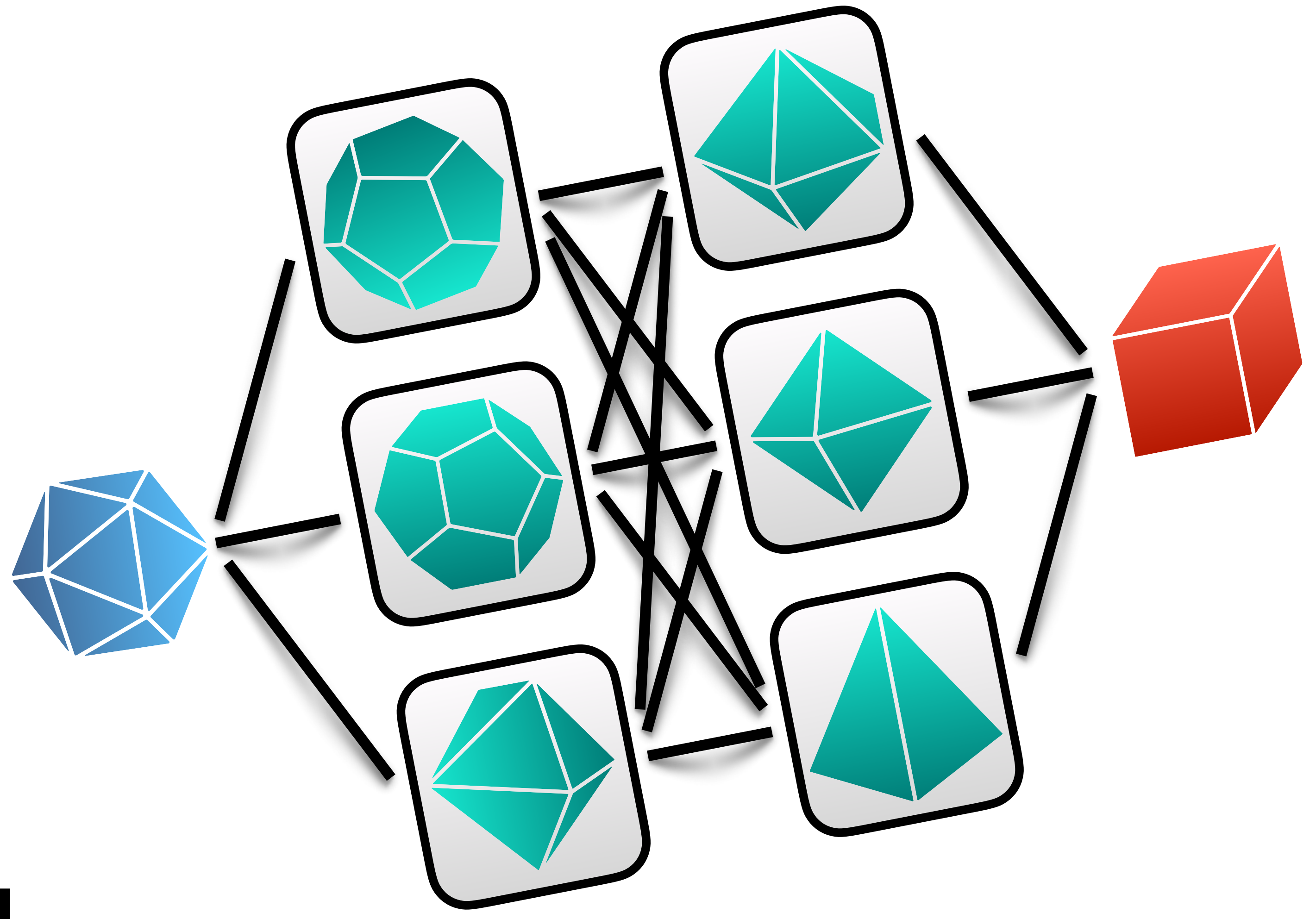
Did I forget to hit record? Please remind me!



Reinforcement Learning for Algorithms

MIE1666: Machine Learning for Mathematical Optimization

Based in part on Mazyavkina, Nina, et al. "Reinforcement learning for combinatorial optimization: A survey." Computers & Operations Research (2021): 105400.

Based in part on Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

Elias B. Khalil — 01/11/21

UNIVERSITY OF TORONTO

Characteristics of Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

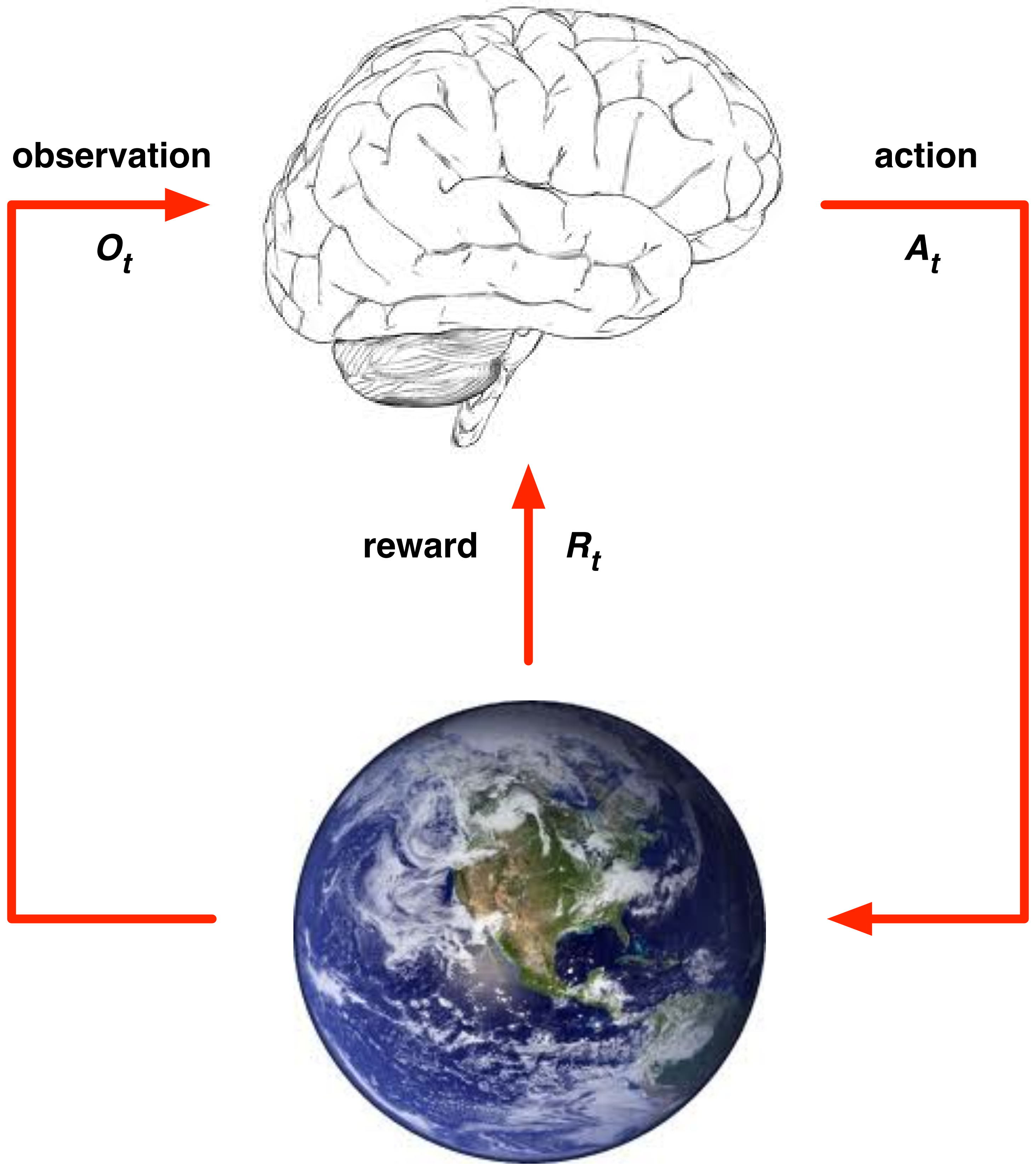
- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

Sequential Decision Making

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

History and State

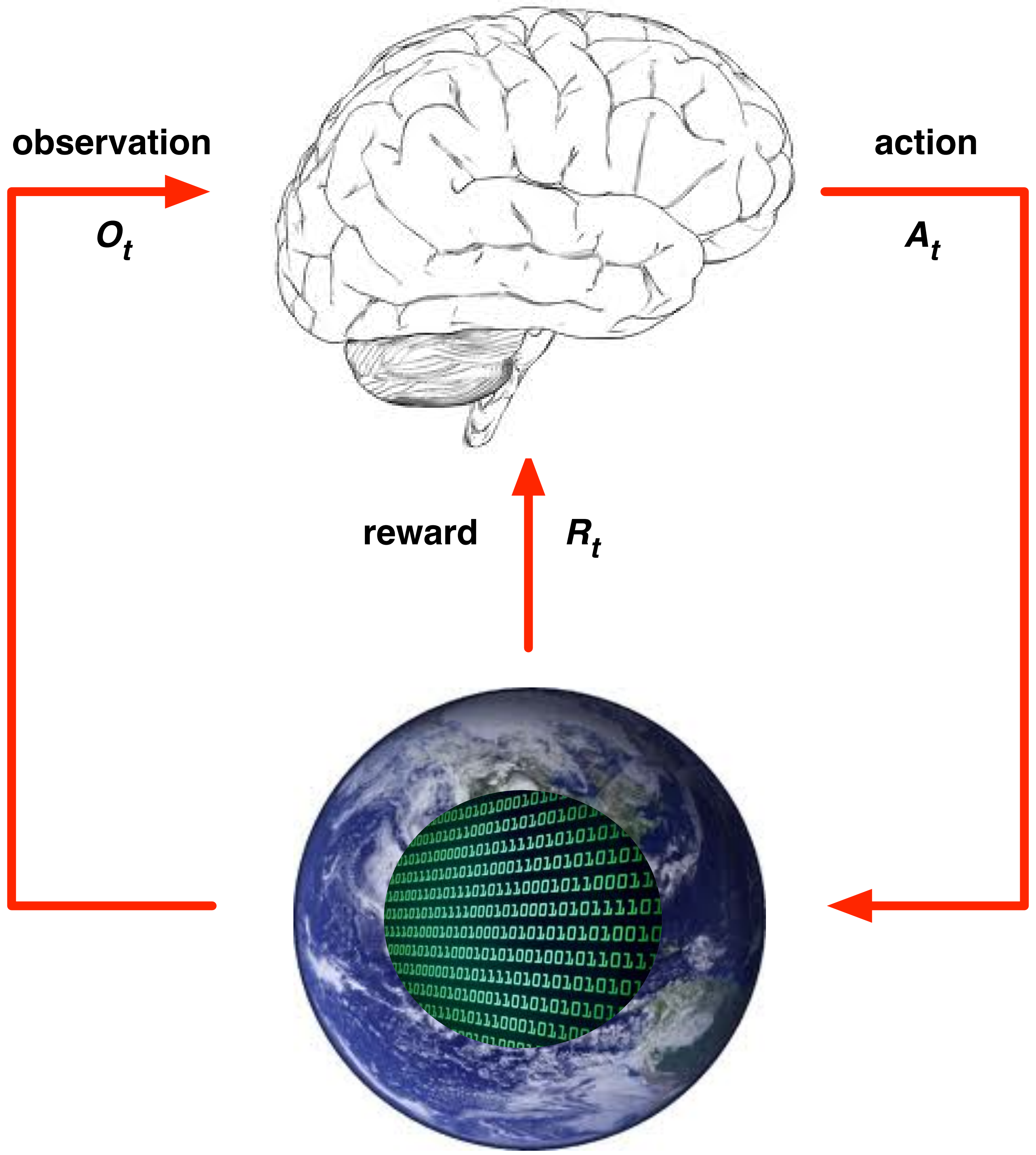
- The history is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- i.e. all observable variables up to time t
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- State is the information used to determine what happens next
- Formally, state is a function of the history:

$$S_t = f(H_t)$$

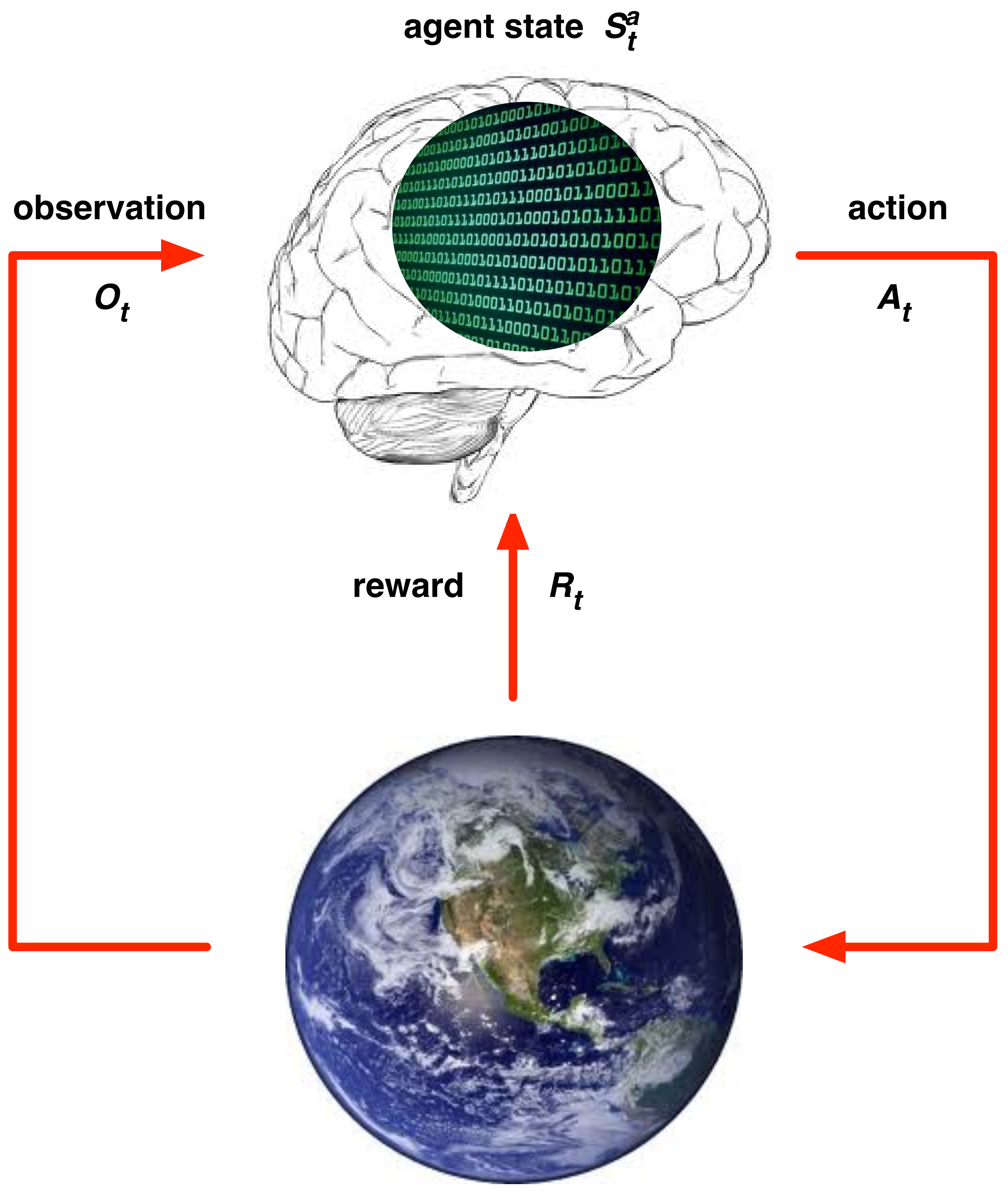
Environment State



- The environment state S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward
- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

Agent State



- The agent state S_t^a is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms
- It can be any function of history:

$$S_t^a = f(H_t)$$

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

Information State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state S_t is Markov if and only if

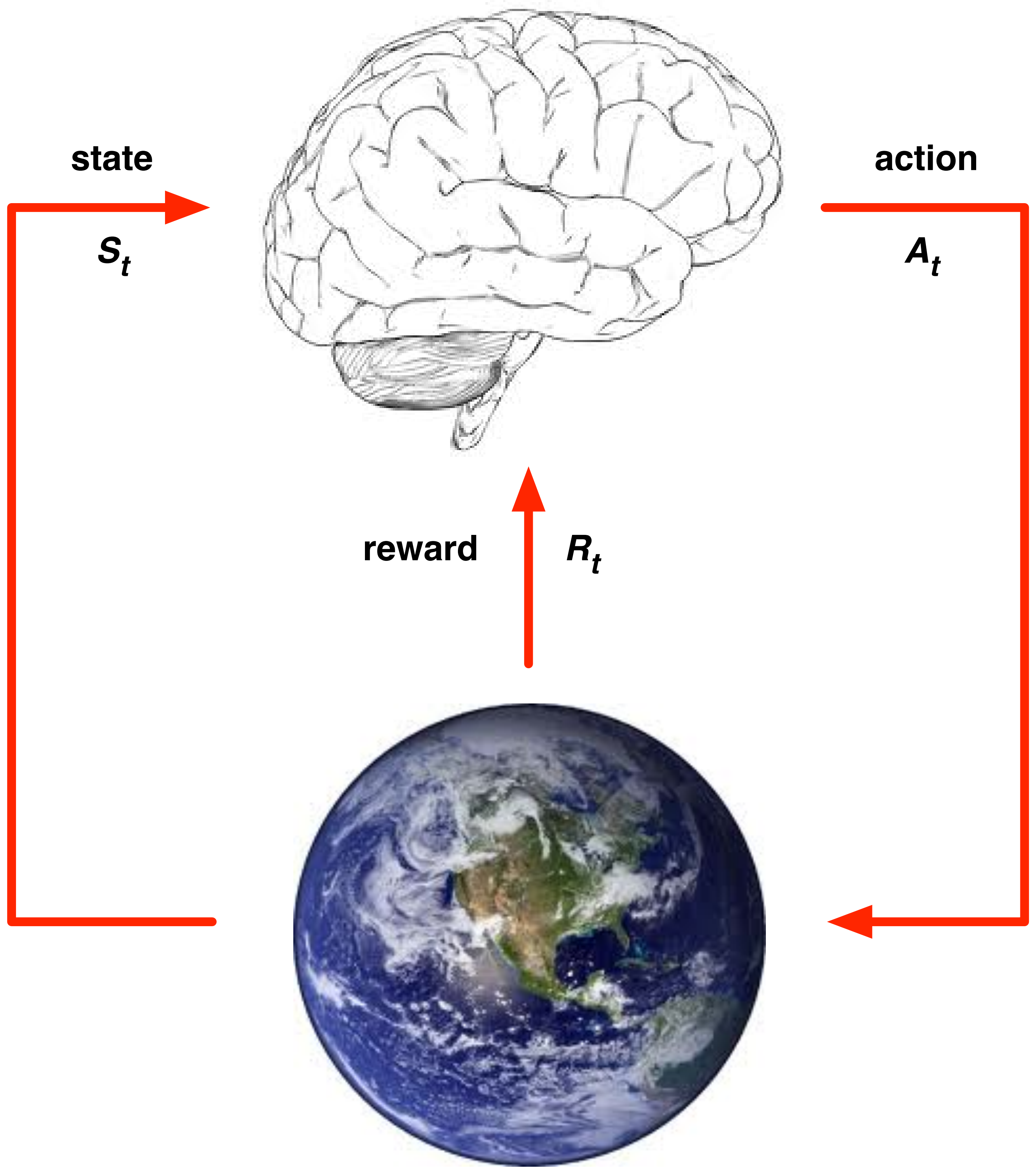
$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- "The future is independent of the past given the present"

