Learning Heuristics for Combinatorial Optimization Problems over Graphs using RL

An Architecture and Discussion of Potential Directions

Alperen Tercan

May 2020

Table of Contents

Proposed Flowchart

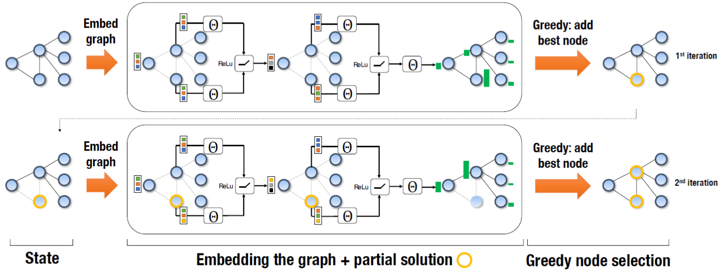


Figure 1: Illustration of the proposed framework as applied to an instance of Minimum Vertex Cover. The middle part illustrates two iterations of the graph embedding, which results in node scores (green bars).

High Level Differences between Two Papers

S2V-DQN vs GCOMB

S2V-DQN:

- No supervision, end-to-end learning using Q-learning

High Level Differences between Two Papers

S2V-DQN vs GCOMB

S2V-DQN:

- No supervision, end-to-end learning using Q-learning
- Node embeddings depend on current solution set $\mathcal{S}$. Therefore, new embeddings must be calculated each of b iterations.

High Level Differences between Two Papers
S2V-DQN vs GCOMB

S2V-DQN:

- No supervision, end-to-end learning using Q-learning
- Node embeddings depend on current solution set $\mathcal{S}$. Therefore, new embeddings must be calculated each of b iterations.
- A more general framework

High Level Differences between Two Papers

S2V-DQN vs GCOMB

S2V-DQN:

- No supervision, end-to-end learning using Q-learning
- Node embeddings depend on current solution set $\mathcal{S}$. Therefore, new embeddings must be calculated each of b iterations.
- A more general framework
- Scalability issues

A more general problem formulation

- Previous paper discussed only budget-constraint problems

A more general problem formulation

- Previous paper discussed only budget-constraint problems
- Using the following three components, we can formulate a much more diverse set of combinatorial problems
    - Objective function $c(\mathcal{S}, G)$
    - Termination condition
    - Helper function

A more general problem formulation - Examples

- **Example - Minimum Vertex Cover:**
  - No helper procedure is needed
  - $c(\mathcal{S}, G) = -|\mathcal{S}|$
  - Terminate when all edges are covered

A more general problem formulation - Examples

- **Example - Minimum Vertex Cover:**
  - No helper procedure is needed
  - $c(\mathcal{S}, G) = -|\mathcal{S}|$
  - Terminate when all edges are covered

- **Example - Maximum Cut:**
  - Helper function maintains a cut-set $\mathcal{C} = \{(u, v) | (u, v) \in E, u \in \mathcal{S}, v \in \mathcal{V} \setminus \mathcal{S}\}$
  - $c(\mathcal{S}, G) = \sum_{(u,v)\in\mathcal{C}} w(u, v)$
  - Terminate when cut weight cannot be improved

Structure2Vec(S2V)

- **Goal :** Compute p-dimensional embeddings $\boldsymbol{\mu}_v$ for each $v \in V$

Structure2Vec(S2V)

- **Goal :** Compute p-dimensional embeddings $\boldsymbol{\mu}_v$ for each $v \in V$
- Generic synchronous update equation:

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow F(x_v, \{\boldsymbol{\mu}_u^{(t)}\}_{u \in \mathcal{N}(v)}, \{w(v,u)\}_{u \in \mathcal{N}(v)}; \Theta)$$

  - x_v is raw features
  - $\boldsymbol{\mu}_v^{(t)}$ is embedding of node v in iteration t of embedding process.
  - $\boldsymbol{\mu}_v^{(0)}$ is 0
  - $\boldsymbol{\mu}_v^{(T)}$ is the desired embedding, where T is *the depth* ~ 4
  - F is a generic nonlinear mapping, like a NN or kernel function
  - $\mathcal{N}(v)$ is the set of neighbors of node v
  - $w(v,u)$ weight of edge (v,u)

Embeddings in This Paper

- **Goal :** Compute p-dimensional embeddings $\boldsymbol{\mu}_v$ for each $v \in V$ **given solution set** S

