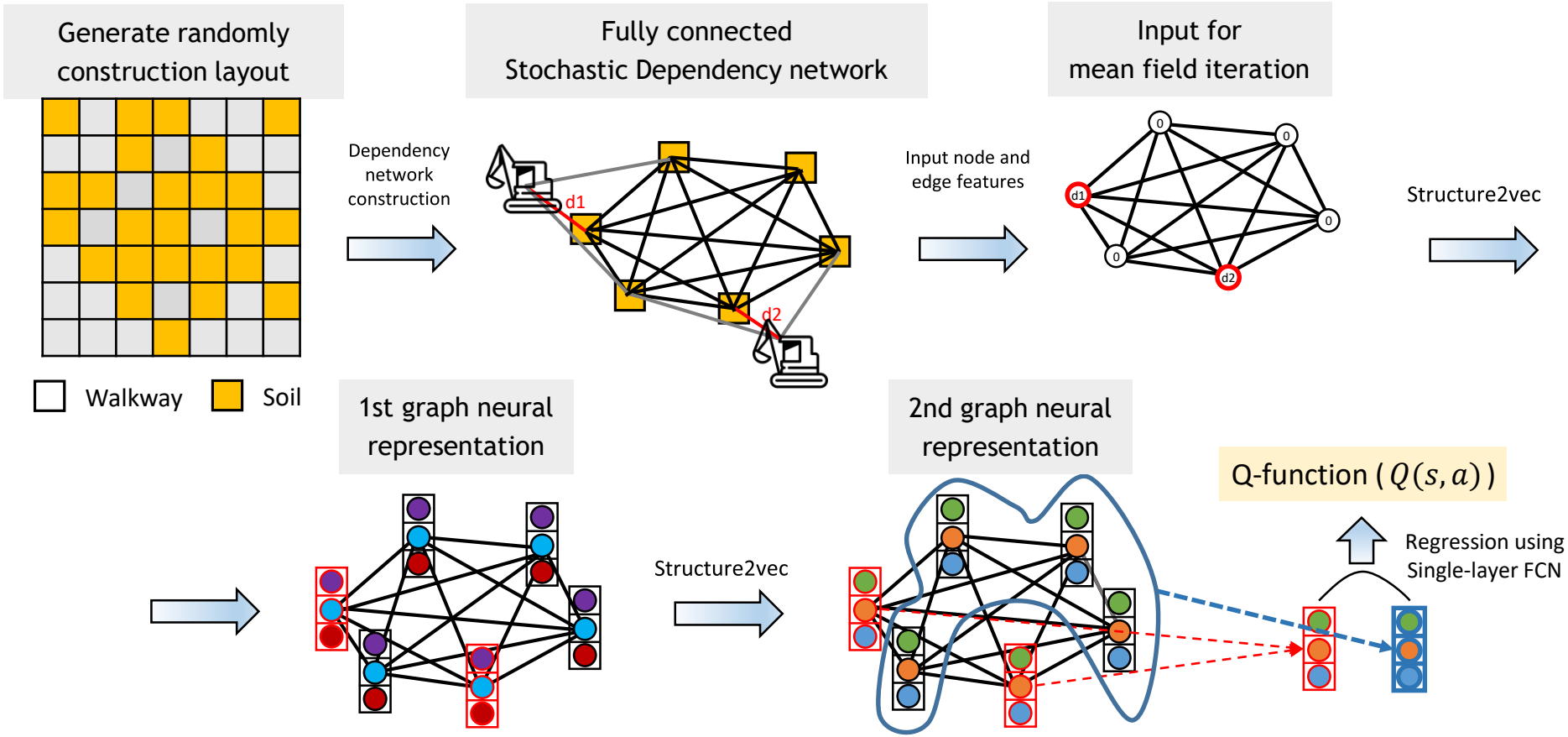
Earthwork Planning using Multiple Excavators



https://www.youtube.com/watch?v=cCsia3xndoA

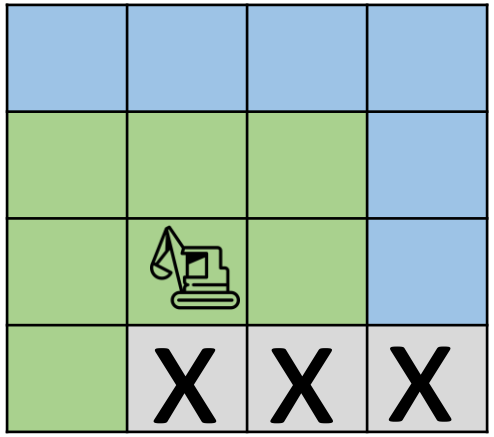
4. Applications : Multi-Agent Vehicle Routing Problem

