Learning Combinatorial Optimization Algorithms over Graphs

Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, Le Song

NeurIPS'17

Overview

Goal: use RL to learn new *greedy strategies* for graph problems

Feasible solution constructed by successively adding nodes to solution

Input: Graph $G = (V, E)$, weights $w(u, v)$ for $(u, v) \in E$

RL state representation: Graph embedding

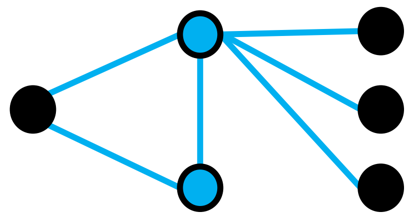
Algorithm training: Fitted Q-learning

Outline

1. **Greedy algorithms**
2. Graph representation
3. RL formulation
4. Q-learning
5. Experiments

Minimum vertex cover

Find smallest vertex subset such that each edge is covered

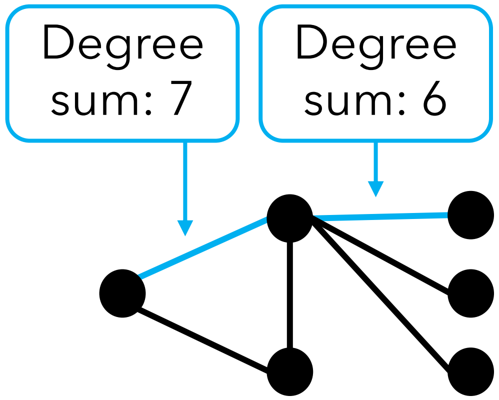


Minimum vertex cover

Find smallest vertex subset such that each edge is covered

2-approximation:

Greedily add vertices of edge with **maximum degree sum**



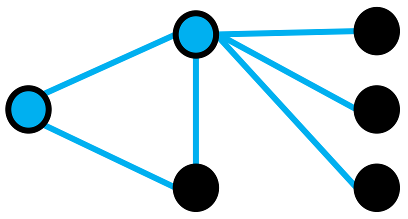
Minimum vertex cover

Find smallest vertex subset such that each edge is covered

2-approximation:

Greedily add vertices of edge with **maximum degree sum**

Scoring function that guides greedy algorithm



Maximum cut

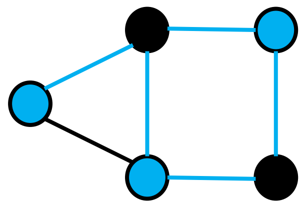
Find partition $(S, V \setminus S)$ of nodes that maximizes

$$\sum_{(u,v)\in C} w(u,v)$$

where $C = \{(u,v) \in E : u \in S, v \notin S\}$

If $w(u,v) = 1$ for all $(u,v) \in E$:

$$\sum_{(u,v)\in C} w(u,v) = 5$$



Maximum cut

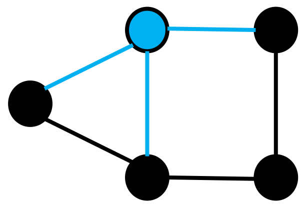
Find partition $(S, V \setminus S)$ of nodes that maximizes

$$\sum_{(u,v)\in C} w(u,v)$$

where $C = \{(u,v) \in E : u \in S, v \notin S\}$

Greedy: move node from one side of cut to the other

Move node that results in the largest improvement in cut weight



Maximum cut

Find partition $(S, V \setminus S)$ of nodes that maximizes

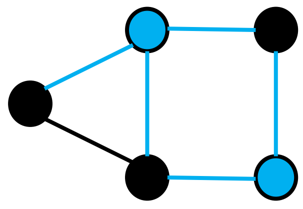
$$\sum_{(u,v)\in C} w(u,v)$$

where $C = \{(u,v) \in E : u \in S, v \notin S\}$

Greedy: move node from one side of cut to the other

Move node that results in the largest improvement in cut weight

Scoring function that guides greedy algorithm



General greedy algorithm formulation

1. **Partial solution** is an ordered list $S = (v_1, v_2, \ldots, v_{|S|}), v_i \in V$
2. ***Helper function*** $h(S)$ maps S to combinatorial structure, eg:
   - **Maxcut:** $h(S)$ returns cut $C = \{(u, v) \in E: u \in S, v \notin S\}$
   - **TSP:** $h(S)$ maintains a partial tour according to order of nodes in S
   - **Min vertex cover:** $h(S)$ does nothing
3. **Quality** of S evaluated by function $c(h(S), G)$, e.g.:
   - **Maxcut:** $c(h(S), G) = \sum_{(u,v)\in C=h(S)} w(u, v)$
   - **TSP:** $c(h(S), G) = -\sum_{i=1}^{|S|-1} w(S[i], S[i+1]) - w(S[|S|], S(1))$
   - **Min vertex cover:** $c(h(S), G) = -|S|$