Embeddings in This Paper

- **Goal :** Compute p-dimensional embeddings $\boldsymbol{\mu}_v$ for each $v \in V$ **given solution set** S
- Update equation:

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow ReLU(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \boldsymbol{\mu}_u^{(t)} + \theta_3 \sum_{u \in \mathcal{N}(v)} ReLU(\theta_4 w(v,u))$$

  - x_v is binary scalar, 1 if $v \in \mathcal{S}$, 0 else $\rightarrow$ **can be generalized to a vector**
  - $ReLU$: Rectified Linear Unit, $ReLU(z) = max(0, z)$ elementwise

Embeddings in This Paper

- **Goal :** Compute p-dimensional embeddings $\boldsymbol{\mu}_v$ for each $v \in V$ **given solution set** S
- Update equation:

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow ReLU(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \boldsymbol{\mu}_u^{(t)} + \theta_3 \sum_{u \in \mathcal{N}(v)} ReLU(\theta_4 w(v,u))$$

  - x_v is binary scalar, 1 if $v \in \mathcal{S}$, 0 else $\rightarrow$ **can be generalized to a vector**
  - $ReLU$: Rectified Linear Unit, $ReLU(z) = max(0, z)$ elementwise
- if needed, we can add more nonlinearities like additional layers of $ReLU$

Embeddings in This Paper

- **Goal :** Compute p-dimensional embeddings $\boldsymbol{\mu}_v$ for each $v \in V$ **given solution set** S
- Update equation:

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow ReLU(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \boldsymbol{\mu}_u^{(t)} + \theta_3 \sum_{u \in \mathcal{N}(v)} ReLU(\theta_4 w(v,u))$$

  - x_v is binary scalar, 1 if $v \in \mathcal{S}$, 0 else $\rightarrow$ **can be generalized to a vector**
  - $ReLU$: Rectified Linear Unit, $ReLU(z) = max(0, z)$ elementwise
- if needed, we can add more nonlinearities like additional layers of $ReLU$
- **Possible Problem:** There are no connections between $\boldsymbol{\mu}_u$ and $w(v, u)$

Embeddings Comparison

Compare two embedding rules :

- **S2V-DQN**

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow ReLU(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \boldsymbol{\mu}_u^{(t)} + \theta_3 \sum_{u \in \mathcal{N}(v)} ReLU(\theta_4 w(v,u))$$

Embeddings Comparison

Compare two embedding rules :

- **S2V-DQN**

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow ReLU(\theta_1 x_v + \theta_2 \sum_{u \in \mathcal{N}(v)} \boldsymbol{\mu}_u^{(t)} + \theta_3 \sum_{u \in \mathcal{N}(v)} ReLU(\theta_4 w(v,u))$$

- **GCOMB**

$$\boldsymbol{\mu}_v^{(t+1)} \leftarrow ReLU(\theta_t\, [\boldsymbol{\mu}_v^{(t)}, \sum_{u \in \mathcal{N}(v)} \frac{w(v,u) \times \boldsymbol{\mu}_u^{(t)}}{\sum_{u' \in \mathcal{N}(v)} w(v,u')}])$$

Q-function approximator

- **Goal :** Approximate Q-function, i.e. predict the value of an action a in state S
  - Action(v) : Add node v, $\boldsymbol{\mu}_v^T$
  - State(h(S)) : A representation that depends on graph G and current solution set $S \rightarrow \sum_{u \in V} \boldsymbol{\mu}_u^T$

Q-function approximator

- **Goal :** Approximate Q-function, i.e. predict the value of an action a in state S
  - Action(v) : Add node v, $\boldsymbol{\mu}_v^T$
  - State(h(S)) : A representation that depends on graph G and current solution set $S \rightarrow \sum_{u \in V} \boldsymbol{\mu}_u^T$
- **Function:**

$$\hat{Q}(h(S), v; \Theta) = \theta_5^T ReLU\,([\,\theta_6 \sum_{u \in V} \boldsymbol{\mu}_u^T,\ \theta_7 \boldsymbol{\mu}_v^T\ )$$

Q-function approximator

- **Goal :** Approximate Q-function, i.e. predict the value of an action a in state S
  - Action(v) : Add node v, $\boldsymbol{\mu}_v^T$
  - State(h(S)) : A representation that depends on graph G and current solution set $S \rightarrow \sum_{u \in V} \boldsymbol{\mu}_u^T$
- **Function:**

$$\hat{Q}(h(S), v; \Theta) = \theta_5^T ReLU \left(\left[\theta_6 \sum_{u \in V} \boldsymbol{\mu}_u^T , \ \theta_7 \boldsymbol{\mu}_v^T \right) \right.$$