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state S_t^e is Markov
- The history H_t is Markov

Fully Observable Environments



Full observability: agent directly observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment state = information state
- Formally, this is a Markov decision process (MDP)
- (Next lecture and the majority of this course)

Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

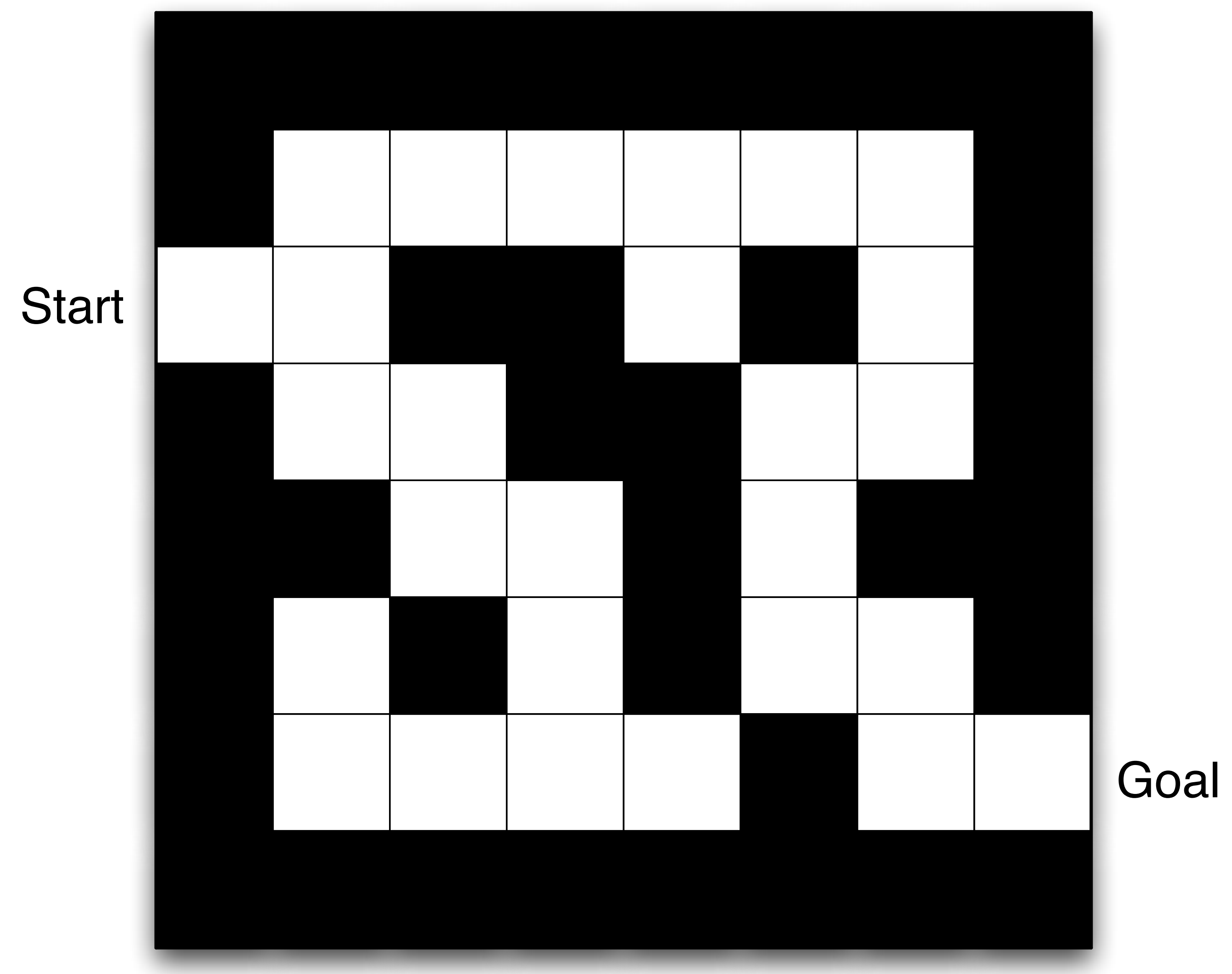
$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \mid S_t = s \right]$$

Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

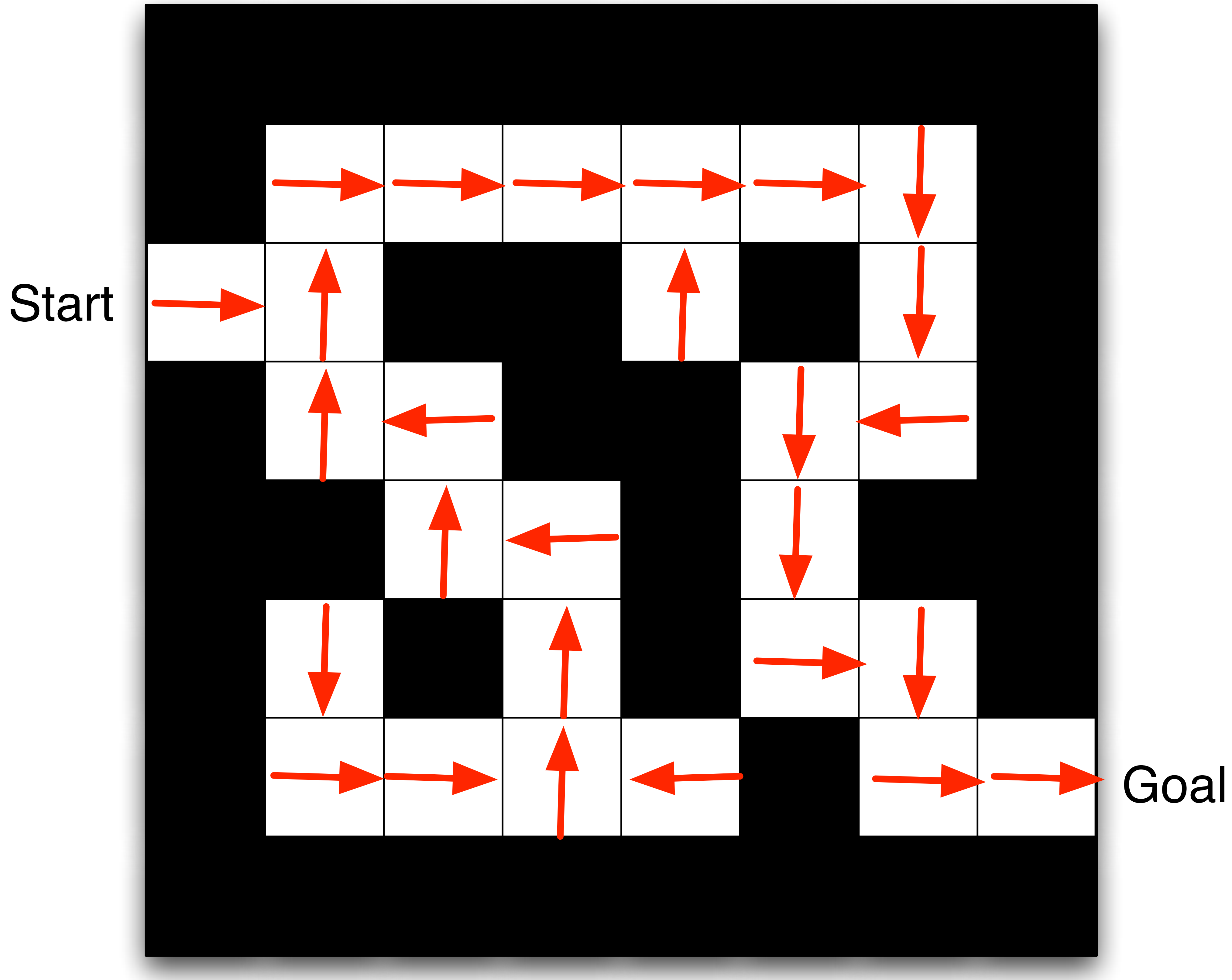
$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

Maze Example



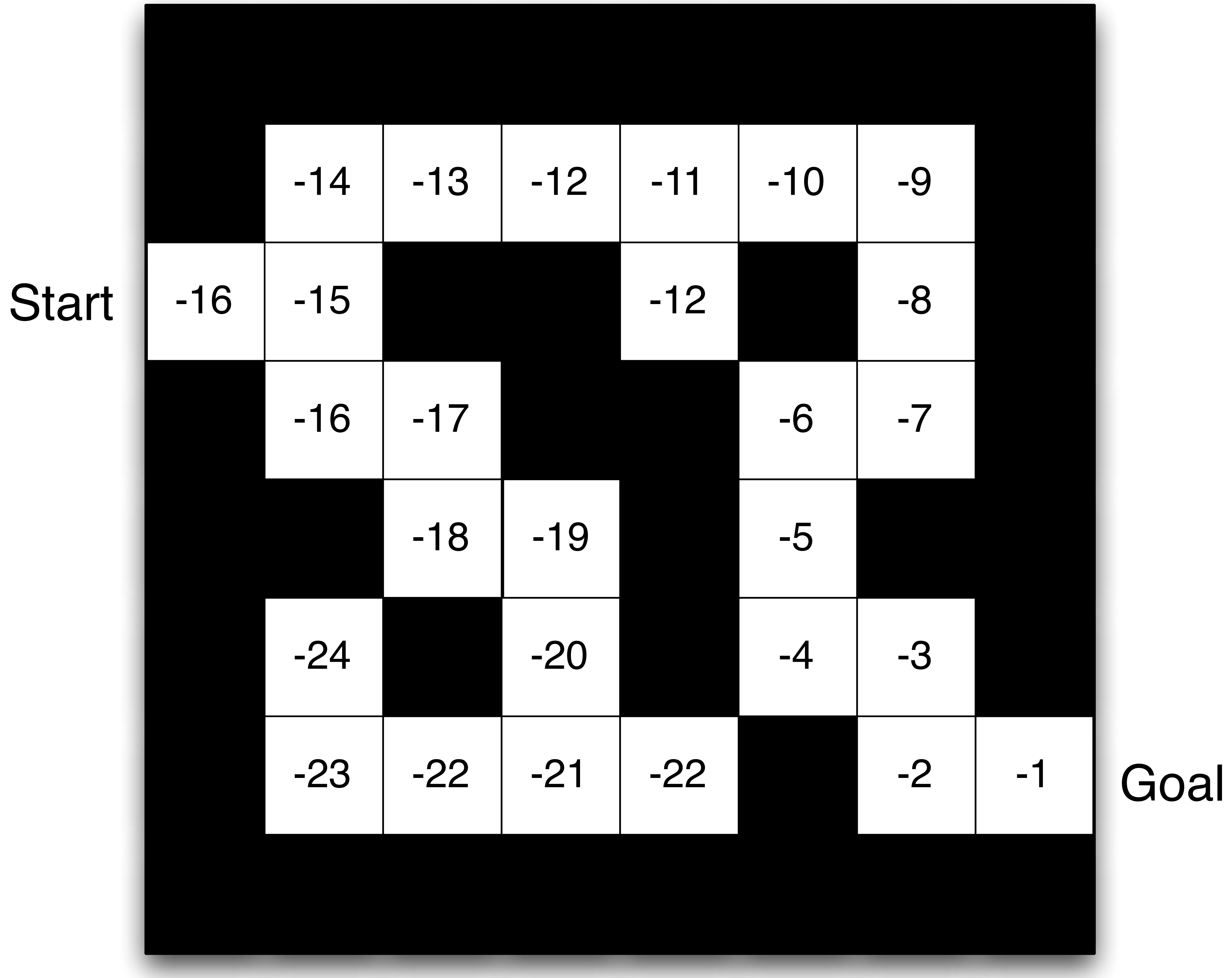
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



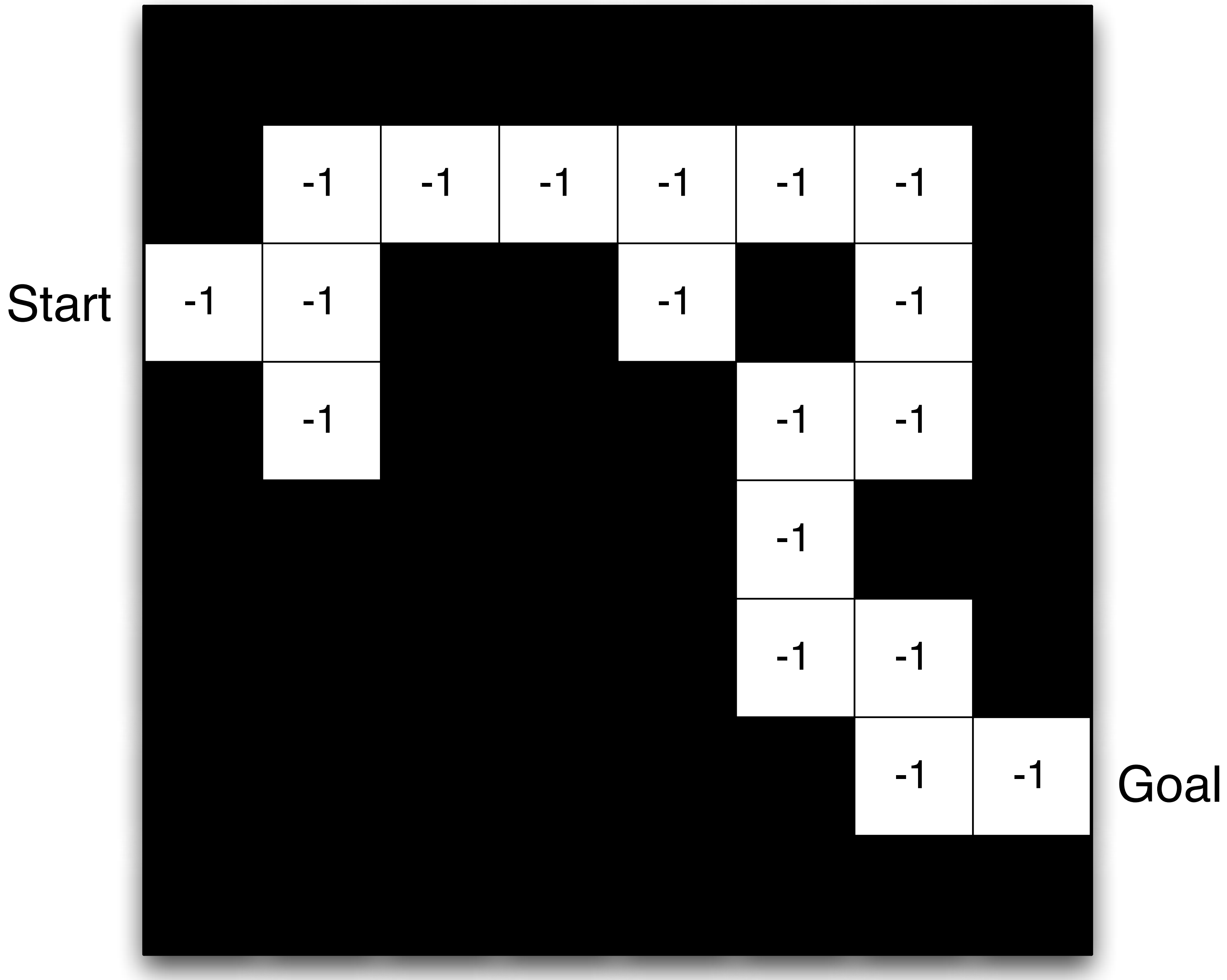
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



- Numbers represent value $v_\pi(s)$ of each state

Maze Example: Model



- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $\mathcal{P}^a_{ss'}$
- Numbers represent immediate reward $\mathcal{R}^a_s$ from each state s (same for all a)

Categorizing RL agents (1)

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

Lecture 1: Introduction to Reinforcement Learning
└ Inside An RL Agent

Categorizing RL agents (2)

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Policy and/or Value Function
  - Model

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

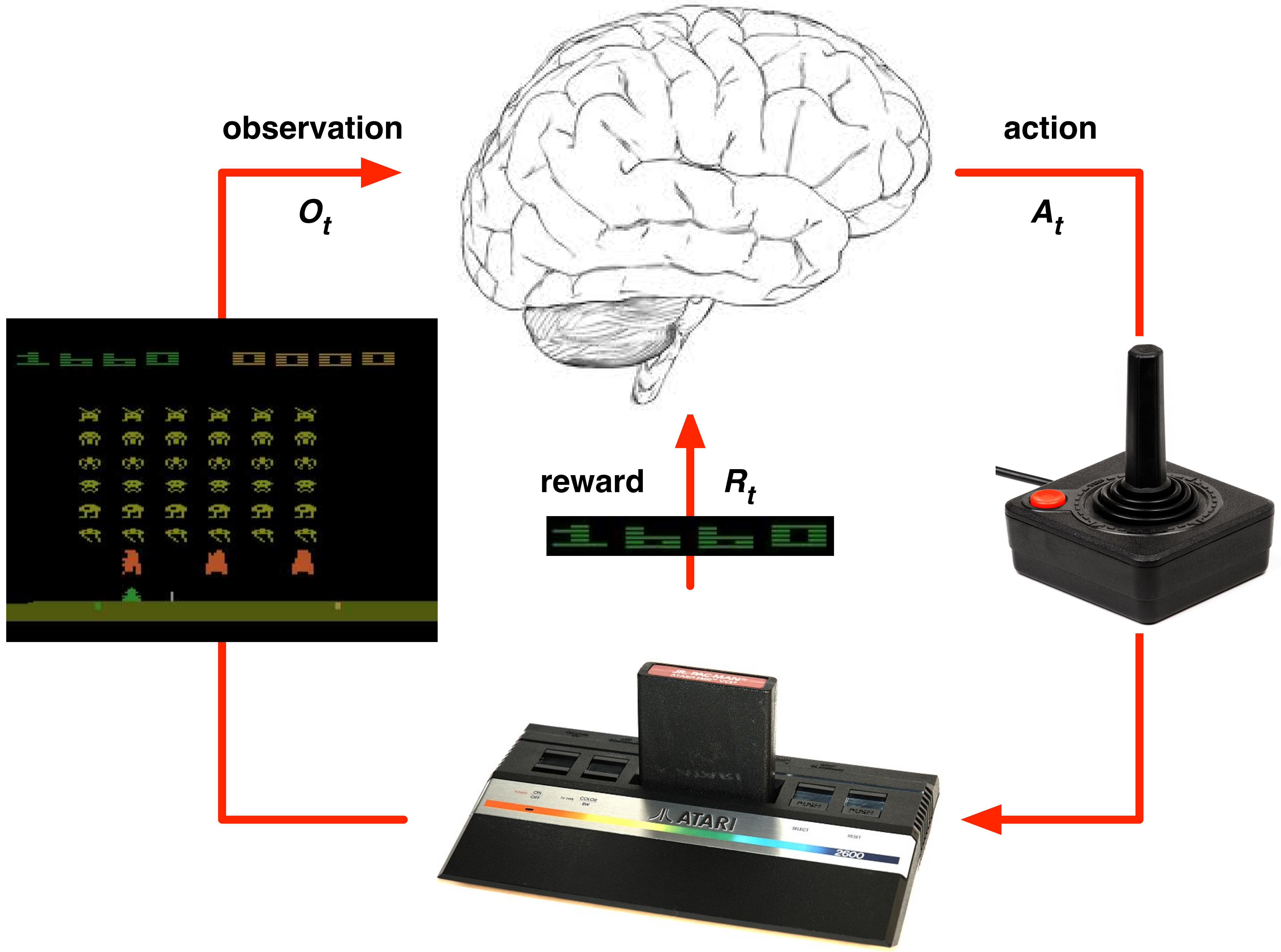
Learning and Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