General greedy algorithm formulation

4. Add node that maximizes an evaluation function $Q(h(S), v)$:

$$S \leftarrow (S, v^*) \text{ where } v^* = \underset{v \notin S}{\operatorname{argmax}}\, Q(h(S), v)$$

5. Terminate based on termination criterion $t\big(h(S)\big)$

This paper: Use RL to learn evaluation function $\hat{Q}(h(S), v; \underbrace{\Theta}_{\text{Model parameters}})$

Outline

1. Greedy algorithms
2. **Graph representation**
3. RL formulation
4. Q-learning
5. Experiments

Representation: graph embedding

- $x_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{else} \end{cases}$
- Compute embedding over T iterations ($\boldsymbol{\mu}_v^{(0)} = \mathbf{0}$):

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow \text{relu}\left(\boldsymbol{\theta}_1 x_v + \boldsymbol{\theta}_2 \sum_{u \in N(v)} \boldsymbol{\mu}_u^{(t)} + \boldsymbol{\theta}_3 \sum_{u \in N(v)} \text{relu}\big(\boldsymbol{\theta}_4 w(v,u)\big)\right)$$

Trainable parameters

structure2vec framework of Dai et al. [ICML'16]

Representation: graph embedding

- $x_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{else} \end{cases}$
- Compute embedding over T iterations ($\boldsymbol{\mu}_v^{(0)} = \mathbf{0}$):

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow \text{relu}\left(\boldsymbol{\theta}_1 x_v + \boldsymbol{\theta}_2 \sum_{u \in N(v)} \boldsymbol{\mu}_u^{(t)} + \boldsymbol{\theta}_3 \sum_{u \in N(v)} \text{relu}\big(\boldsymbol{\theta}_4 w(v,u)\big)\right)$$

(Usually $T = 4$)

- $$\hat{Q}(h(S), v; \Theta) = \boldsymbol{\theta}_5^{\top} \text{relu}\left(\underbrace{\left[\boldsymbol{\theta}_6 \textstyle\sum_{u \in V} \boldsymbol{\mu}_u^{(T)}, \boldsymbol{\theta}_7 \boldsymbol{\mu}_u^{(T)}\right]}_{\text{Concatenation}}\right)$$

structure2vec framework of Dai et al. [ICML'16]

Representation: graph embedding

- $x_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{else} \end{cases}$
- Compute embedding over T iterations ($\boldsymbol{\mu}_v^{(0)} = \mathbf{0}$):

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow \text{relu}\left(\boldsymbol{\theta}_1 x_v + \boldsymbol{\theta}_2 \sum_{u \in N(v)} \boldsymbol{\mu}_u^{(t)} + \boldsymbol{\theta}_3 \sum_{u \in N(v)} \text{relu}(\boldsymbol{\theta}_4 w(v,u))\right)$$

(Usually $T = 4$)

- $\hat{Q}(h(S), v; \Theta) = \boldsymbol{\theta}_5^\top \text{relu}\left(\left[\underbrace{\boldsymbol{\theta}_6 \sum_{u \in V} \boldsymbol{\mu}_u^{(T)}}_{\text{Surrogate for } h(S)}, \underbrace{\boldsymbol{\theta}_7 \boldsymbol{\mu}_u^{(T)}}_{\text{Surrogate for } v}\right]\right)$

structure2vec framework of Dai et al. [ICML'16]