- **Note:** $\hat{Q}$ depends on all 7 parameters $\{\theta_i\}_{i=1}^7$

Q-function approximator - Comparison

Compare two Q-approximators:

- **S2V-DQN:**

$$\hat{Q}(h(S), v; \Theta) = \theta_5^T \, ReLU \, ([\, \theta_6 \sum_{u \in \mathcal{V}} \boldsymbol{\mu}_u^T, \, \theta_7 \boldsymbol{\mu}_v^T \,)$$

Q-function approximator - Comparison

Compare two Q-approximators:

- **S2V-DQN:**

$$\hat{Q}(h(S), v; \Theta) = \theta_5^T ReLU\left([\, \theta_6 \sum_{u \in \mathcal{V}} \boldsymbol{\mu}_u^T,\ \theta_7 \boldsymbol{\mu}_v^T\ \right)$$

- **GCOMB:**

$$\begin{aligned}\hat{Q}(h(S), v; \Theta) = \vartheta_4^T\ ReLU\,([&\vartheta_1 MAXPOOL(\{\boldsymbol{\mu}_u, u \in \mathcal{S}\}),\\ &\vartheta_2 MAXPOOL(\{\boldsymbol{\mu}_u, u \in \mathcal{V} \setminus \mathcal{S}\}),\\ &\vartheta_3 \boldsymbol{\mu}_v])\end{aligned}$$

Training: Q-learning

- **Goal :** Train the parameter set $\{\theta_i\}_{i=1}^{7}$

Training: Q-learning

- **Goal :** Train the parameter set $\{\theta_i\}_{i=1}^{7}$
- **How ?:** Use Q-learning

Quick overview of Q-learning

- Calculate TD-target for $\hat{Q}(\mathcal{S}_t, v_t)$:
  $y = \gamma max_{v'}(\hat{Q}(\mathcal{S}_{t+1}, v'; \Theta)) + r(\mathcal{S}_t, v_t)$

Quick overview of Q-learning

- Calculate TD-target for $\hat{Q}(\mathcal{S}_t, v_t)$:
  $y = \gamma max_{v'}(\hat{Q}(\mathcal{S}_{t+1}, v'; \Theta)) + r(\mathcal{S}_t, v_t)$
- Minimize MSE with prediction : $minimize(y - \hat{Q}(\mathcal{S}_t, v_t))$

Quick overview of Q-learning

- Calculate TD-target for $\hat{Q}(\mathcal{S}_t, v_t)$:
  $y = \gamma max_{v'}(\hat{Q}(\mathcal{S}_{t+1}, v'; \Theta)) + r(\mathcal{S}_t, v_t)$
- Minimize MSE with prediction : $minimize(y - \hat{Q}(\mathcal{S}_t, v_t))$
- **Target Networks:**
  - Notice that both y and $\hat{Q}(\mathcal{S}_t, v_t)$ depend on $\Theta \longrightarrow$ stability issues
  - **Solution:** Use two set of parameters, Θ_{act} and Θ_{target}. Use $\Theta = \Theta_{target}$ for y and update Θ_{target} to Θ_{act} every τ iterations.

Quick overview of Q-learning

- Calculate TD-target for $\hat{Q}(\mathcal{S}_t, v_t)$:
  $y = \gamma max_{v'}(\hat{Q}(\mathcal{S}_{t+1}, v'; \Theta)) + r(\mathcal{S}_t, v_t)$
- Minimize MSE with prediction : $minimize(y - \hat{Q}(\mathcal{S}_t, v_t))$
- **Target Networks:**
  - Notice that both y and $\hat{Q}(\mathcal{S}_t, v_t)$ depend on $\Theta \longrightarrow$ stability issues
  - **Solution:** Use two set of parameters, Θ_{act} and Θ_{target}. Use $\Theta = \Theta_{target}$ for y and update Θ_{target} to Θ_{act} every τ iterations.
- **Replay Buffer:**
  - Notice that cost function only depends on the information about 1-step transition, i.e. policy independent(*off-policy*)
  - Save transitions to a buffer and use them repeatedly $\longrightarrow$ Sample efficiency

Quick overview of Q-learning

- *Off-policy vs on-policy*
  - *Off-policy:*
    $Q(s,a) \leftarrow Q(s,a) + \alpha[(r(s,a) + \gamma\, max_{a'} Q(s',a')) - Q(s,a)]$