Atari Example: Reinforcement Learning

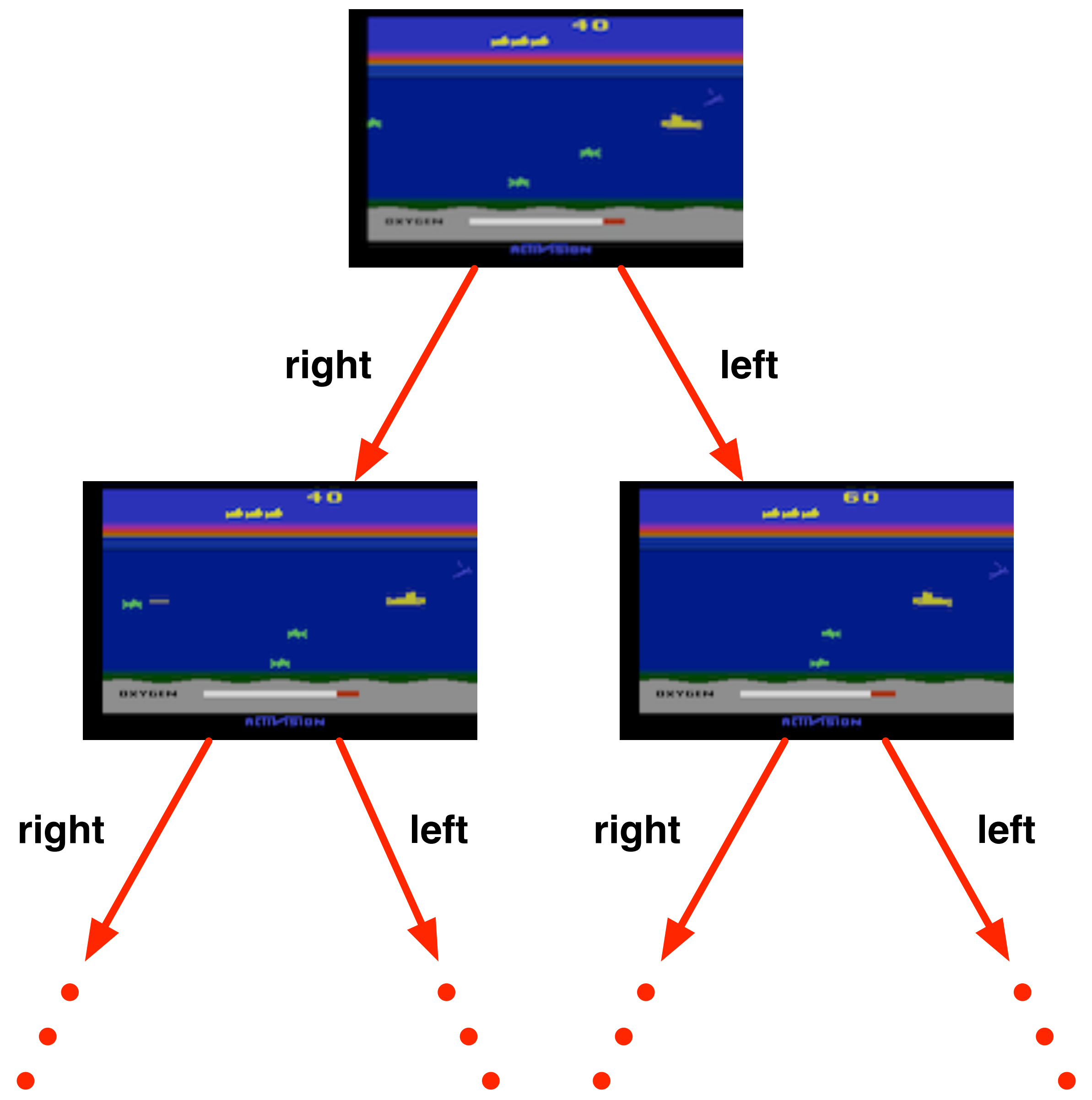


- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Introduction to Reinforcement Learning with David Silver, https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver

Atari Example: Planning

- Rules of the game are known
- Can query emulator
  - perfect model inside agent's brain
- If I take action *a* from state *s*:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  - e.g. tree search



Exploration and Exploitation (1)

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

Exploration and Exploitation (2)

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

Combinatorial Optimization Problem

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j : S \in \mathcal{F} \right\}$$

$N = \{1, \ldots, n\}$ is a finite set,
$c_j \in \mathbb{R}$ is a *weight* for each $j \in N$,
$\mathcal{F}$ is a set of feasible subsets of N.

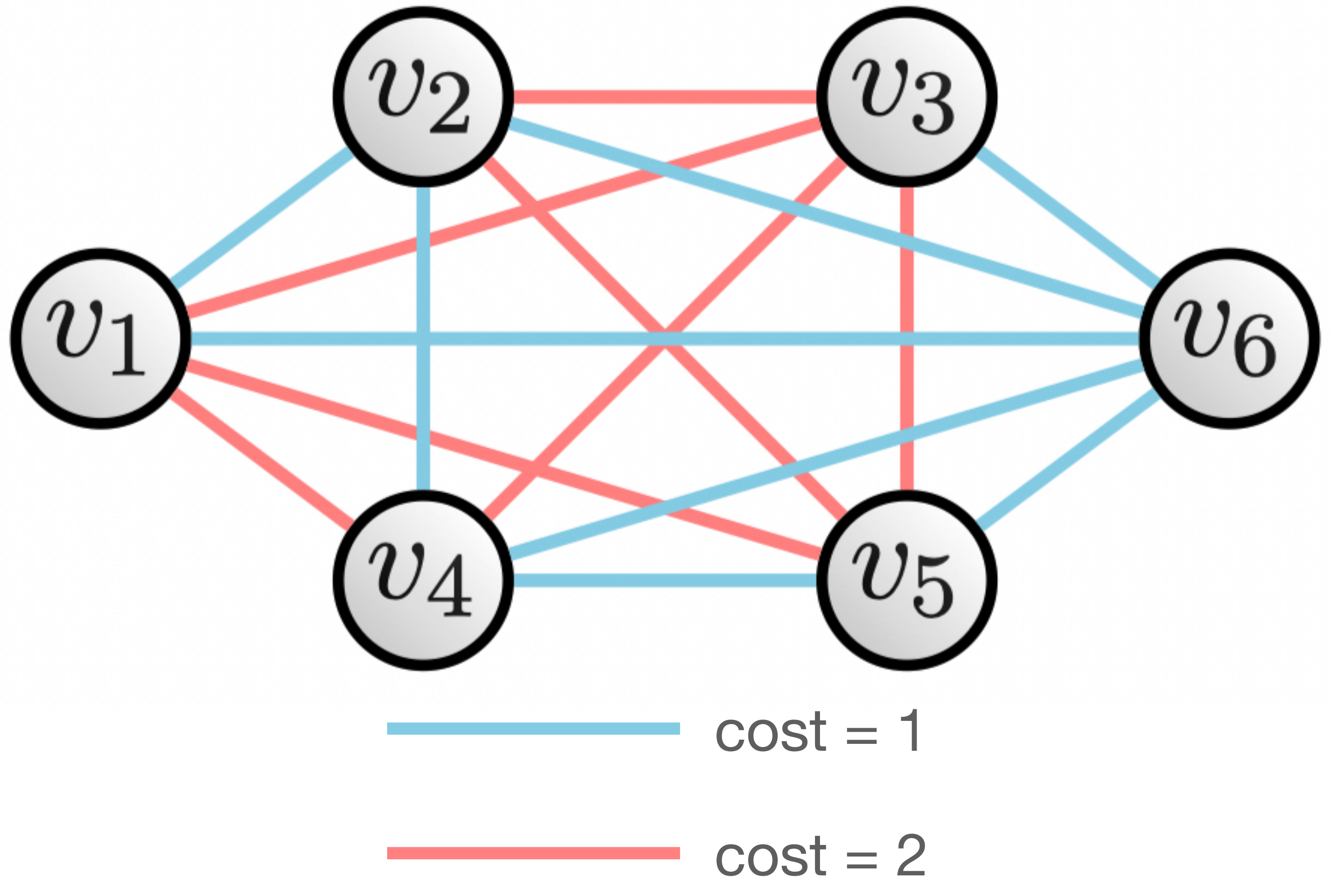
Set Covering Problem

Given a certain number of regions, the problem is to decide where to install a set of emergency service centers. For each possible center, the cost of installing a service center and which regions it can service are known. For instance, if the centers are fire stations, a station can service those regions for which a fire engine is guaranteed to arrive on the scene of a fire within eight minutes. The goal is to choose a minimum cost set of service centers so that each region is covered.

$$\min_{T \subseteq N} \left\{ \sum_{j \in T} c_j : \cup_{j \in T} S_j = M \right\}$$

$$M = \{1, \dots, M\} \text{ is the set of regions,}$$
$$N = \{1, \dots, n\} \text{ is the set of potential centers,}$$
$$c_j \in \mathbb{R}^+ \text{ is the per-region installation cost.}$$

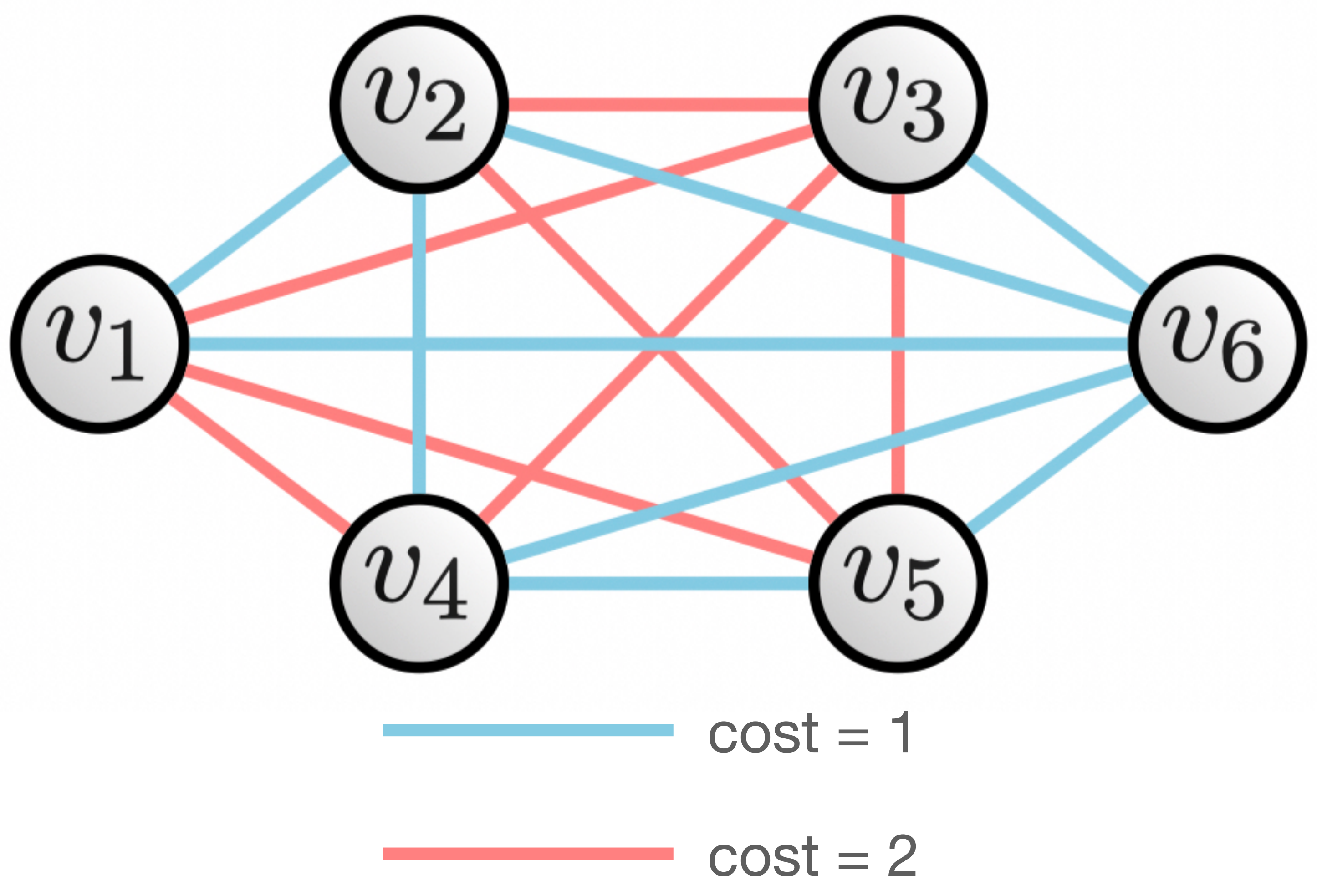
Nearest Neighbour heuristic for the TSP:



- always choose at the current city the closest unvisited city
  - choose an arbitrary initial city $\pi(1)$
  - at the ith step choose city $\pi(i+1)$ to be the city j that minimises $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$
- running time $\mathcal{O}(n^2)$
- worst case performance $NN(x)/OPT(x) \leq 0.5(\lceil \log_2 n \rceil + 1)$
- other construction heuristics for TSP are available

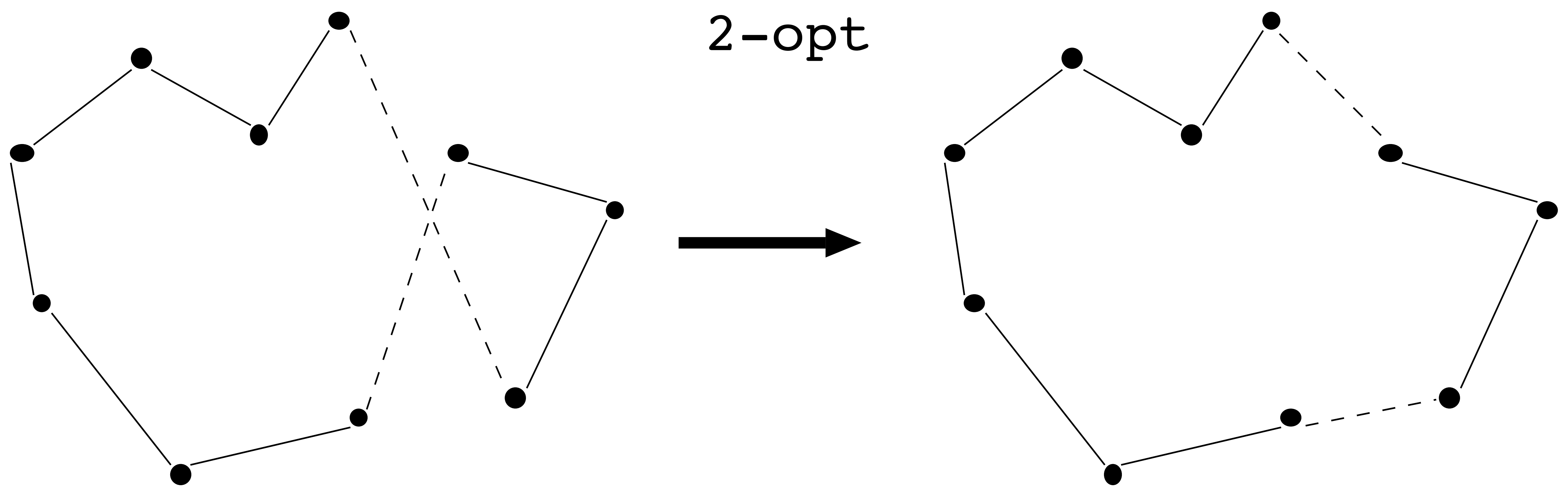
Nearest Neighbour heuristic for the TSP:

- always choose at the current city the closest unvisited city
  - choose an arbitrary initial city $\pi(1)$
  - at the ith step choose city $\pi(i+1)$ to be the city j that minimises $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$



Iterative Improvement for the TSP

- initial solution is a complete tour
- k-opt neighbourhood: solutions which differ by at most k edges



- neighbourhood size $\mathcal{O}(n^k)$

Definition 2. MDP can be defined as a tuple $M = \langle S, A, R, T, \gamma, H \rangle$, where

- S - *state space* $\mathbf{s}_t \in S$. State space for combinatorial optimization problems in this survey is typically defined in one of two ways. One group of approaches constructs solutions incrementally define it as a set of partial solutions to the problem (e.g. a partially constructed path for TSP problem). The other group of methods starts with a suboptimal solution to a problem and iteratively improves it (e.g. a suboptimal tour for TSP).
- A - *action space* $\mathbf{a}_t \in A$. Actions represent addition to partial or changing complete solution (e.g. changing the order of nodes in a tour for TSP);
- R - *reward function* is a mapping from states and actions into real numbers $R : S \times A \rightarrow \mathbb{R}$. Rewards indicate how action chosen in particular state improves or worsens a solution to the problem (i.e. a tour length for TSP);
- T - *transition function* $T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ that governs transition dynamics from one state to another in response to action. In combinatorial optimization setting transition dynamics is usually deterministic and known in advance;
- γ - scalar *discount factor*, $0 < \gamma \leq 1$. Discount factor encourages the agent to account more for short-term rewards;
- H - *horizon*, that defines the length of the episode, where episode is defined as a sequence $\{s_t, a_t, s_{t+1}, a_{t+1}, s_{t+2}, \dots\}_{t=0}^{H}$. For methods that construct solutions incrementally episode length is defined naturally by number of actions performed until solution is found. For iterative methods some artificial stopping criteria are introduced.