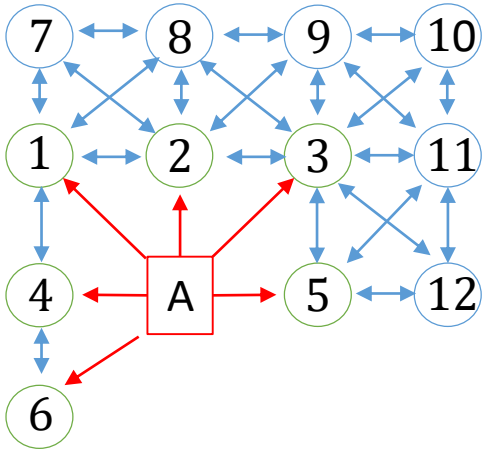
Overview



4. Applications : Multi-Agent Vehicle Routing Problem

Graph & State Representation





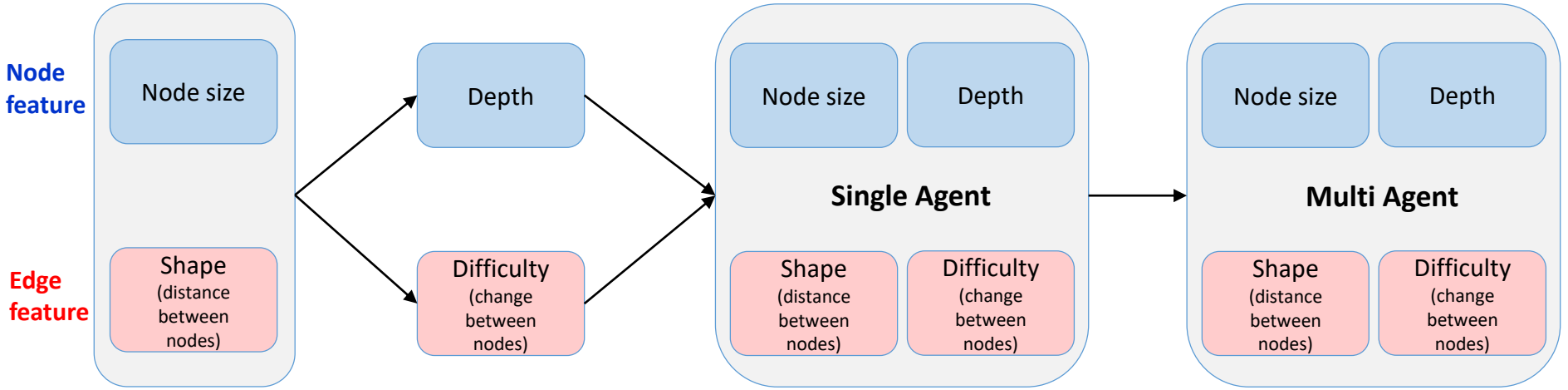
decision edge

travel edge: soil node

soil node adjacent to agent

soil node not adjacent to agent

A agent



4. Applications : Multi-Agent Vehicle Routing Problem

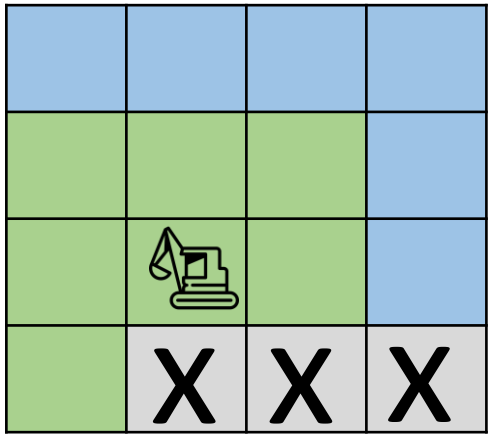
Node Embedding

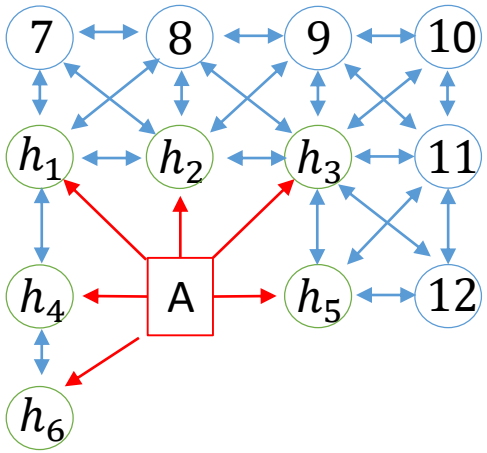
$$for\ t = 1{:}T,$$
$$for\ all\ node\ v \in V,$$
$$\mu_v^{(t+1)} \leftarrow relu\left(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \mu_u{}^{(t)} + \theta_3 \sum_{u \in \mathcal{N}(v)} relu\big(\theta_4 w(u,v)\big)\right)$$

- $\mu_v^{(t+1)}$: Embedding vector (Hidden vector)
- x_v: Node attribute
- $\mu_u{}^{(t)}$: Previous hidden vectors
- $\mathcal{N}(v)$: Neighbors of v
- $w(u,v)$: Edge attribute

4. Applications : Multi-Agent Vehicle Routing Problem

Action Selection





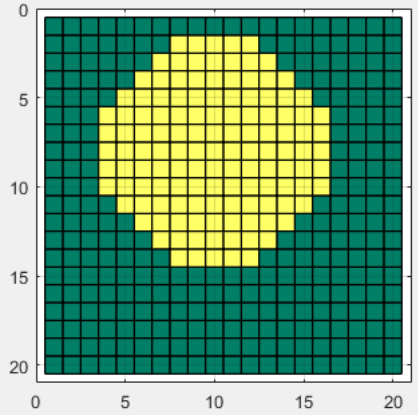
$$Q(s, a_1) \approx f(h_1; \theta) = f(GNN(s); \theta)$$

$$a^* = \max_{a \in Neigh} Q(s, a)$$

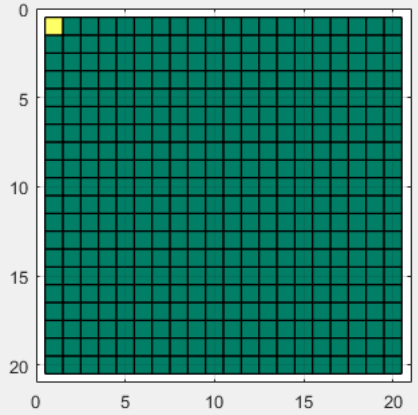
4. Applications : Multi-Agent Vehicle Routing Problem