Quick overview of Q-learning

- *Off-policy vs on-policy*
  - *Off-policy:* $Q(s,a) \leftarrow Q(s,a) + \alpha[(r(s,a) + \gamma\, max_{a'} Q(s',a')) - Q(s,a)]$
  - *On-policy:* $Q(s,a) \leftarrow Q(s,a) + \alpha[(r(s,a) + \gamma\, Q(s',a')) - Q(s,a)]$

Training: Q-learning, cont'd

- **Goal :** Train the parameter set $\{\theta_i\}_{i=1}^{7}$
- **How ?:** Use Q-learning
- **With a modification:** Use $R_{t,t+n} = \sum_{i=0}^{n-1} r(S_{t+i}, a_{t+i})$ instead of $r(S_t, v_t)$

Training: Q-learning, cont'd

- **Goal :** Train the parameter set $\{\theta_i\}_{i=1}^{7}$
- **How ?:** Use Q-learning
- **With a modification:** Use $R_{t,t+n} = \sum_{i=0}^{n-1} r(S_{t+i}, a_{t+i})$ instead of $r(S_t, v_t)$
- **Idea:** Get the algorithm foresighted faster

Training: Q-learning, cont'd

- **Goal :** Train the parameter set $\{\theta_i\}_{i=1}^7$
- **How ?:** Use Q-learning

- **With a modification:** Use $R_{t,t+n} = \sum_{i=0}^{n-1} r(S_{t+i}, a_{t+i})$ instead of $r(S_t, v_t)$
- **Idea:** Get the algorithm foresighted faster
- **Possible Problem:** It breaks the *off-policy* assumptions.

Q-learning

```
Algorithm 1: Q-learning
Input: Data distribution $\mathbb{D}$
Output: Parameter set $\Theta$
1  Initialize experience replay memory $M$ to capacity $N$
2  for episode $e = 1$ to $L$ do
3      Draw graph $G$ from distribution $\mathbb{D}$
4      Initialize the state to empty $S_1 = \{\}$
5      for step $t = 1$ to $T$ do
6          $v_t = \begin{cases} \textit{random node } v \in V \setminus S_t, & \text{with probability } \epsilon \\ \arg\max_{v \in V \setminus S_t} \hat{Q}(S_t, v; \Theta), & \text{otherwise} \end{cases}$
7          Add $v_t$ to partial solution: $S_{t+1} := S_t \cup \{v_t\}$
8          if $t >= n$ then
9              Add tuple $(S_{t-n}, v_{t-n}, R_{t-n}, S_t)$ to $M$
10             Sample random batch $B \sim M$
11             Update $\Theta$ by SGD over $(y - \hat{Q}(S_t, v_t; \Theta))^2$ for $B$
12         end
13     end
14 end
Return: $\Theta$
```

Q-learning

```
Algorithm 2: Q-learning
Input: A set of training graphs $\mathbb{D}$, hyperparameters $M, N, n, L, T$
Output: Parameter set $\Theta$
1  Initialize experience replay memory $M$ to capacity $N$
2  for episode $e = 1$ to $L$ do
3      Draw graph $G$ from dataset $\mathbb{D}$
4      Initialize the state to $s_0 = (G, \emptyset)$                /* $s_t = (G, S_t)$ */
5      for step $t = 1$ to $T$ do
6          $a_t = \begin{cases} \text{random node } v \in G.V \setminus S_t, & \text{with probability } \epsilon \\ \arg\max_{v \in G.V \setminus S_t} \hat{Q}(s_t, v; \Theta), & \text{otherwise} \end{cases}$
7          Add $a_t$ to partial solution: $S_{t+1} := S_t \cup \{a_t\}$
8          if $t >= n$ then
9              Add tuple $(s_{t-n}, a_{t-n}, R_{t-n}, s_t)$ to $M$
10             Sample random batch $B \sim M$
11             Update $\Theta$ by SGD over $(y - \hat{Q}(s_t, a_t; \Theta))^2$ for $B$
12         end
```