Nearest Neighbour heuristic for the TSP:

- always choose at the current city the closest unvisited city
  - choose an arbitrary initial city $\pi(1)$
  - at the ith step choose city $\pi(i+1)$ to be the city j that minimises $\{d(\pi(i), j)\}; j \neq \pi(k), 1 \leq k \leq i$
- running time $\mathcal{O}(n^2)$
- worst case performance $NN(x)/OPT(x) \leq 0.5(\lceil \log_2 n \rceil + 1)$
- other construction heuristics for TSP are available

Hoos / Stützle Stochastic Search Algorithms

The goal of an agent acting in Markov Decision Process is to find *a policy function* $\pi(s)$ that maps states into actions. Solving MDP means finding the *optimal policy* that maximizes the expected cumulative discounted sum of rewards:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)], \tag{1}$$

Mazyavkina, Nina, et al. "Reinforcement learning for combinatorial optimization: A survey." Computers & Operations Research (2021): 105400.

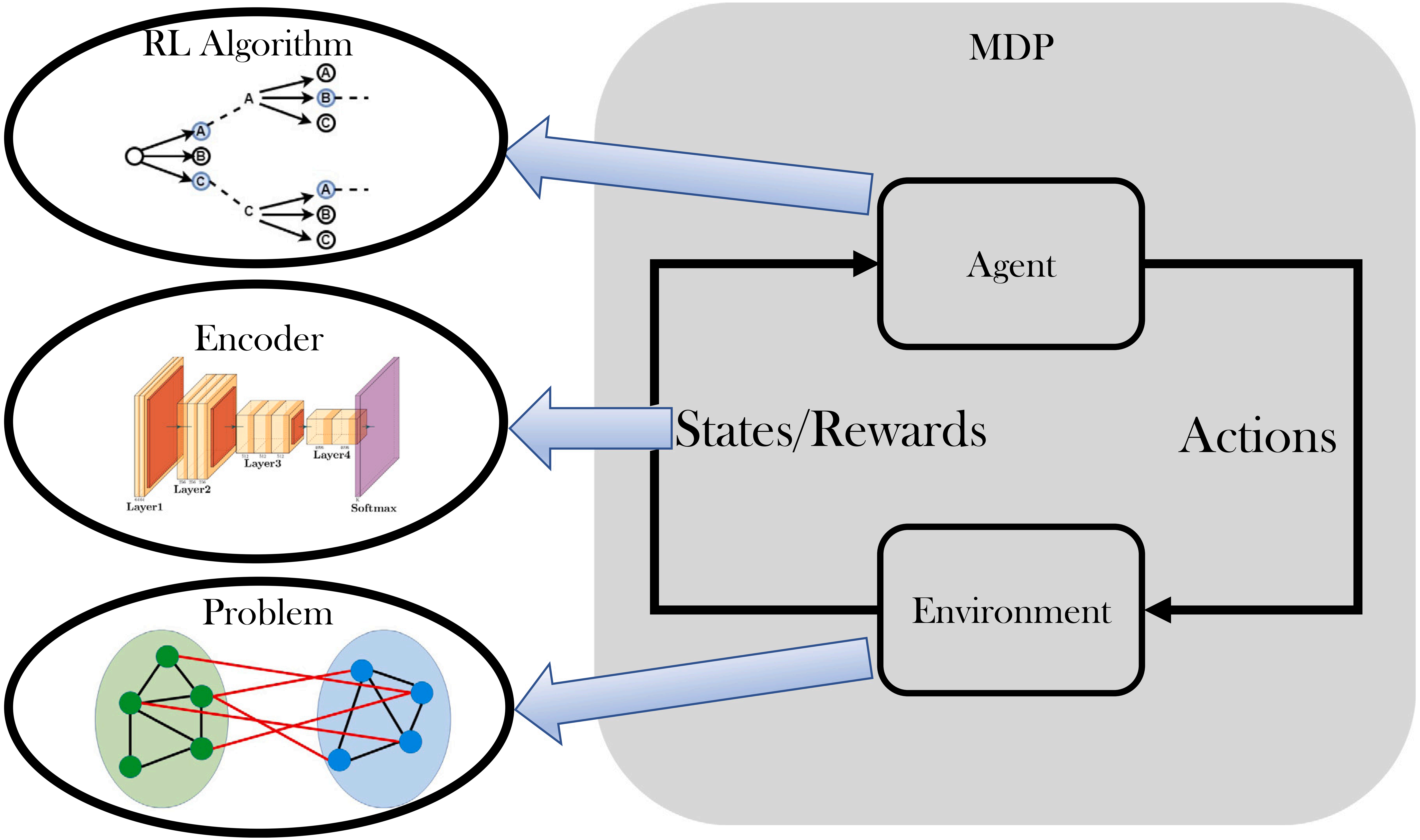
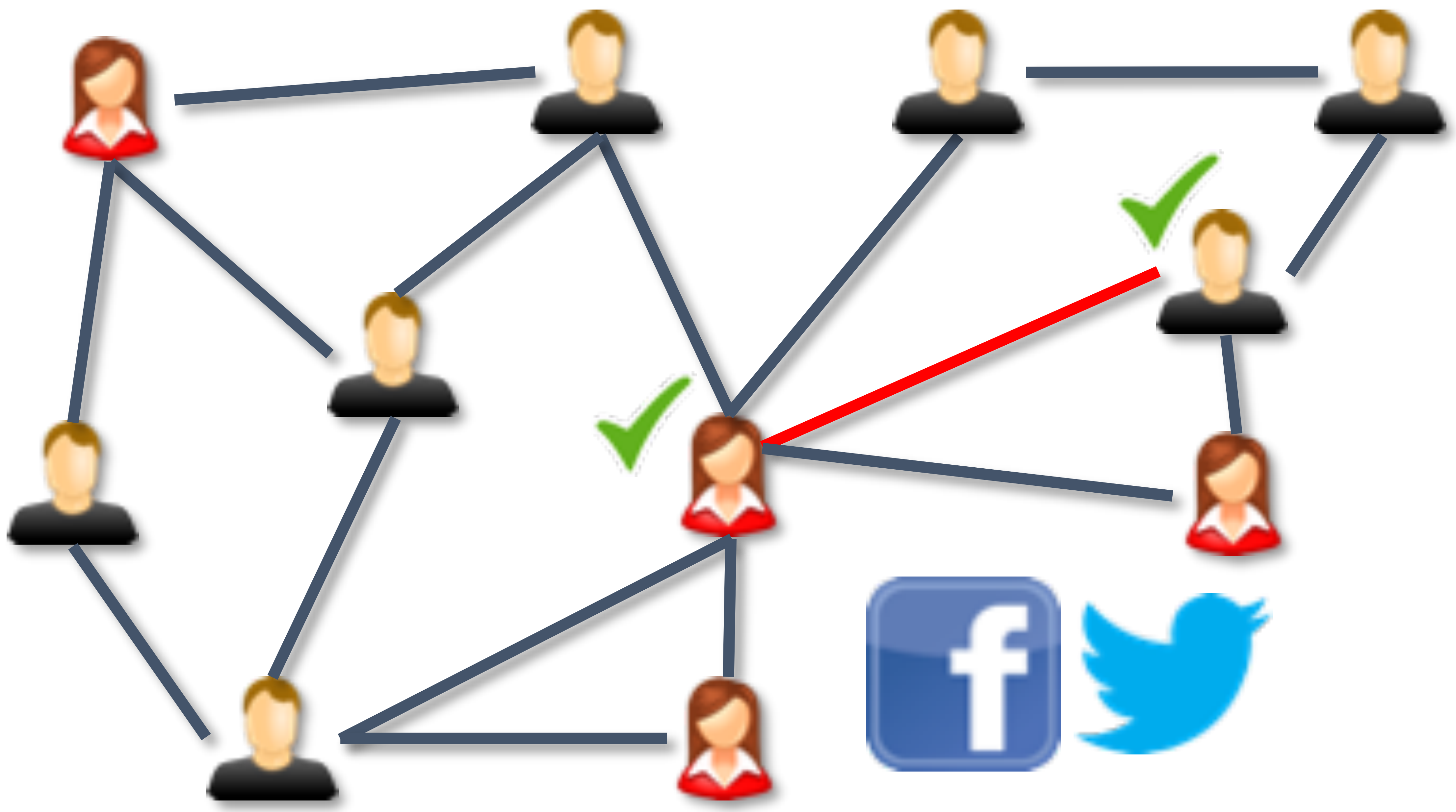


Fig. 1. Solving a CO problem with the RL approach requires formulating MDP. The environment is defined by a particular instance of CO problem (e.g. Max-Cut problem). States are encoded with a neural network model (e.g. every node has a vector representation encoded by a graph neural network). The agent is driven by an RL algorithm (e.g. Monte-Carlo Tree Search) and makes decisions that move the environment to the next state (e.g. removing a vertex from a solution set).

Greedy Graph Optimization

Minimum Vertex Cover
Find smallest vertex subset such that each edge is covered

2-Approximation:
Greedily add vertices of edge with **max degree sum**



Learning Combinatorial Optimization Algorithms over Graphs

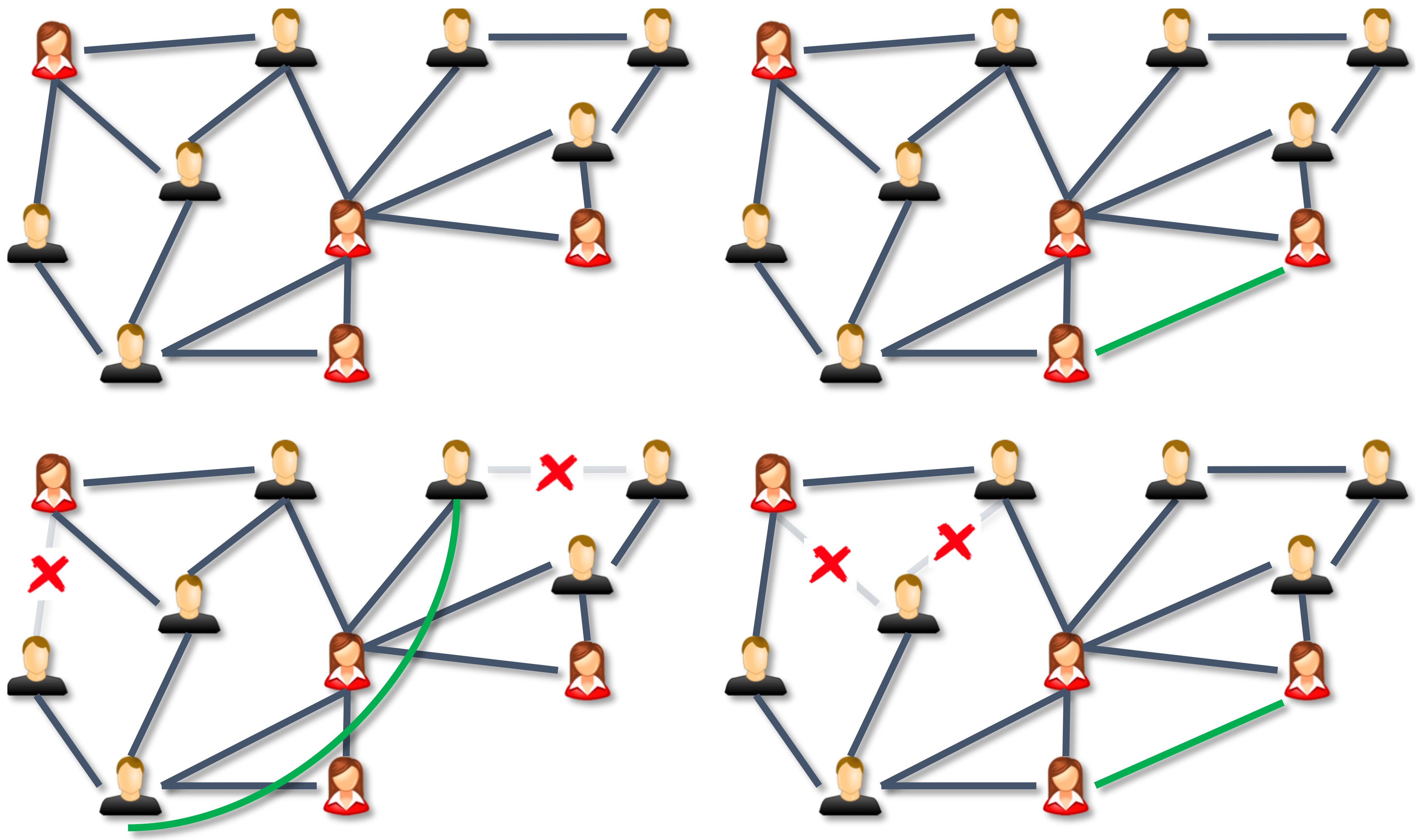
Hanjun Dai[†*], **Elias B. Khalil**[†*], **Yuyu Zhang**[†], **Bistra Dilkina**[†], **Le Song**[†§]
[†] College of Computing, Georgia Institute of Technology
[§] Ant Financial
{hanjun.dai, elias.khalil, yuyu.zhang, bdilkina, lsong}@cc.gatech.edu

NeurIPS 2017

Problem Statement

Dai & Khalil et al., Learning Combinatorial Optimization Algorithms over Graphs. NeurIPS 2017.

Given a **graph optimization problem G** and a **distribution $\mathcal{D}$** of problem instances, can we **learn better greedy heuristics** that generalize to unseen instances from $\mathcal{D}$?



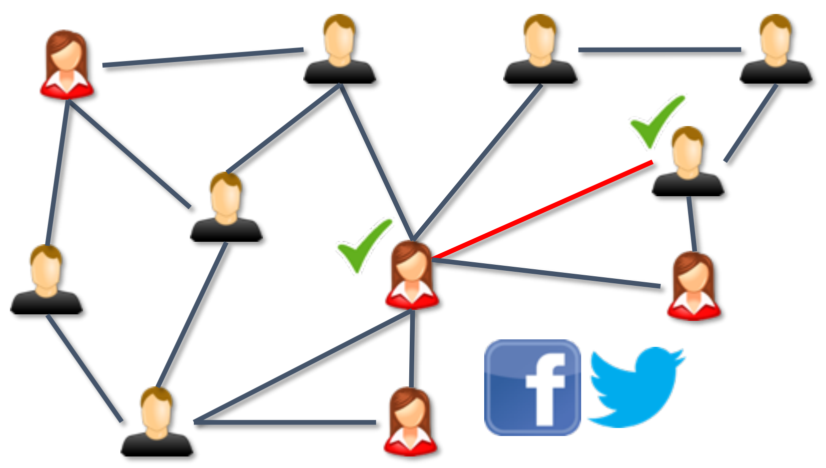
$\sim \mathcal{D}$

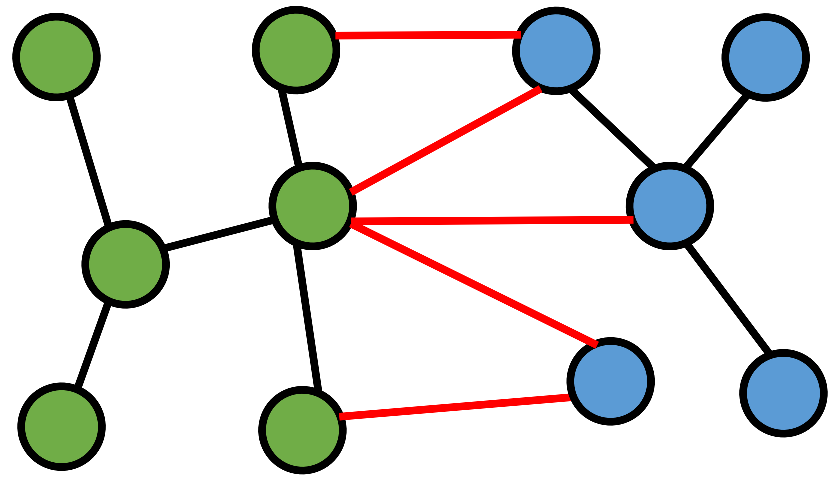
Learning Greedy Heuristics

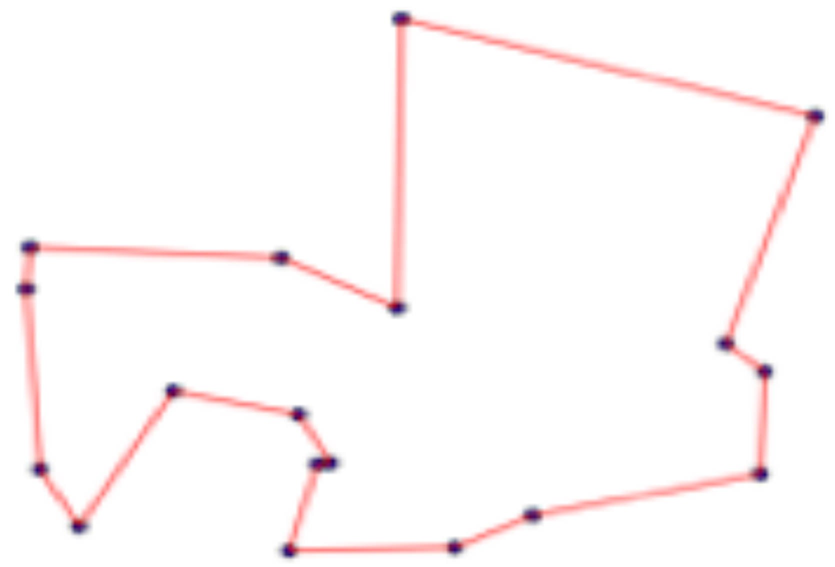
Given: graph problem, family of graphs

Learn: a **scoring function** to **guide** a **greedy** algorithm

Problem	Minimum Vertex Cover	Maximum Cut	Traveling Salesman Problem
Domain	Social network snapshots	Spin glass models	Package delivery
Greedy operation	Insert nodes into cover	Insert nodes into subset	Insert nodes into sub-tour