Learning & Simulation Results : Transfer to different Agent Number

DQN-20

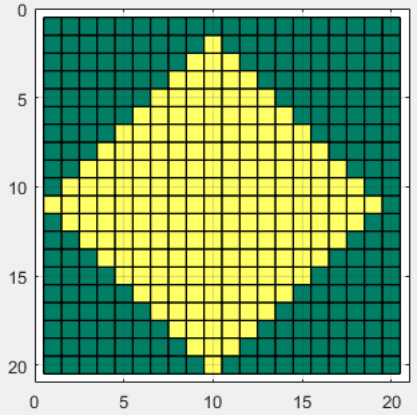


Soil Status - tick 1 , ACTION : 0 0

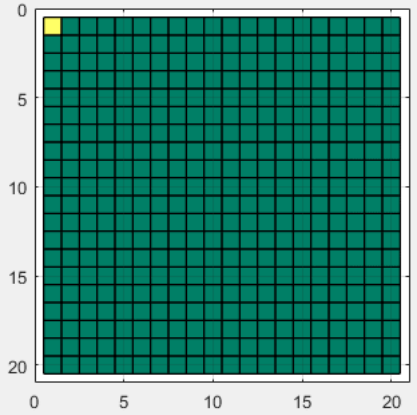


Digger Postion - tick 1 , ACTION : 0 0

DQN-20



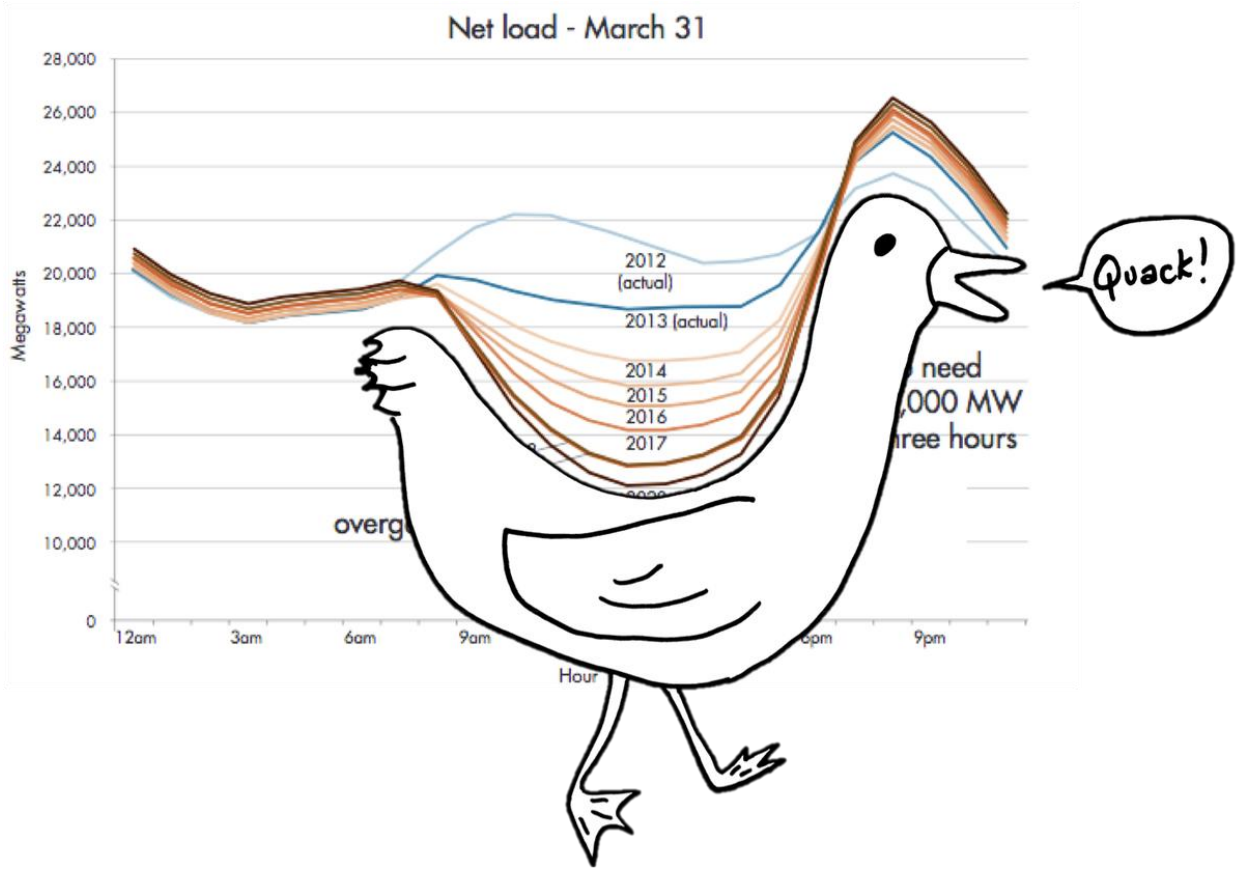
Soil Status - tick 1 , ACTION : 0 0