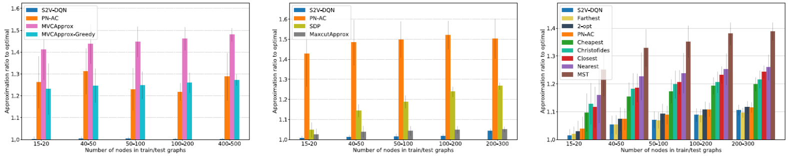
test

test

Experimental Results

- Baseline Algorithms:
  - PN-AC $\longrightarrow$ RNN based sequence-to-sequence network
  - Problem specific well-known approximation algorithms
- Metric: Approximation ratio $R(S, G) = max(\frac{OPT(G)}{c(S)}, \frac{c(S)}{OPT(G)})$ where $OPT(G)$ is the answer from CPLEX or Concorde in 1 hour

Approximation Ratios



(a) MVC BA (b) MAXCUT BA (c) TSP random

Figure 2: Approximation ratio on 1000 test graphs. Note that on MVC, our performance is pretty close to optimal. In this figure, training and testing graphs are generated according to the same distribution.

- Notice the good approximation ratios of S2V-DQN

Generalization

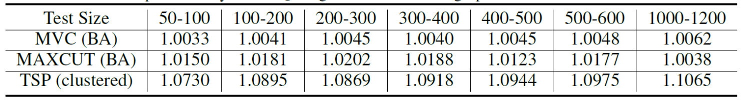
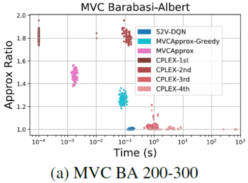


Table 2: S2V-DQN's generalization ability. Values are average approximation ratios over 1000 test instances. These test results are produced by S2V-DQN algorithms trained on graphs with 50-100 nodes.

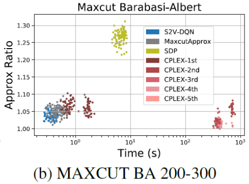
Test Size	50-100	100-200	200-300	300-400	400-500	500-600	1000-1200
MVC (BA)	1.0033	1.0041	1.0045	1.0040	1.0045	1.0048	1.0062
MAXCUT (BA)	1.0150	1.0181	1.0202	1.0188	1.0123	1.0177	1.0038
TSP (clustered)	1.0730	1.0895	1.0869	1.0918	1.0944	1.0975	1.1065

- Notice that for some approximation ratio is getting better $\longrightarrow$ Time-constrainted performance of CPLEX get worse for large graphs

Scalability Trade-off between running time and approximation ratio



(a) MVC BA 200-300



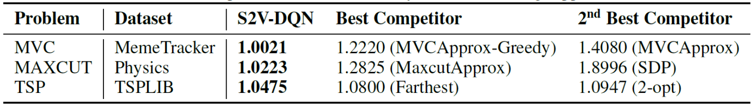
(b) MAXCUT BA 200-300

Figure 3: Time-approximation trade-off for MVC and MAX-CUT. In this figure, each dot represents a solution found for a single problem instance, for 100 instances. For CPLEX, we also record the time and quality of each solution it finds, e.g. CPLEX-1st means the first feasible solution found by CPLEX.

- Testing complexity : $O(k|E|)$ where k is the number of greedy steps $k <= |V|$

Real-world Datasets

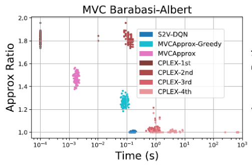
Table 3: Realistic data experiments, results summary. Values are average approximation ratios.



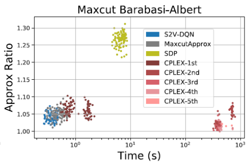
Problem	Dataset	S2V-DQN	Best Competitor	2^{nd} Best Competitor
MVC	MemeTracker	**1.0021**	1.2220 (MVCApprox-Greedy)	1.4080 (MVCApprox)
MAXCUT	Physics	**1.0223**	1.2825 (MaxcutApprox)	1.8996 (SDP)
TSP	TSPLIB	**1.0475**	1.0800 (Farthest)	1.0947 (2-opt)

- They use a very small proportion of datasets
    - MemeTracker : 96m nodes originally, $\sim$ 1000 nodes here.
    - TSPLib : 85k nodes originally, $\sim$ 300 nodes here.

Scalibility Trade-off between running time and approximation ratio



(a) MVC BA 200-300



(b) MAXCUT BA 200-300

Figure 3: Time-approximation trade-off for MVC and MAXCUT. In this figure, each dot represents a solution found for a single problem instance, for 100 instances. For CPLEX, we also record the time and quality of each solution it finds, e.g. CPLEX-1st means the first feasible solution found by CPLEX.

- Testing complexity : $O(k|E|)$ where k is the number of greedy steps $< |V|$

Discovery of interesting new algorithms?