Outline

1. Greedy algorithms
2. Graph representation
3. **RL formulation**
4. Q-learning
5. Experiments

Reinforcement learning formulation

State: $\sum_{u \in V} \boldsymbol{\mu}_u^{(T)}$

Action: Choose vertex $v \in V \setminus S$ to add to solution

Transition (deterministic): For chosen $v \in V \setminus S$, set $x_v = 1$

Reinforcement learning formulation

Reward: $r(S, v)$ is objective change when move to $S' = (S, v)$

$$r(S, v) = c(h(S'), G) - c(h(S), G)$$

$c(h(\emptyset), G) = 0$, so cumulative reward of **terminal state** $\hat{S}$ is

$$\sum_{i=1}^{|\hat{S}|} r(S_i, v_i) = c\big(h(\hat{S}), G\big)$$

Policy (deterministic): $\pi(v|S) = \begin{cases} 1 & \text{if } v = \operatorname{argmax}_{v' \notin S} \hat{Q}(h(S), v'; \Theta) \\ 0 & \text{else} \end{cases}$

Outline

1. Greedy algorithms
2. Graph representation
3. RL formulation
4. **Q-learning**
5. Experiments

Q-learning

Recall standard (1-step) Q-learning:

$$\min_{\Theta}\left(y-\hat{Q}(h(S_t),v_t;\Theta)\right)^2$$

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

Challenge:

- Final objective value only revealed after many steps
- 1-step update may be too myopic

Instead, use ***n*-step** Q-learning [Watkins, '89]

n-step Q-learning

$$\min_{\Theta} \left(y - \hat{Q}(h(S_t), v_t; \Theta) \right)^2$$

where $y = \sum_{i=0}^{n-1} \gamma^i r(S_{t+1}, v_{t+i}) + \gamma^n \max_{v'} \hat{Q}(h(S_{t+n}), v'; \Theta)$

Q-learning for the greedy algorithm

initialize set $M = \emptyset$

for episode $e = 1, \dots, L$:

    sample graph G from underlying distribution D

    initialize state to empty $S_1 = ()$

Q-learning for the greedy algorithm

for episode $e = 1, \ldots, L$:

for step $t = 1, \ldots, T$:

$$v_t = \begin{cases} \text{random node } v \notin S_t & \text{with probability } \epsilon \\ \underset{v \notin S_t}{\text{argmax}}\, \hat{Q}(h(S_t), v; \Theta) & \text{otherwise} \end{cases}$$

add v_t to **partial solution** $S_{t+1} = (S_t, v_t)$

if $t \geq n$:

experience replay

add tuple $(S_{t-n}, v_{t-n}, \sum_{i=1}^{n} R(S_{t-i}, v_{t-i}), S_t)$ to M

sample batch $B \sim M$

update Θ using SGD over B

Outline

1. Greedy algorithms
2. Graph representation
3. RL formulation
4. Q-learning
5. **Experiments**

Approximation ratio

Results measured in terms of approximation ratio

$$\frac{\text{Algorithm's solution}}{\text{OPT}}$$

Min vertex cover

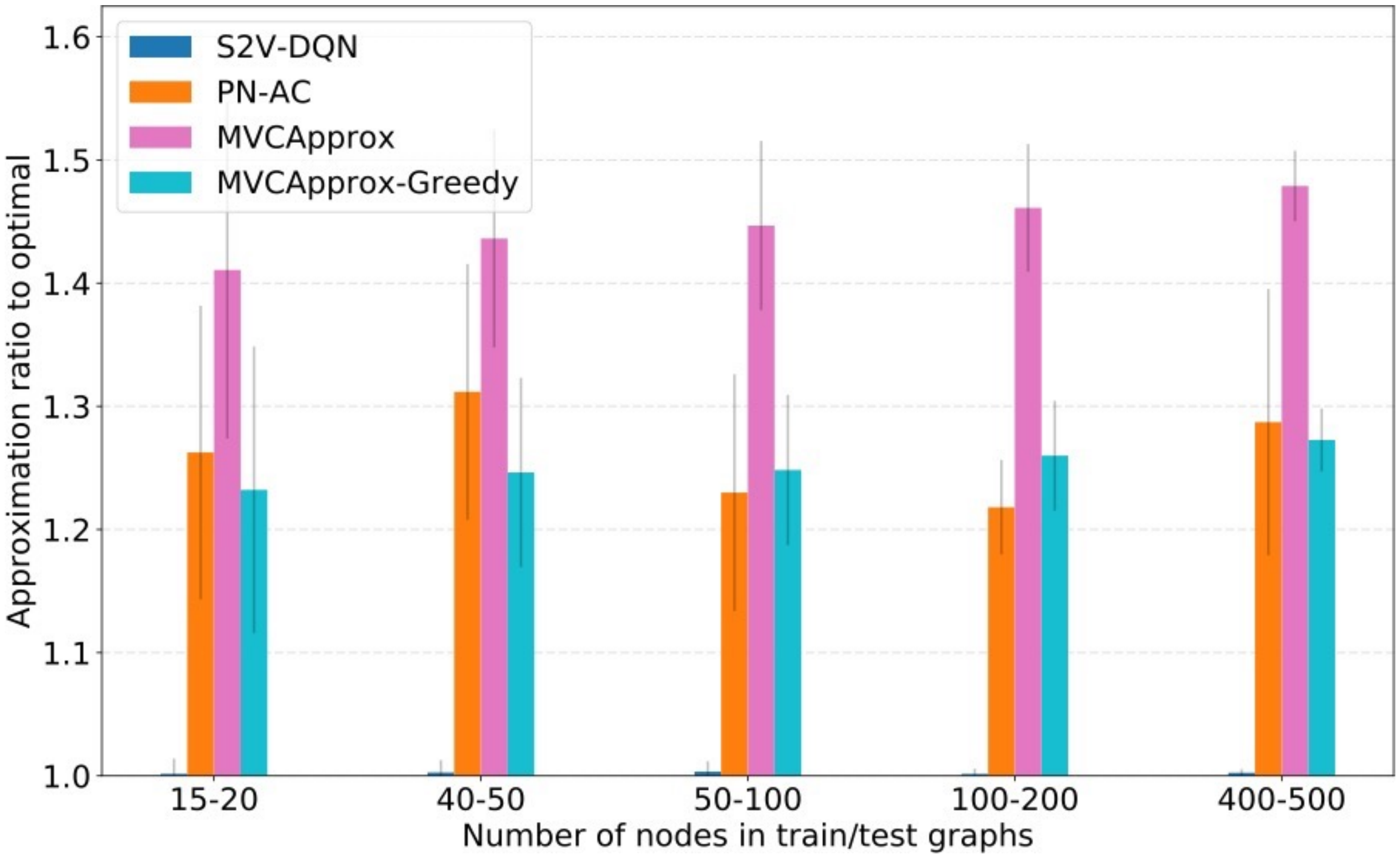
Barabasi-Albert
random graphs

Paper's approach

Another DL approach
[Bello et al., arXiv'16]