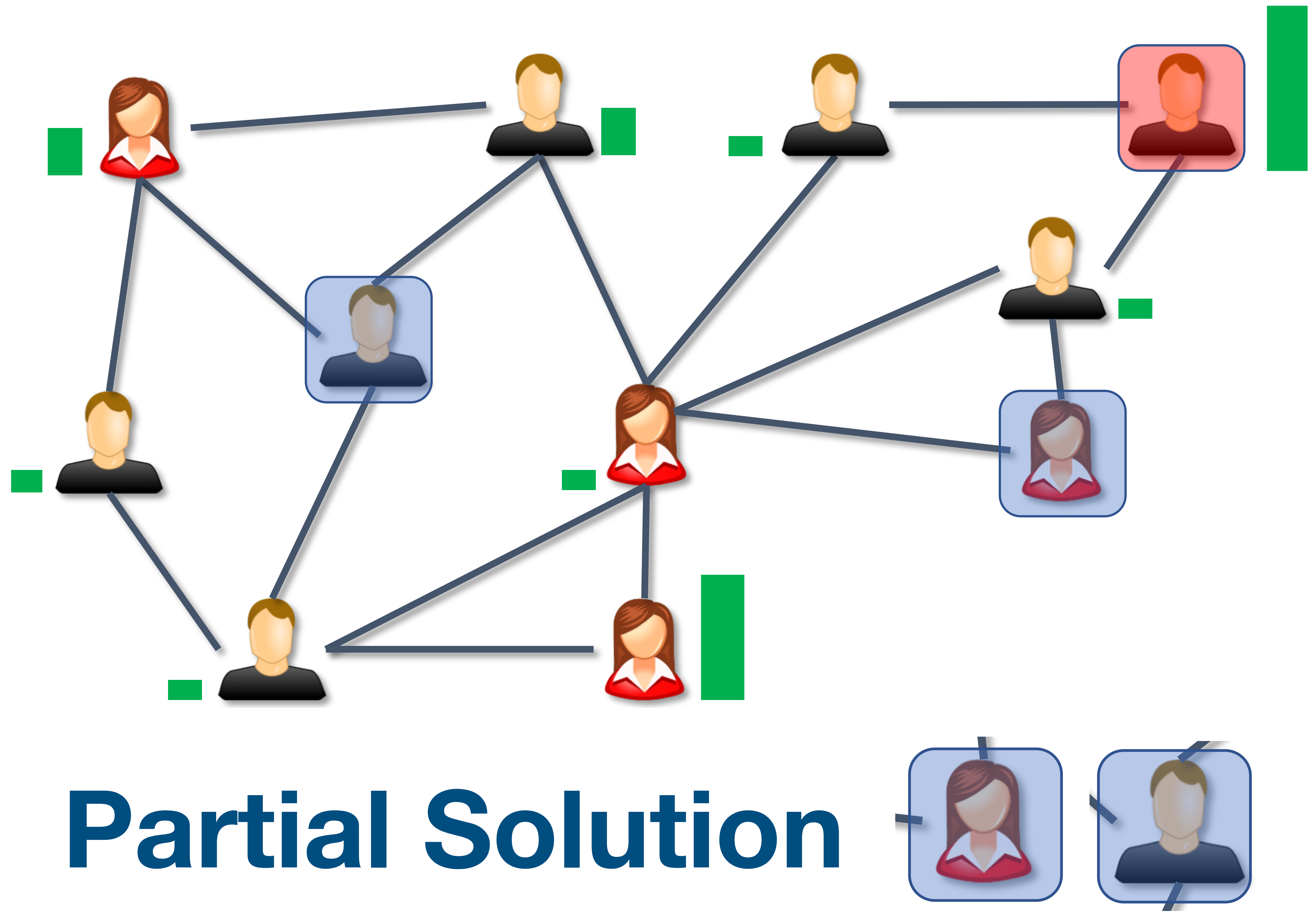


Reinforcement Learning

Greedy Algorithm		Reinforcement Learning
Partial solution	≡	State
Scoring function	≡	Q-function
Select best node	≡	Greedy Policy

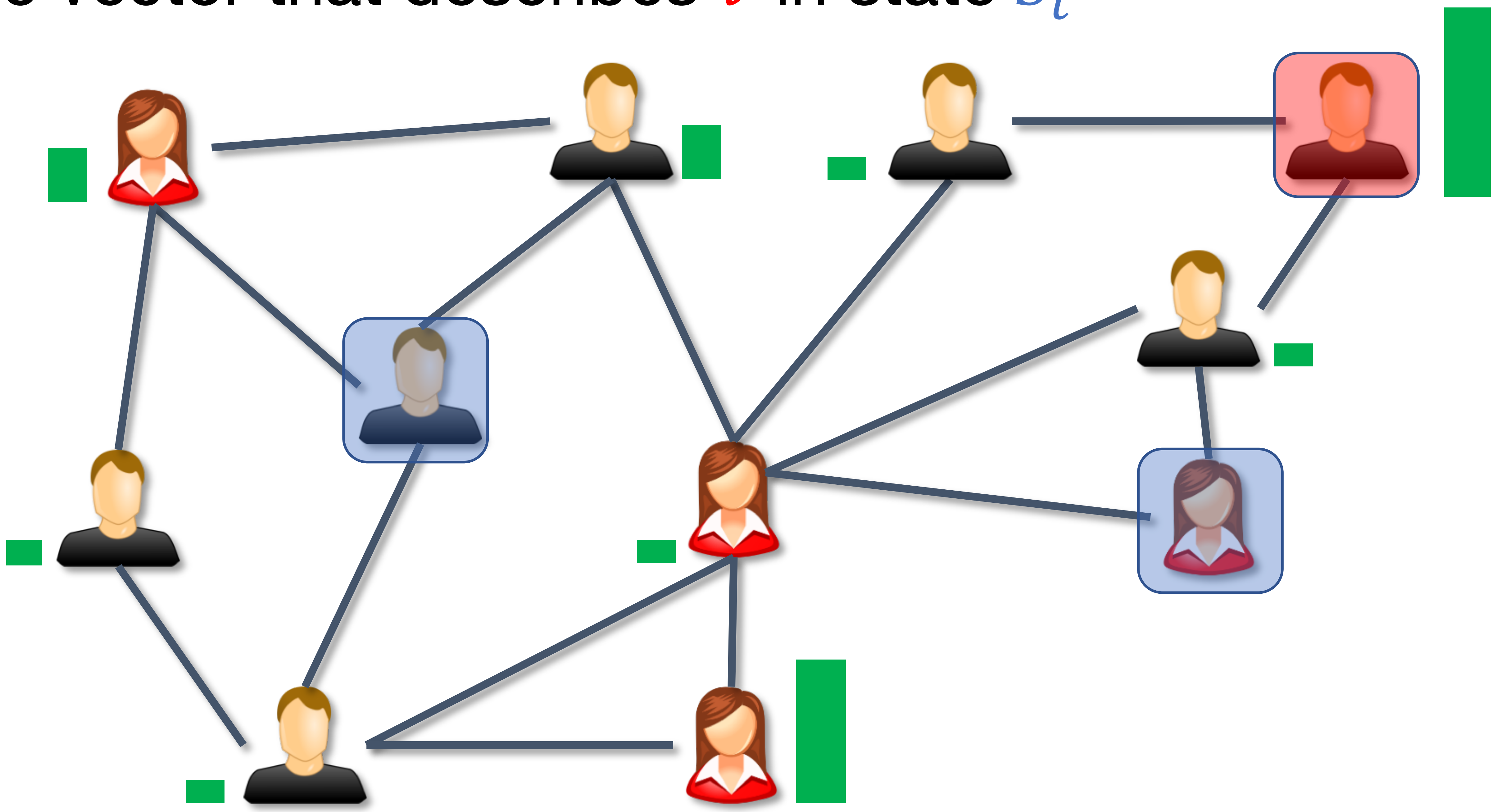
Repeat until all edges are covered:

1. Compute node scores
2. Select best node w.r.t. score
3. Add best node to partial sol.



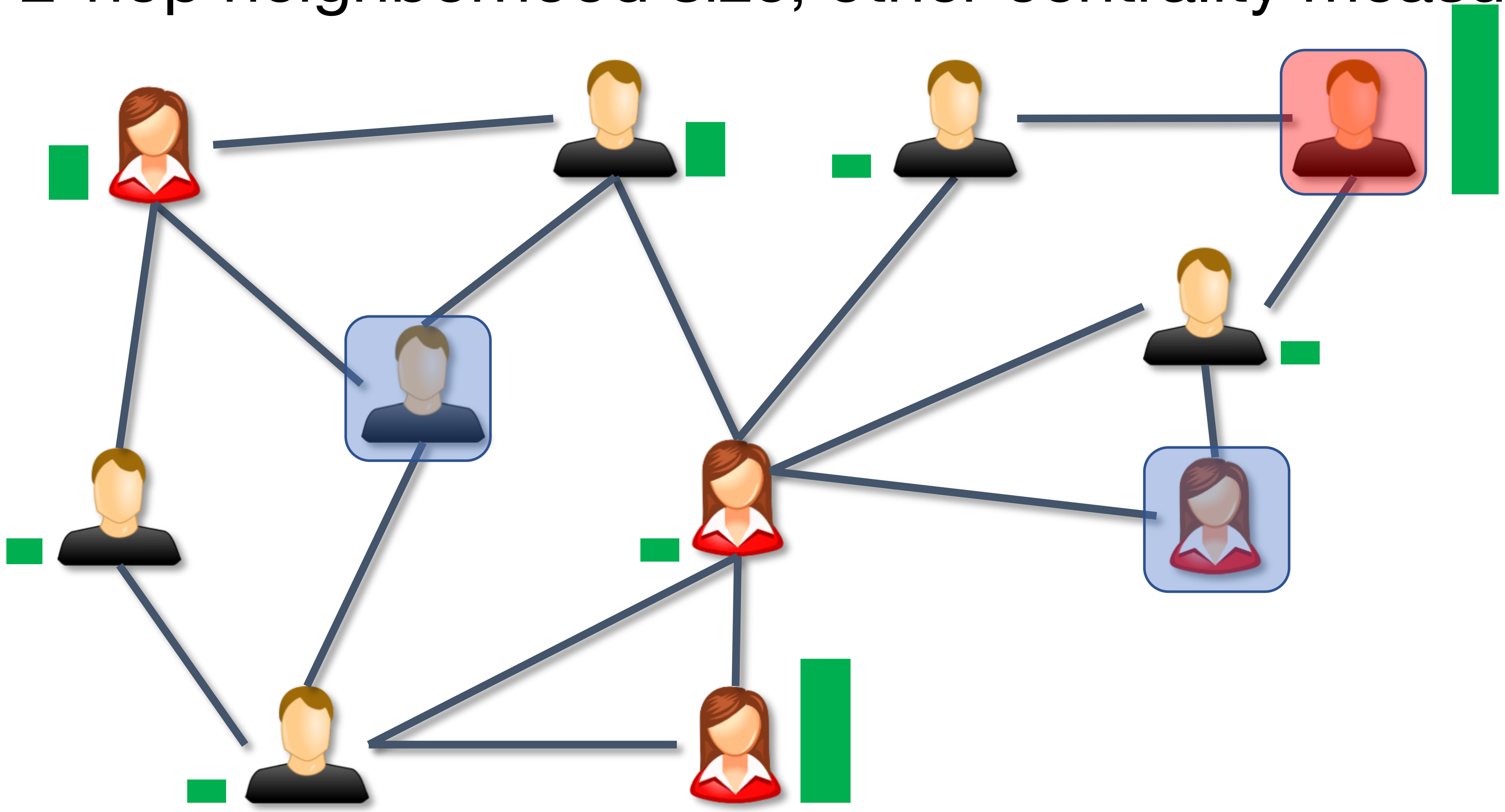
Representing Nodes

- **Action value function:** $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex v in state S_t
- **Representation of v**
  - A feature vector that describes v in state S_t



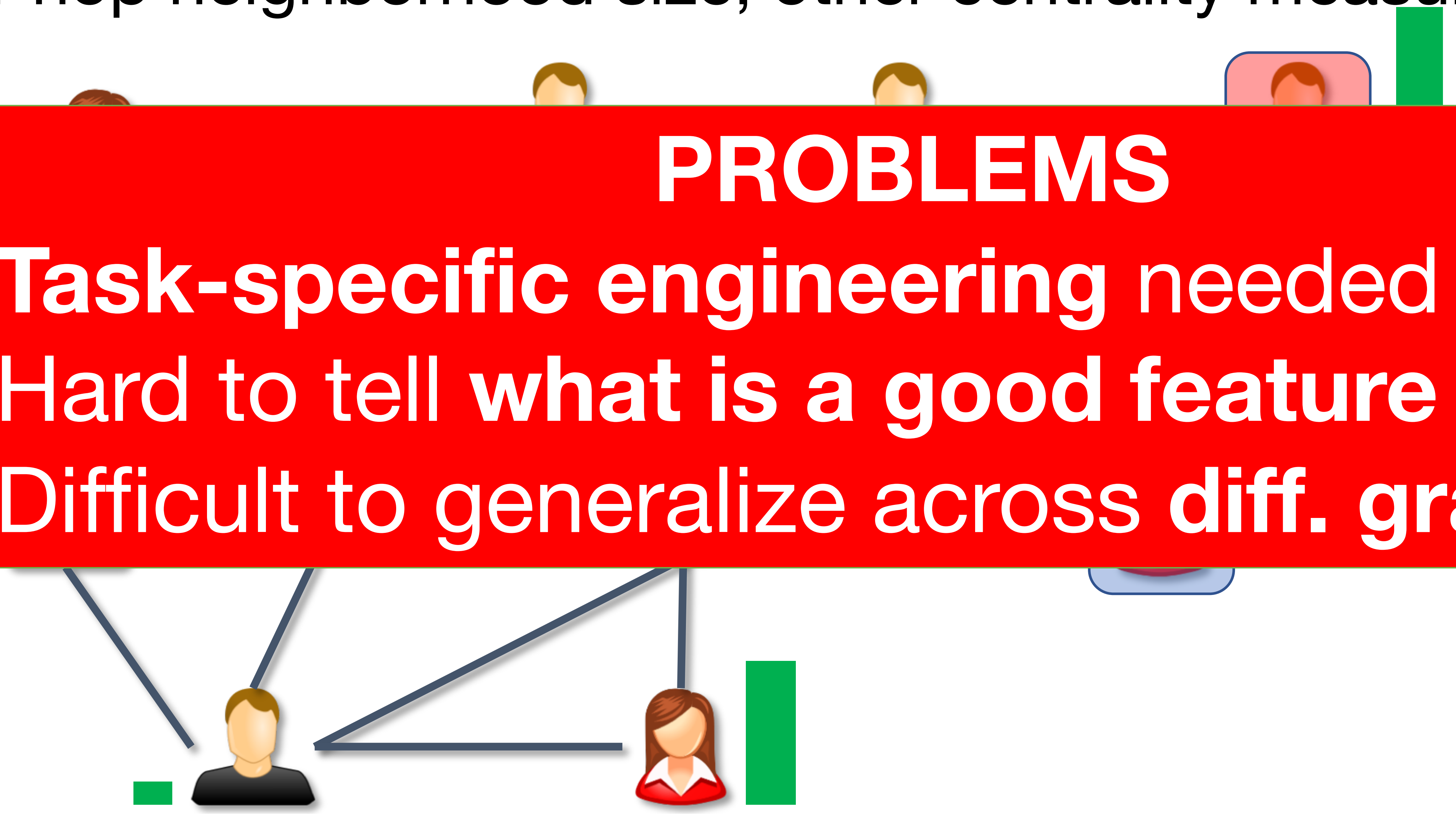
Representing Nodes

- **Action value function:** $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex v in state S_t
- **Representation of v: Feature engineering**
  - Degree, 2-hop neighborhood size, other centrality measures…



Representing Nodes

- **Action value function:** $\hat{Q}(S_t, v; \Theta)$
  - Estimate of goodness of vertex v in state S_t
- **Representation of v: Feature engineering**
  - Degree, 2-hop neighborhood size, other centrality measures…