Digger Postion - tick 1 , ACTION : 0 0

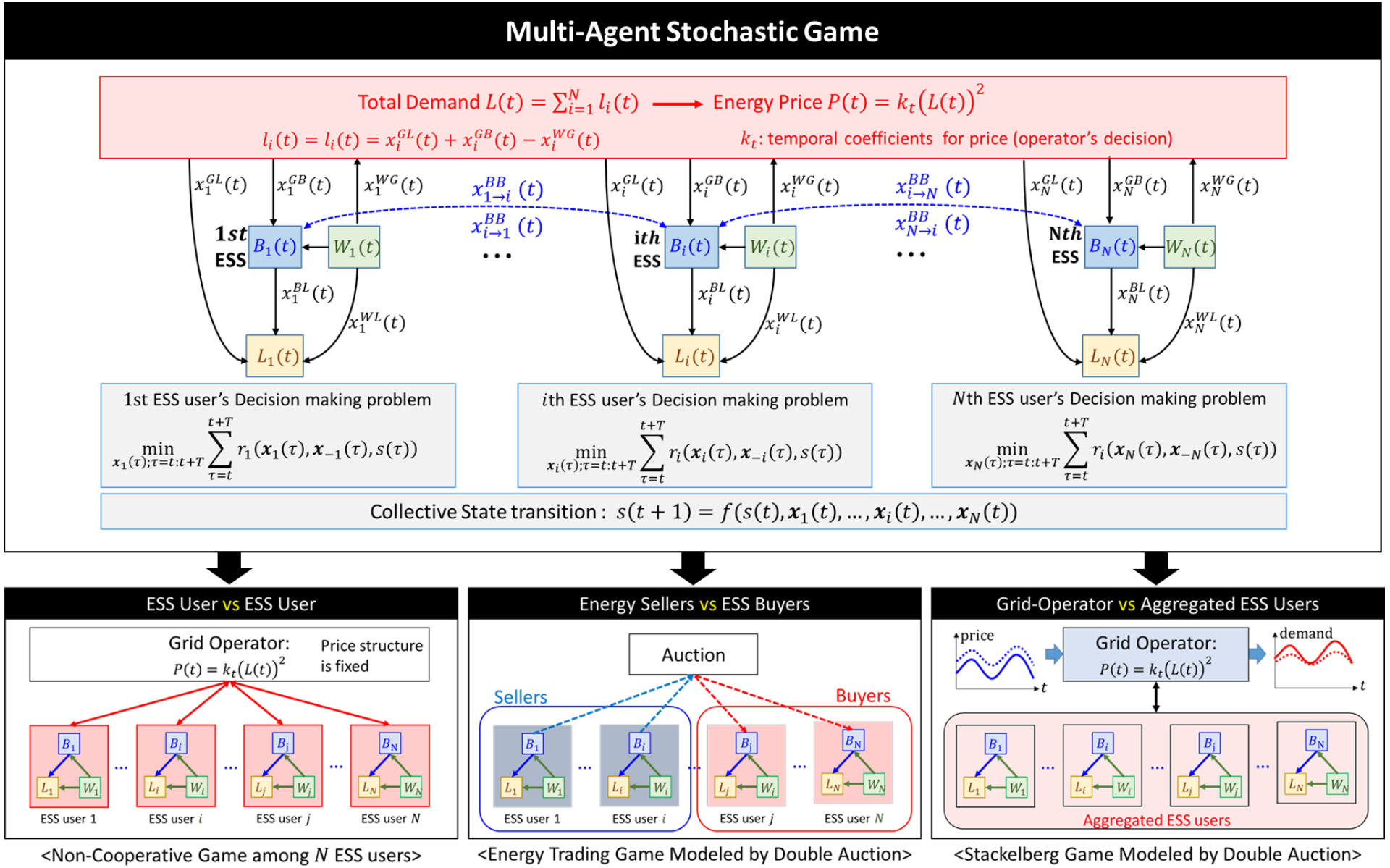
4. Applications : Multiple Energy Storage System Control

Motivation



4. Applications : Multiple Energy Storage System Control

Formulation



<Non-Cooperative Game among N ESS users>

<Energy Trading Game Modeled by Double Auction>

<Stackelberg Game Modeled by Double Auction>

Q & A