2-approximation
algorithm

Greedy algorithm
from first few slides



Max cut

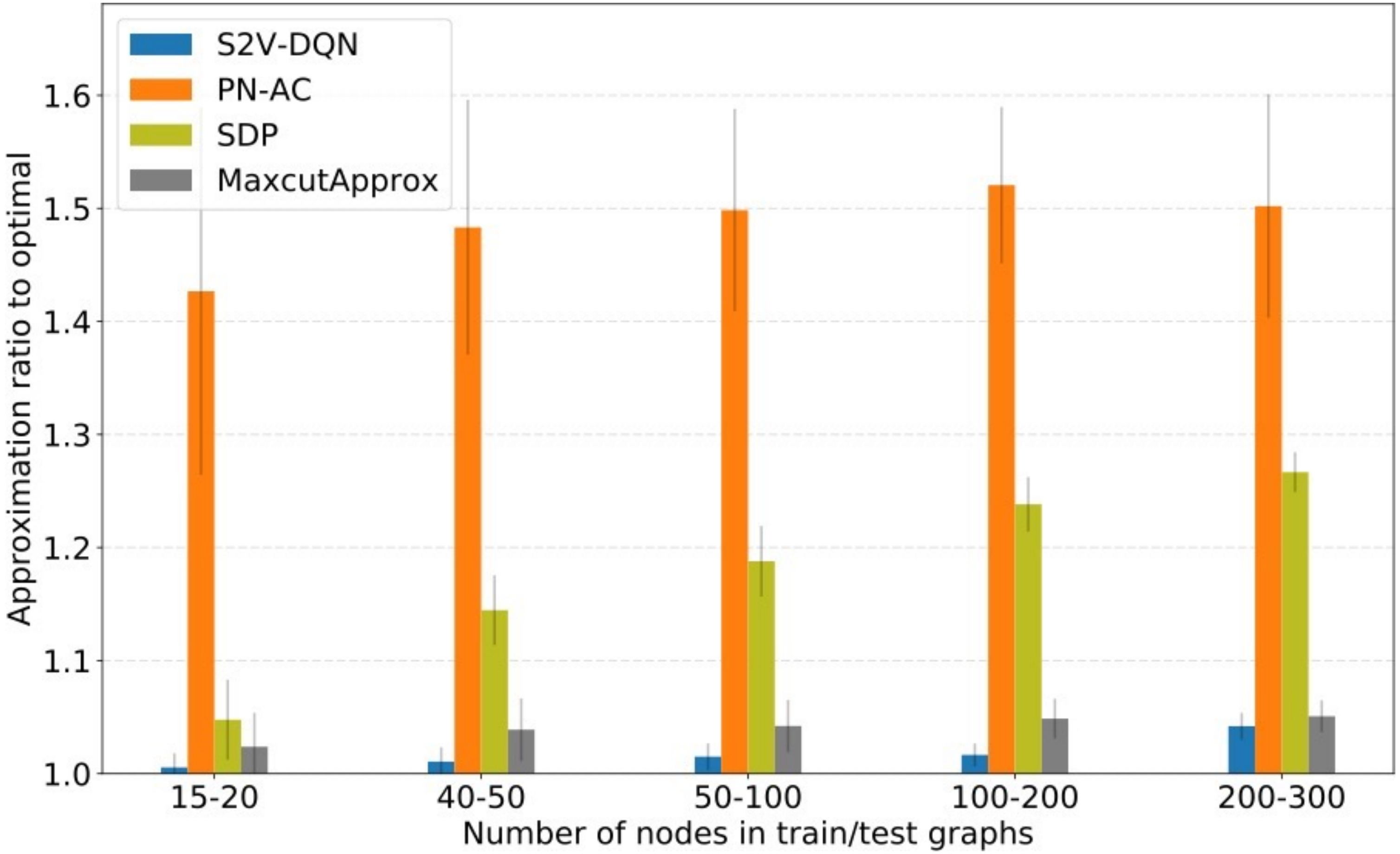
Barabasi-Albert random graphs

Paper's approach

Another DL approach [Bello et al., arXiv'16]