PROBLEMS

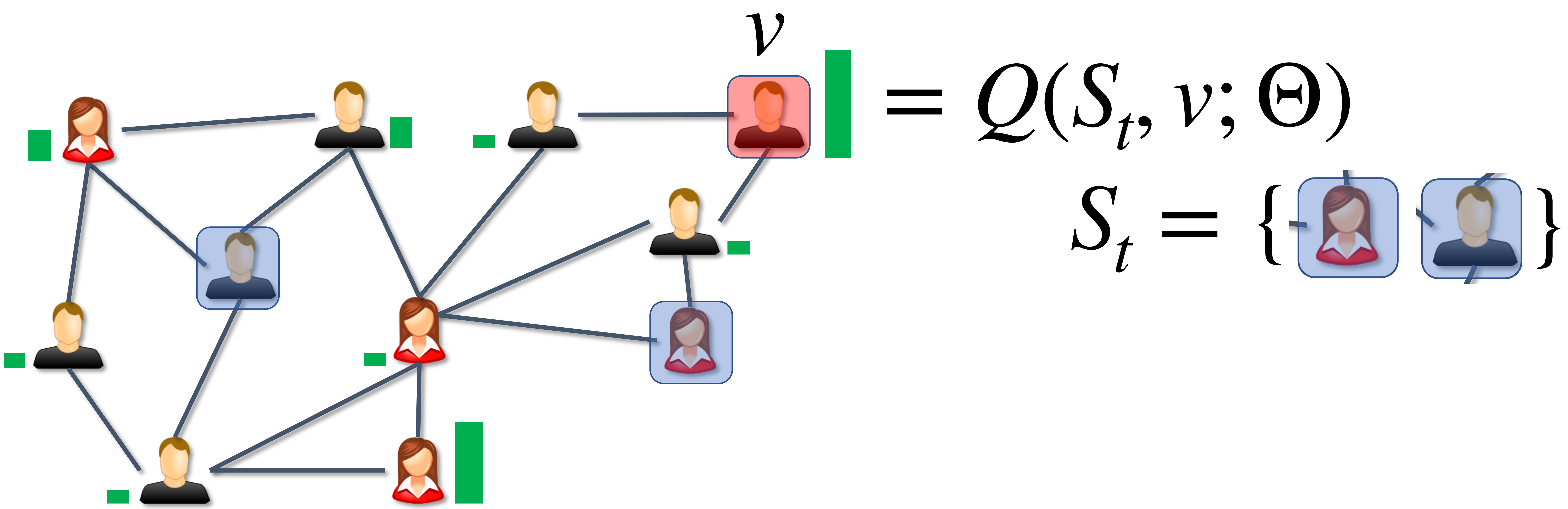
1- **Task-specific engineering** needed

2- Hard to tell **what is a good feature**

3- Difficult to generalize across **diff. graph sizes**

Learning Node Features

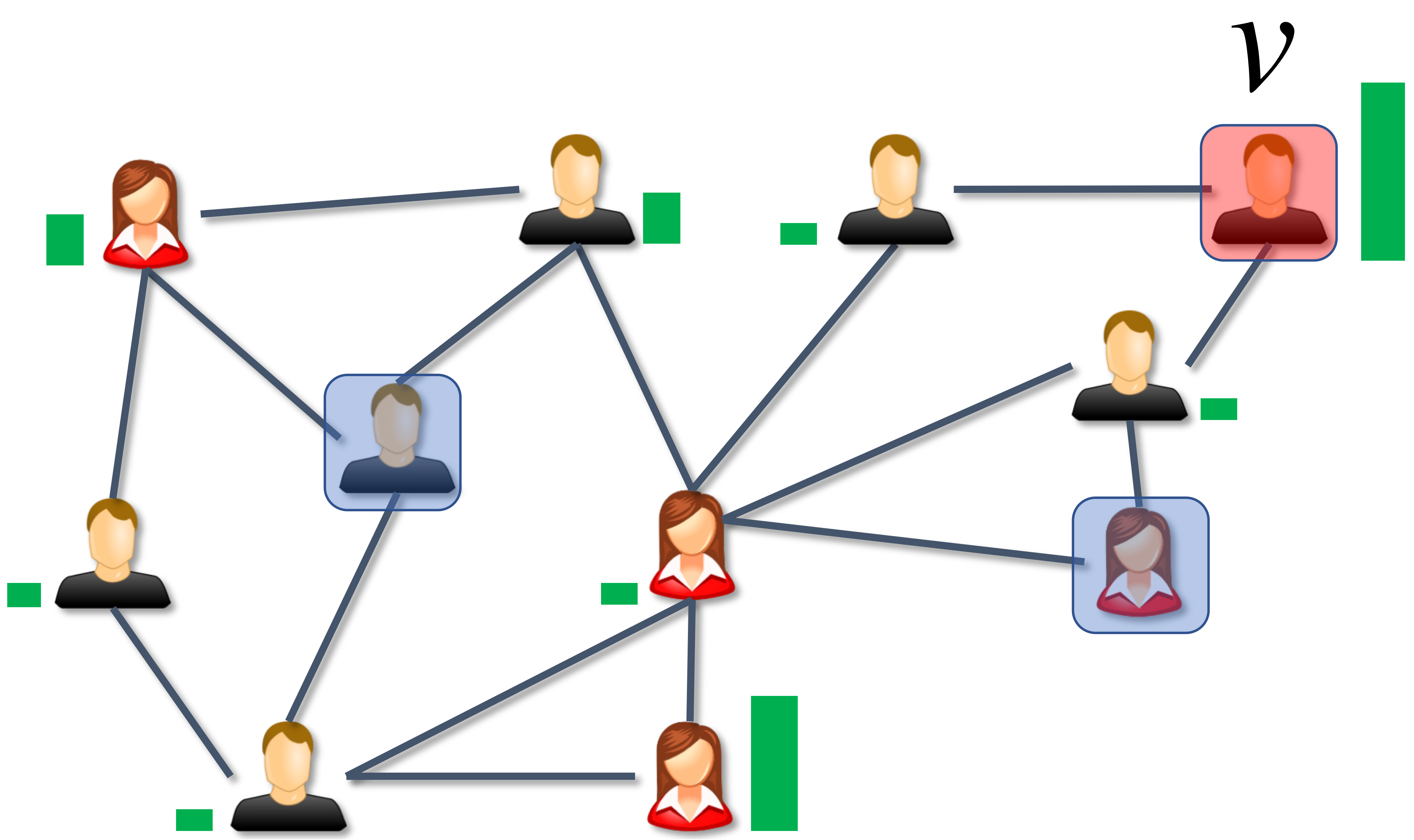
Scoring Function: Need to represent node with a **feature vector** first



Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!



$$= Q(S_t, v; \Theta)$$

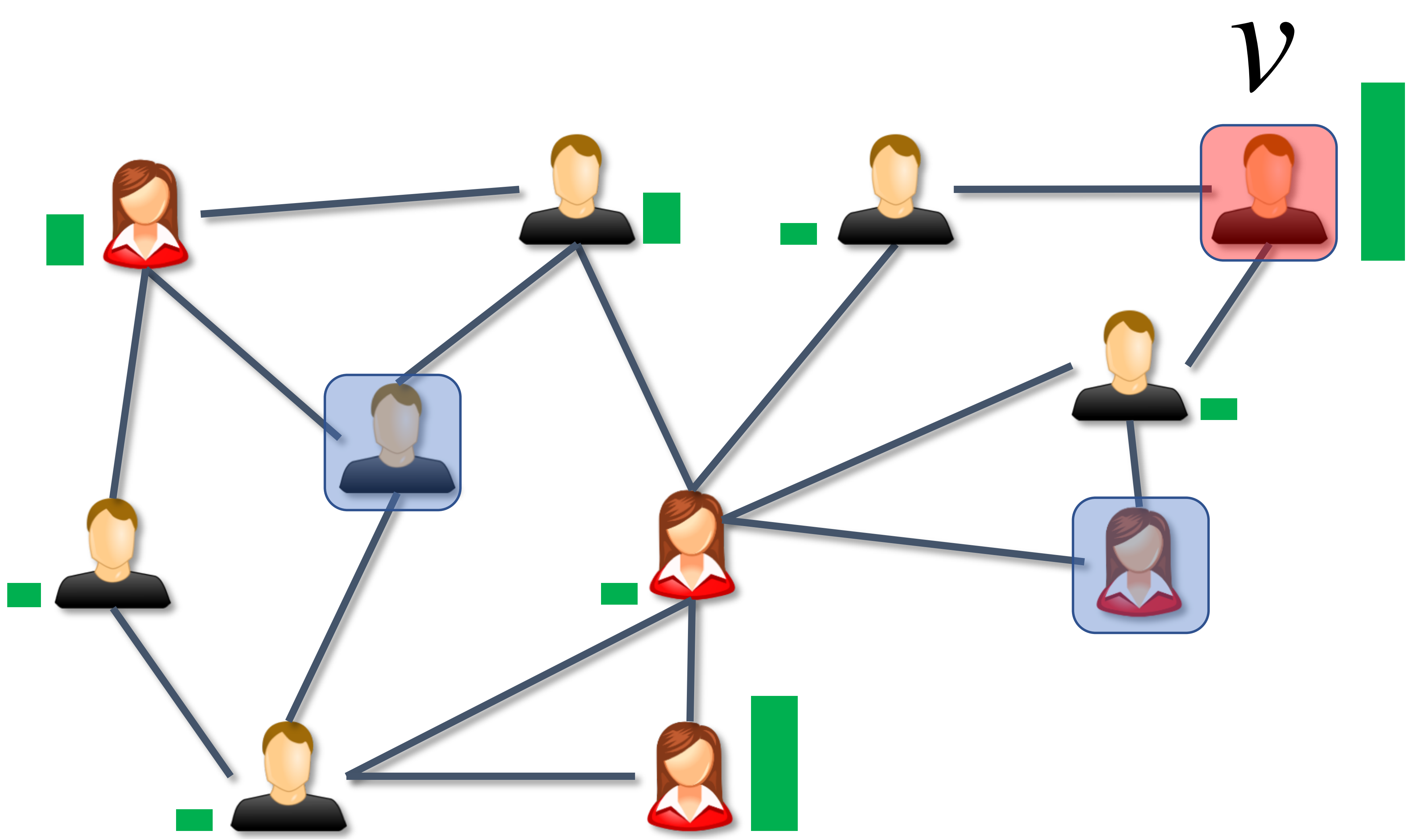
$S_t = \{$  $\}$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!

Solution: Parametrize a **Graph Neural Network** with parameters Θ



$$= Q(S_t, v; \Theta)$$

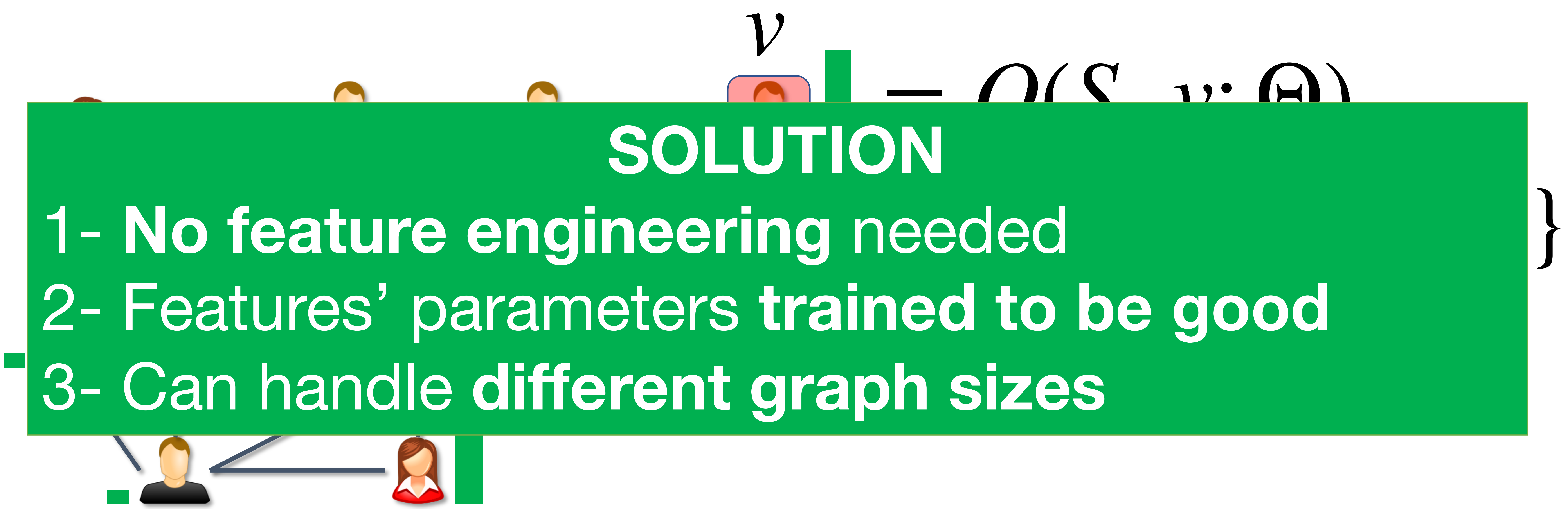
$S_t = \{$  $\}$

Learning Node Features

Scoring Function: Need to represent node with a **feature vector** first

Problem: Not clear what good node features are!

Solution: Parametrize a **Graph Neural Network** with parameters Θ



SOLUTION

1- **No feature engineering** needed

2- Features' parameters **trained to be good**

3- Can handle **different graph sizes**

Reinforcement Learning Algorithm

Algorithm 1 **Q-learning for the Greedy Algorithm**

Initialize experience replay memory $\mathcal{M}$ to capacity N
for episode $e = 1$ **to** L **do**
  Draw graph G from distribution $\mathbb{D}$
  Initialize the state to empty $S_1 = ()$
  for step $t = 1$ **to** T **do**
    $v_t = \begin{cases} \text{random node } v \in \overline{S}_t, & \text{w.p. } \epsilon \\ \text{argmax}_{v \in \overline{S}_t} \widehat{Q}(h(S_t), v; \Theta), & \text{otherwise} \end{cases}$
    Add v_t to partial solution: $S_{t+1} := (S_t, v_t)$
    if $t \geq n$ **then**
      Add tuple $(S_{t-n}, v_{t-n}, R_{t-n,t}, S_t)$ to $\mathcal{M}$
      Sample random batch from $B \overset{iid.}{\sim} \mathcal{M}$
      Update Θ by SGD over (6) for B
    end if
  end for
end for
return Θ

Θ: model parameters
Depend on vertex features

Sample graph instance

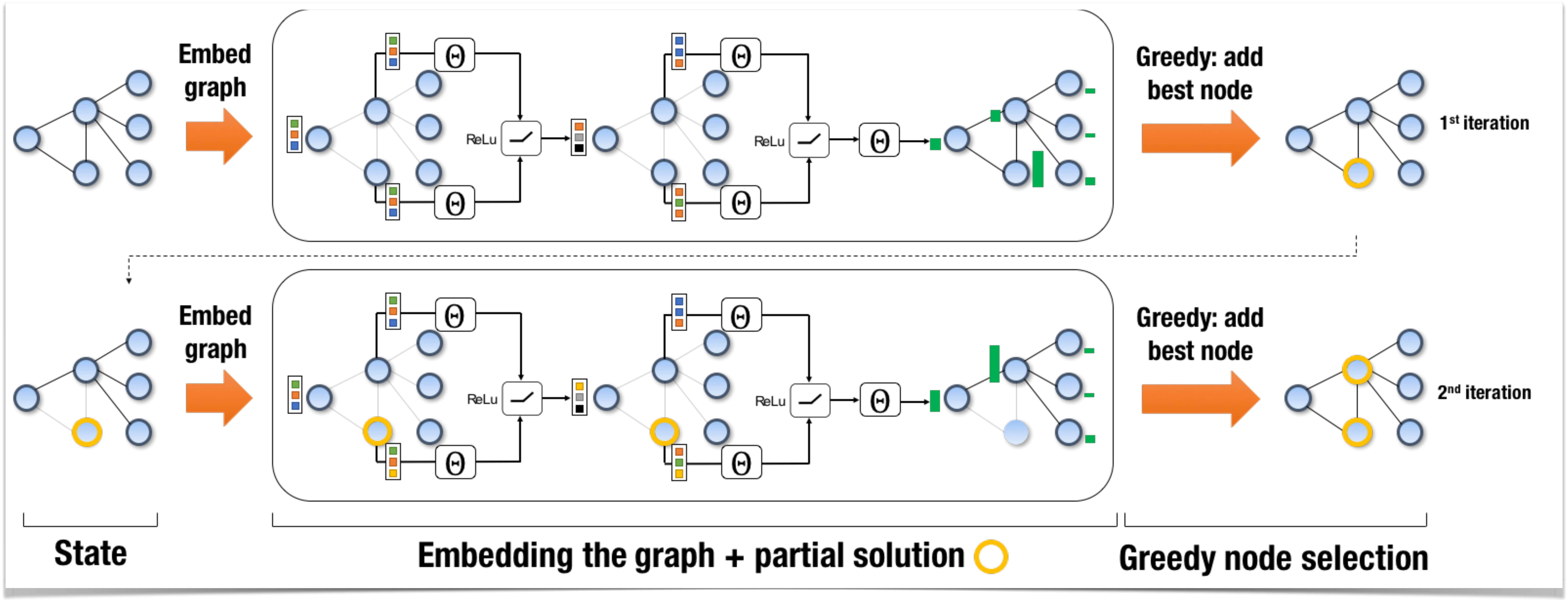
Explore or
Exploit according to current policy

Update state

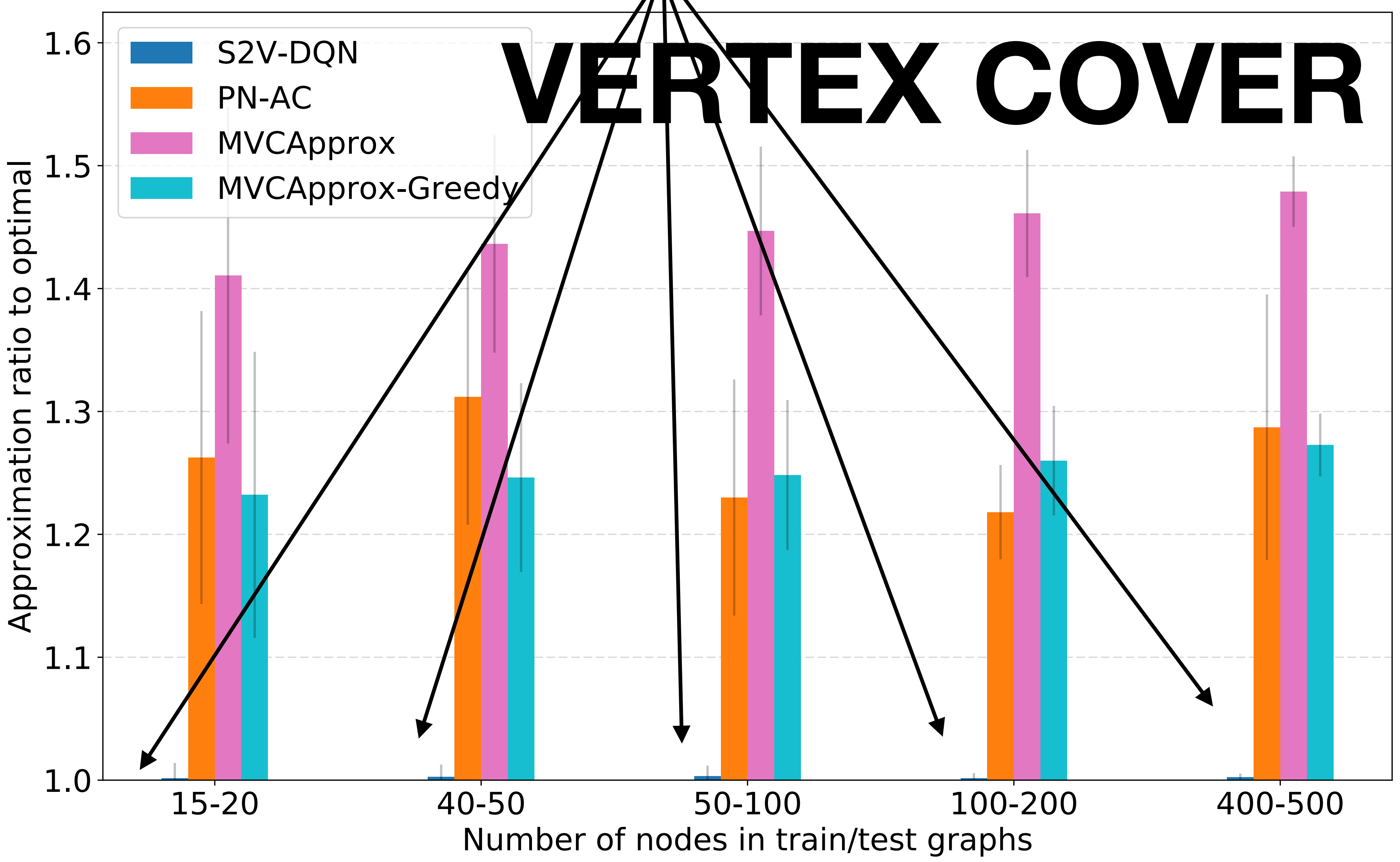
Optimize model parameters

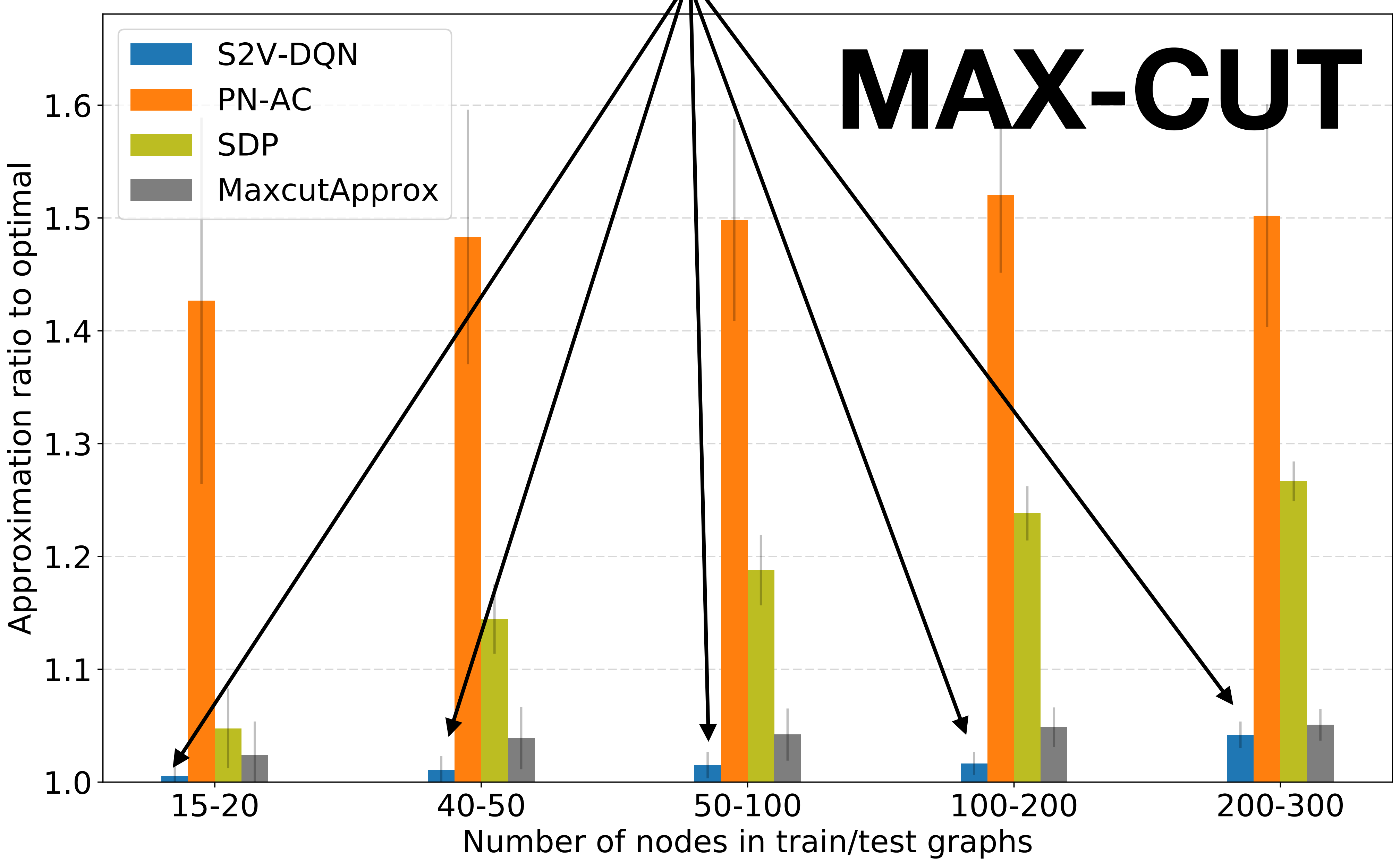
$$(y - \widehat{Q}(h(S_t), v_t; \Theta))^2$$
$$y = \gamma \max_{v'} \widehat{Q}(h(S_{t+1}), v'; \Theta) + r(S_t, v_t)$$