Goemans-Williamson algorithm

Greedy algorithm from first few slides



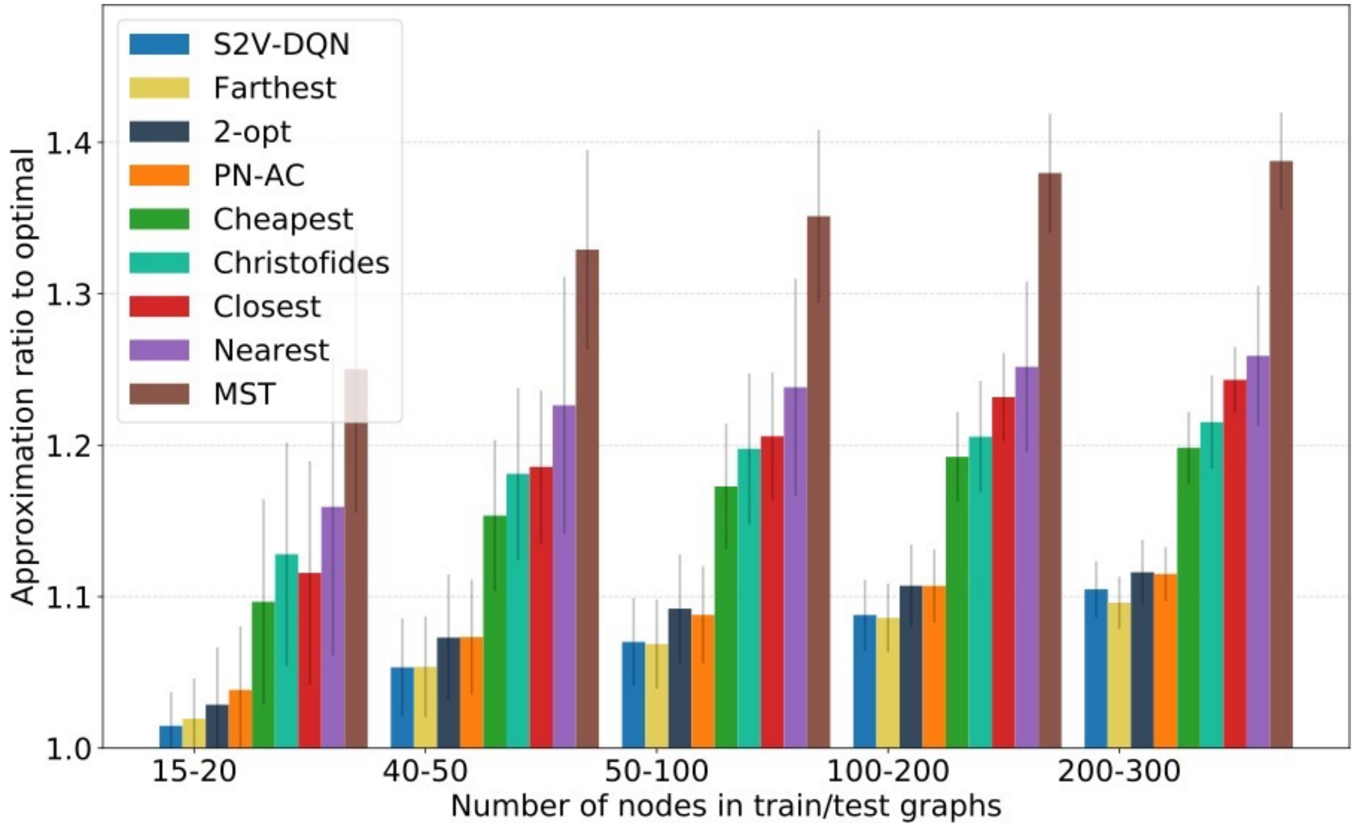
TSP

Uniform random points on 2-D grid

Paper's approach

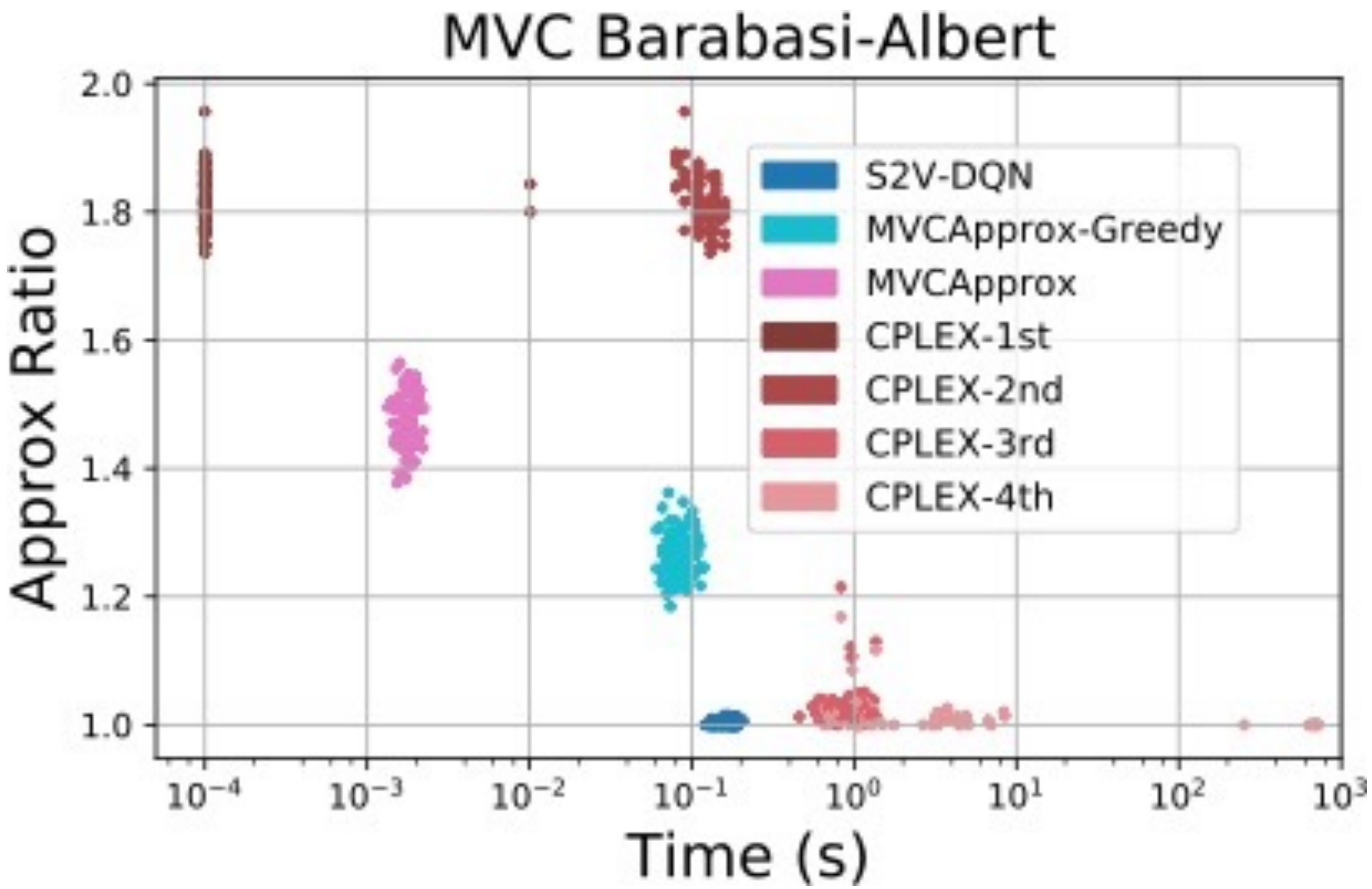
- Initial subtour: 2 cities that are farthest apart
- Repeat the following:
    - Choose city that's *farthest* from any city in the subtour
    - Insert in position where it causes the smallest distance increase