Overall Framework

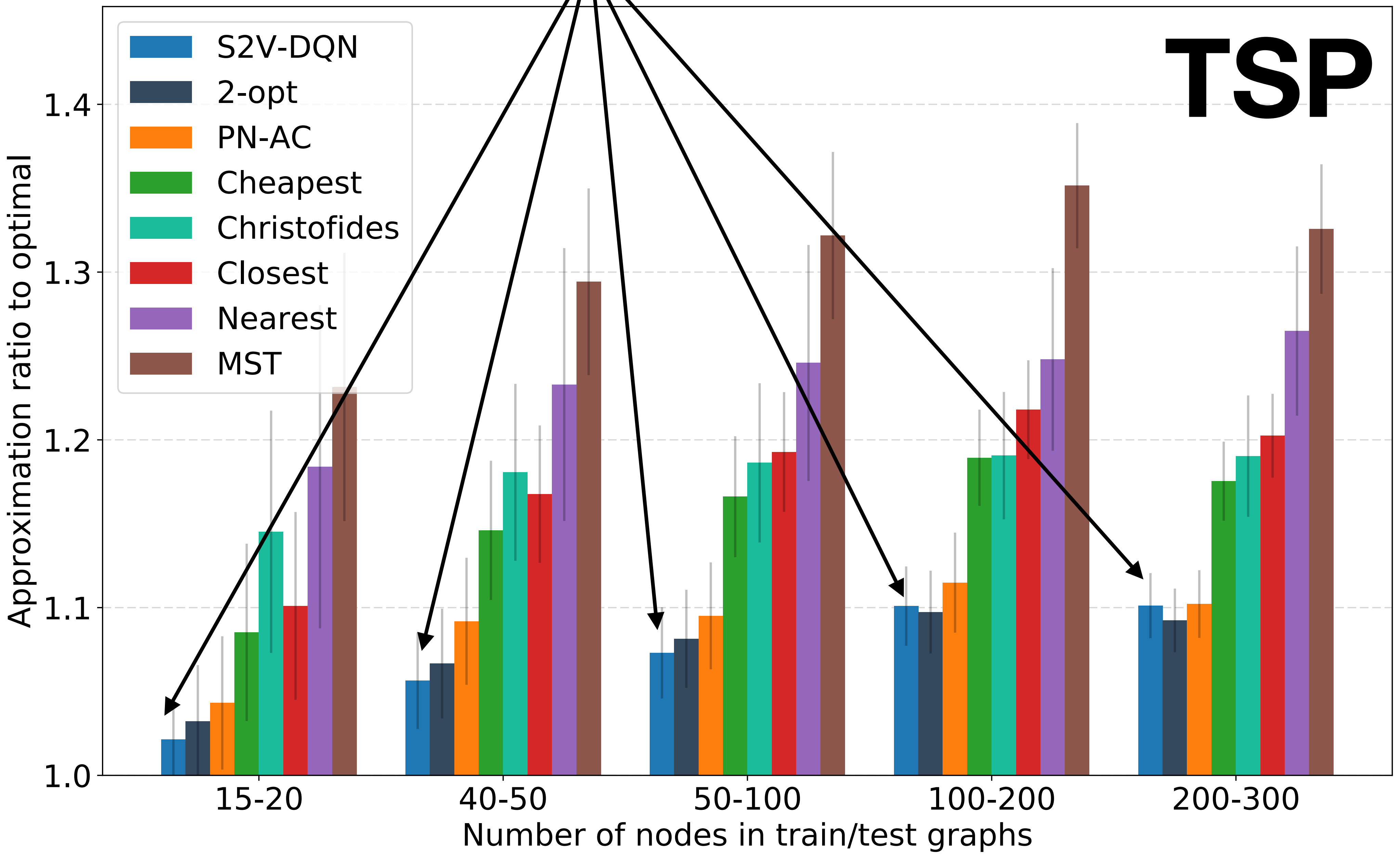


Learning Greedy in Practice





Approximation Ratio

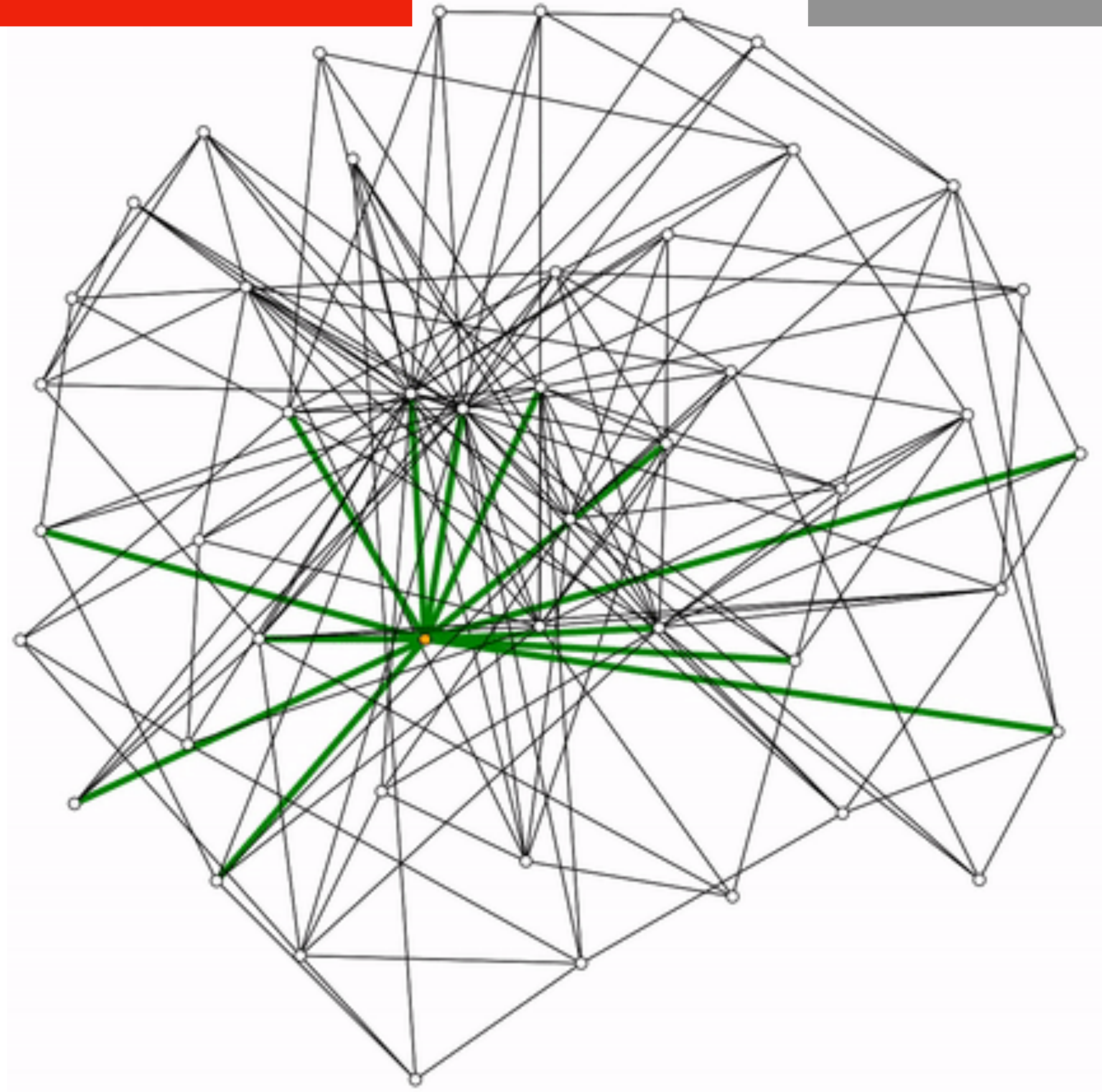


Code: https://github.com/Hanjun-Dai/graph_comb_opt

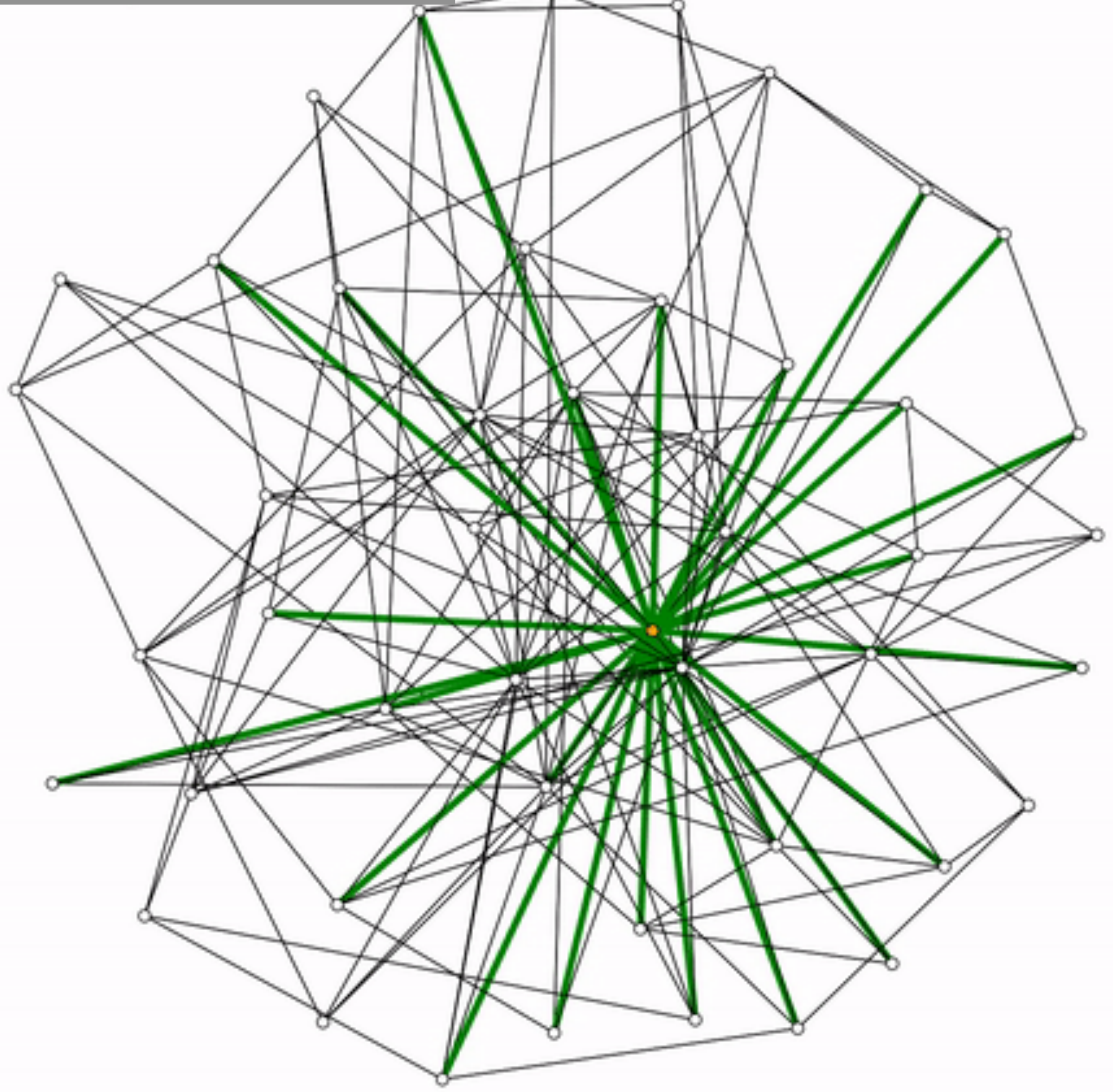
Data-Driven Algorithm Design

automatically **discovers** **novel** search **strategies**

Learned Heuristic



Classical Heuristic



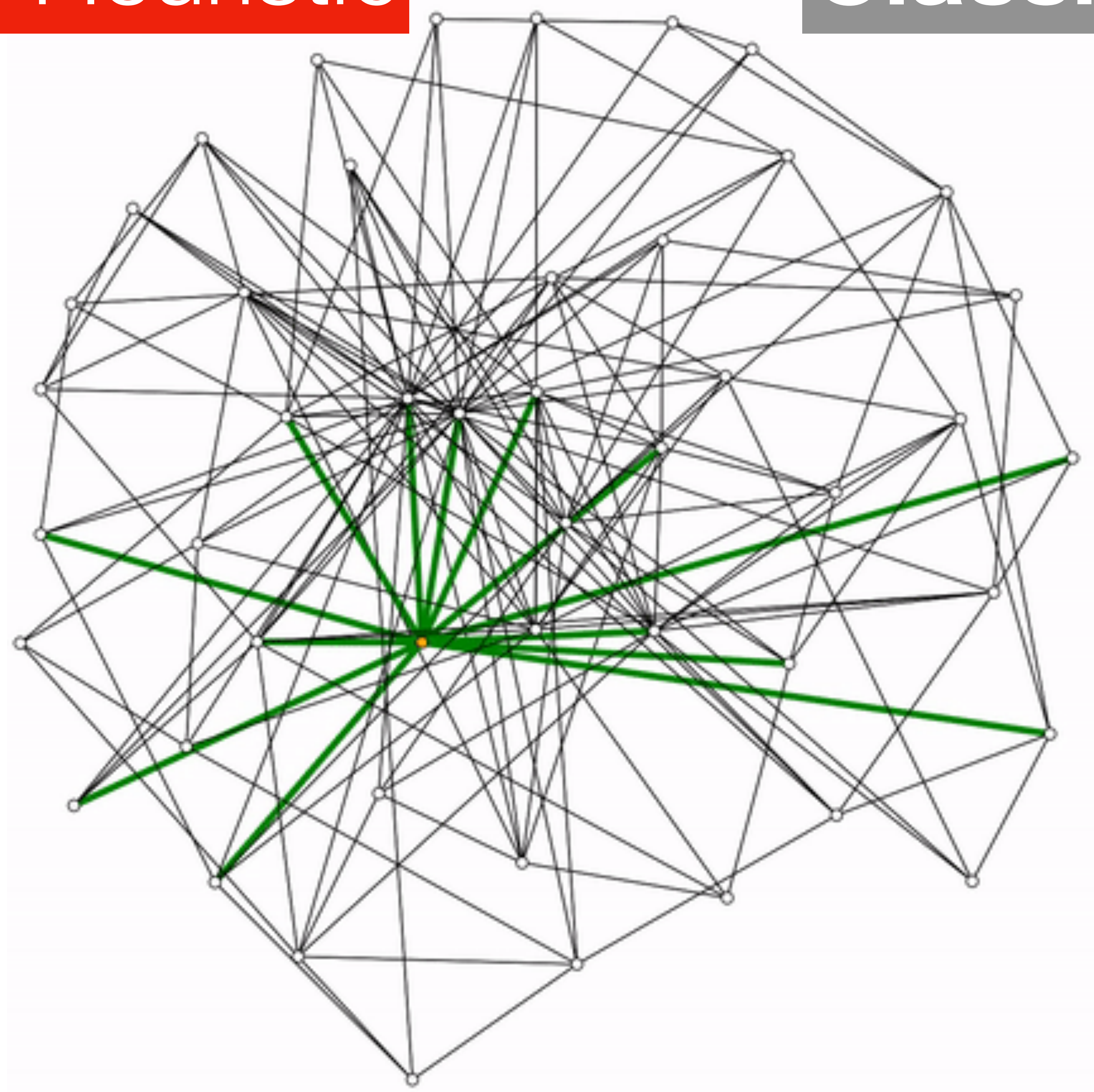
Minimum Vertex Cover

Find **smallest vertex subset** such that each **edge is covered**

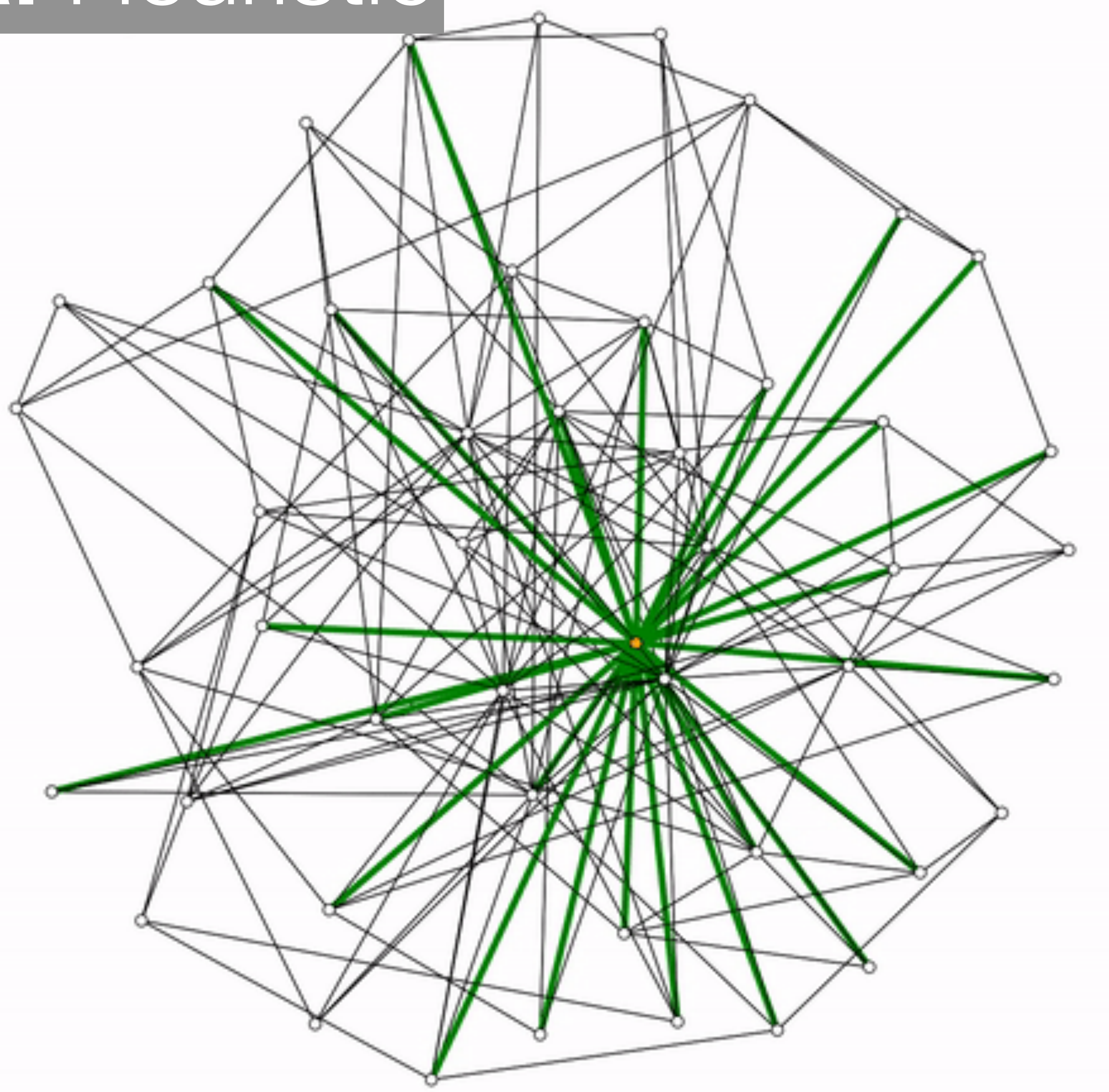
Data-Driven Algorithm Design

automatically **discovers** **novel** search **strategies**

Learned Heuristic

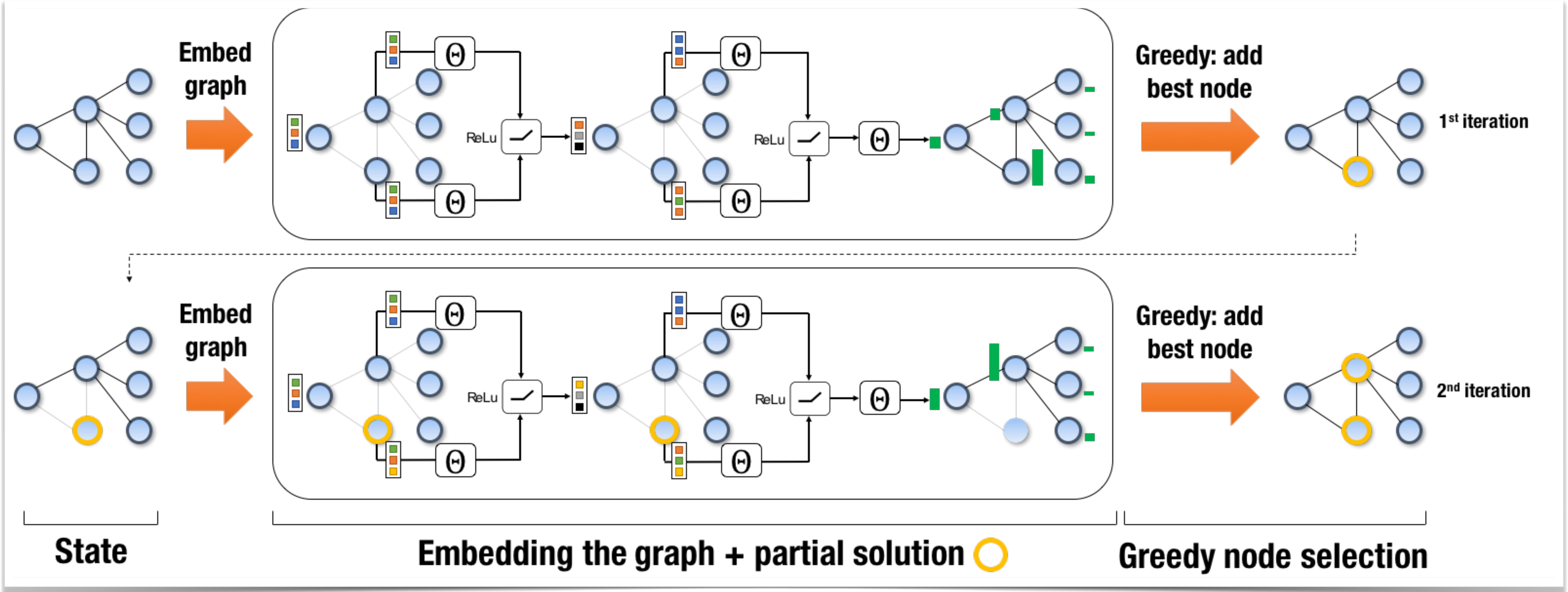


Classical Heuristic



Minimum Vertex Cover

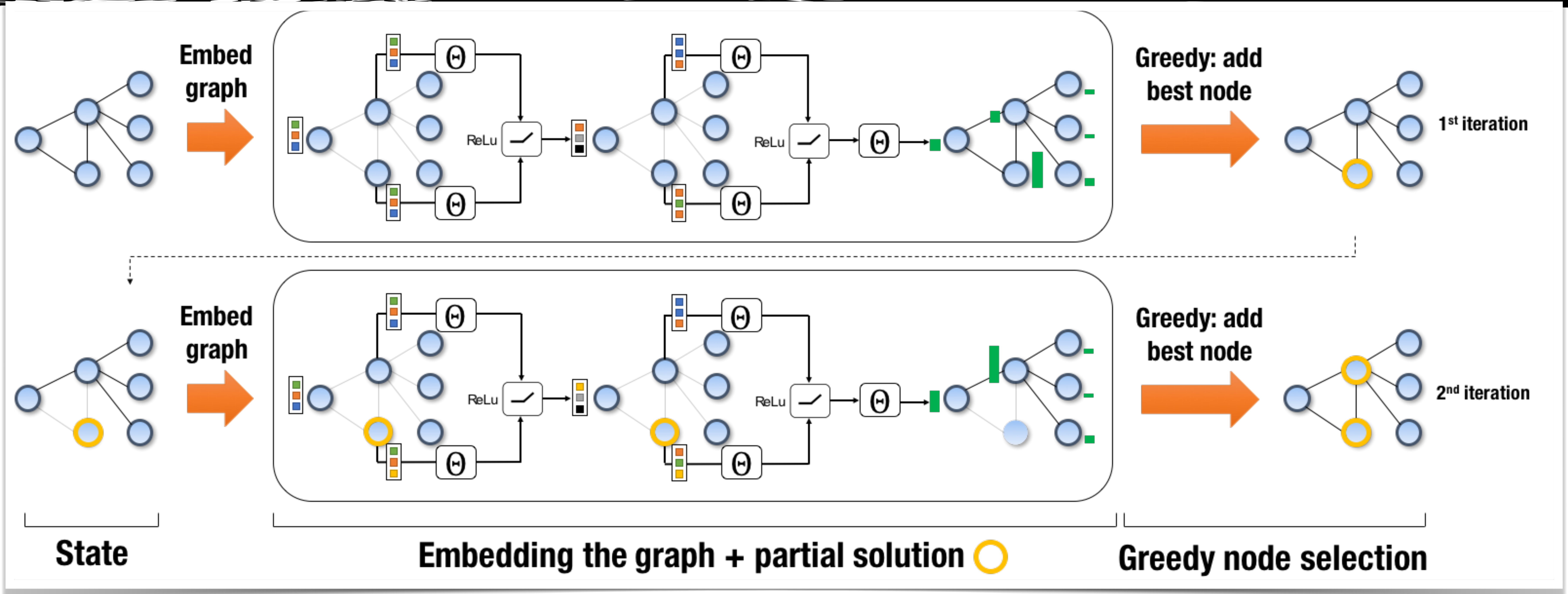
Find **smallest vertex subset** such that each **edge is covered**



Embed graph
ReLu
Greedy: add best node
1st iteration
Embed graph
ReLu
Greedy: add best node
2nd iteration
State
Embedding the graph + partial solution
Greedy node selection

Takeaways

- **Reinforcement Learning tailors greedy** search to your instances
- Learn **features jointly with** greedy **policy**
- **Human priors** encoded via (greedy) **meta-algorithm**
- **Interesting, novel strategies emerge**



Learning Heuristics: Recent progress

Published as a conference paper at ICLR 2019

ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS!

Wouter Kool
University of Amsterdam
ORTEC
w.w.m.kool@uva.nl

Herke van Hoof
University of Amsterdam
h.c.vanhoof@uva.nl

Max Welling
University of Amsterdam
CIFAR
m.welling@uva.nl

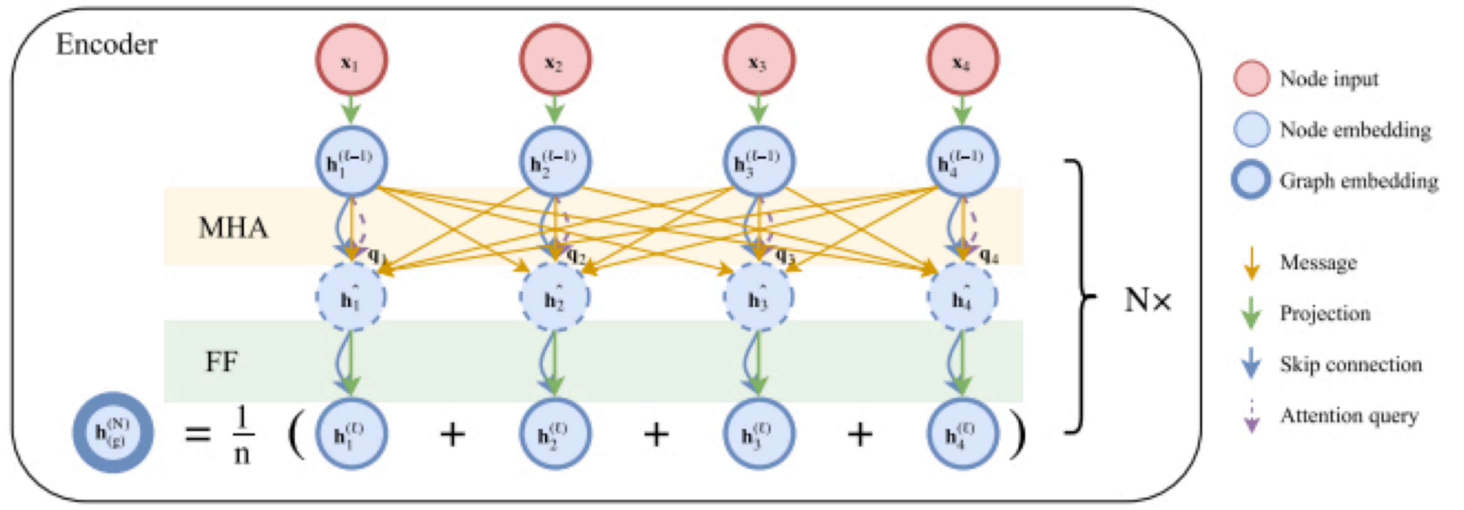


Figure 1: Attention based encoder. Input nodes are embedded and processed by N sequential layers, each consisting of a multi-head attention (MHA) and node-wise feed-forward (FF) sub-layer. The graph embedding is computed as the mean of node embeddings. Best viewed in color.

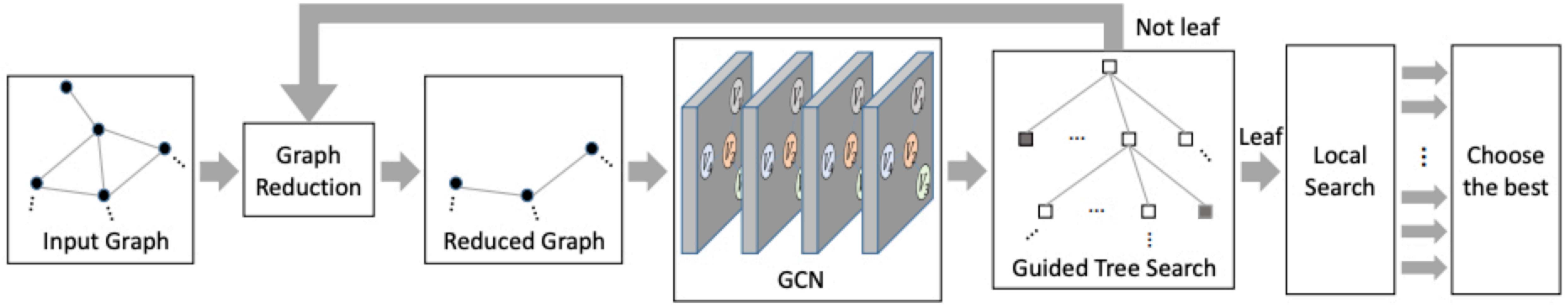
Learning Heuristics: Recent progress

Reinforcement Learning for Solving the Vehicle Routing Problem

Mohammadreza Nazari Afshin Oroojlooy Martin Takáč Lawrence V. Snyder
Department of Industrial and Systems Engineering
Lehigh University, Bethlehem, PA 18015
{mon314,afo214,takac,lvs2}@lehigh.edu

Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search

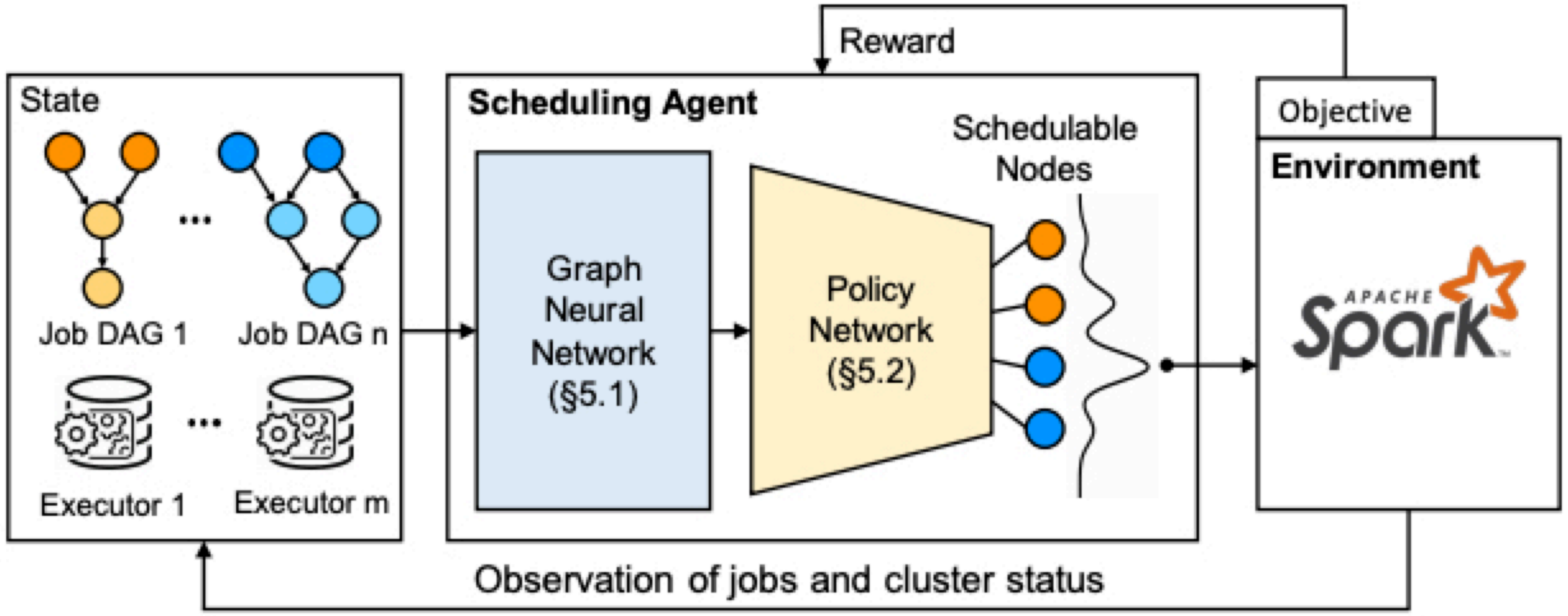
Zhuwen Li Intel Labs
Qifeng Chen HKUST
Vladlen Koltun Intel Labs



Learning Heuristics: Recent progress

Learning Scheduling Algorithms for Data Processing Clusters

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng*, Mohammad Alizadeh
MIT Computer Science and Artificial Intelligence Laboratory *Tsinghua University



Learning Heuristics: Recent progress

Learning to Perform Local Rewriting for Combinatorial Optimization

Xinyun Chen *
UC Berkeley
xinyun.chen@berkeley.edu

Yuandong Tian
Facebook AI Research
yuandong@fb.com