[Rosenkrantz et al., SIAM JoC'77]

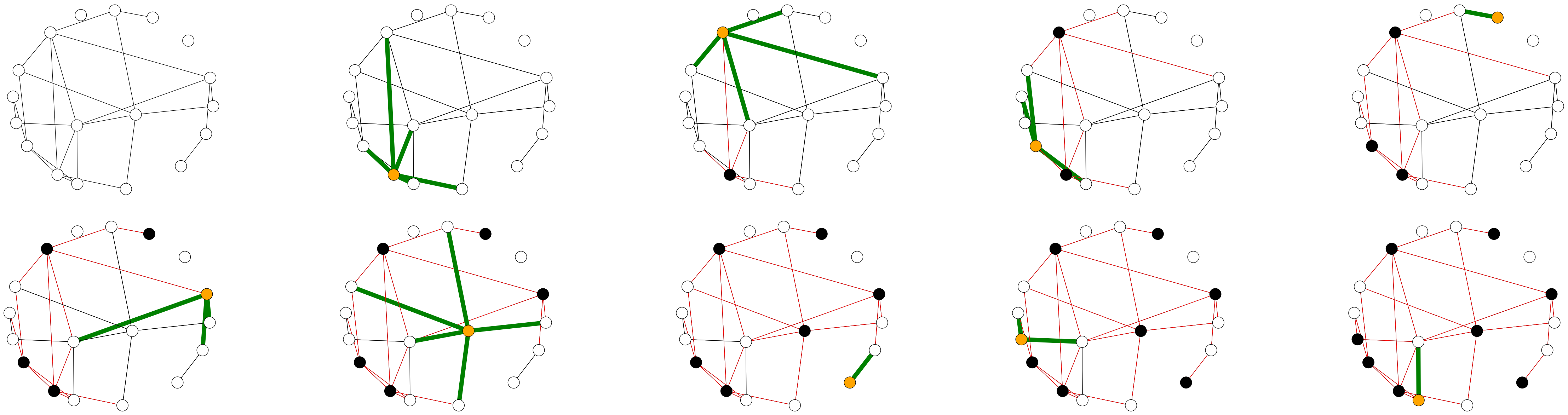


Runtime comparisons

CPLEX-1st: 1st feasible solution found by CPLEX



Min vertex cover visualization



Nodes seem to be selected to balance between:

- Degree
- Connectivity of the remaining graph

Overview

Learn greedy heuristics for hard combinatorial problem

Approach based on graph representation + RL

Suggest approach could be used for **algorithm discovery**

“New and interesting” greedy strategies

“which **intuitively make sense** but have **not been analyzed** before,” thus could be a “good **assistive tool** for discovering new algorithms.”