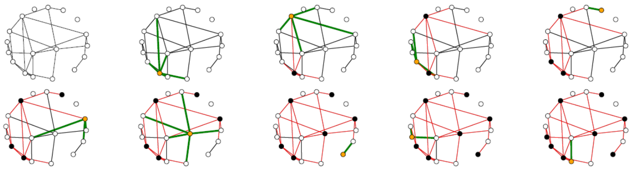


Figure D.4: Minimum Vertex Cover: an optimal solution to an ER graph instance found by S2V-DQN. Selected node in each step is colored in orange, and nodes in the partial solution up to that iteration are colored in black. Newly covered edges are in thick green, previously covered edges are in red, and uncovered edges in black. We show that the agent is not only picking the node with large degree, but also trying to maintain the connectivity after removal of the covered edges. For more detailed analysis, please see Appendix D.10.

- **Claim :** S2V-DQN learns interesting and novel heuristics.
- For instance:
    - for MVC nodes are selected to balance between their degrees and the connectivity of the remaining graph ($\rightarrow$ Can result in fewer steps than node and edge heuristics
    - For MAXCUT, nodes are picked to avoid cancelling out existing edges in the cut set

Table of Contents

A discussion on reliability of solutions

- There are no guarantees with these data-driven approaches.

A discussion on reliability of solutions

- There are no guarantees with these data-driven approaches.
- But testing on a large dataset gives an idea about what performance we should expect

A discussion on reliability of solutions

- There are no guarantees with these data-driven approaches.
- But testing on a large dataset gives an idea about what performance we should expect
- What if we get a sample that is different than anything we've seen so far?

A discussion on reliability of solutions

- There are no guarantees with these data-driven approaches.
- But testing on a large dataset gives an idea about what performance we should expect
- What if we get a sample that is different than anything we've seen so far?
- Probably, we shouldn't trust our model. But, how we will know that it is different? $\longrightarrow$ Anomaly detection, novelty detection

Approaches for novelty detection : Traditional

- Represent graphs somehow, then use some unsupervised learning or statistical methods to see if the new point is far from the ones in training data.

  Example method:

  1. Use an auto-encoder.
  2. Model will learn what kind of features of input is important, i.e. latent vector
  3. and how to construct whole thing back from these important features
  4. In testing, if reconstruction error is large $\longrightarrow$
     - it hasn't seen something similar to this before or
     - it was so rare that it didn't care about its error.

Approaches for novelty detection : Traditional

- Advantages:
  - Huge literature, a well-explored field.

Approaches for novelty detection : Traditional

- Advantages:
  - Huge literature, a well-explored field.
- Disadvantages:
  - We have only node embeddings. Is it enough? We may need to learn a separate representation that captures general features.
  - In addition to training overhead, these methods *might* be computationally expensive to run.
  - Although it is well-explored in traditional ML, new research is needed to adapt existing methods for graphs

Approaches for novelty detection : TD-error

- Q-learning tries to minimize TD-error
- If a state is visited frequently during training, predictions for its value will be more accurate
- $\longrightarrow$ we can use absolute value of TD-error to assess novelty of a state-action pair.

Approaches for novelty detection : TD-error

- Advantages:
  - We already have everything needed, no additional learning required

Approaches for novelty detection : TD-error

- Advantages:
  - We already have everything needed, no additional learning required
- Disadvantages:
  - We need a call to reward function r(s,a), to calculate TD-error. This means additional computation each step.
  - A new heuristic, not enough experimental results to evaluate its success (Simmons-Edler et al. 2019)

Approaches for novelty detection : TD-error

- Advantages:
  - We already have everything needed, no additional learning required
- Disadvantages:
  - We need a call to reward function r(s,a), to calculate TD-error. This means additional computation each step.
  - A new heuristic, not enough experimental results to evaluate its success (Simmons-Edler et al. 2019)
- **Note that these approaches tries to predict only the novelty of a graph, not how successful the model would be on that graph.**

A discussion on suboptimality of supervision

- In GCOMB work, a *score*(v) metric is calculated as a supervision signal using solutions from *greedy algorithm* $\rightarrow$ suboptimal

A discussion on suboptimality of supervision

- In GCOMB work, a $score(v)$ metric is calculated as a supervision signal using solutions from *greedy algorithm* $\rightarrow$ suboptimal
- Getting optimal solutions for large graphs is not tractable $\rightarrow$ the very problem we try to solve

A discussion on suboptimality of supervision

- In GCOMB work, a $score(v)$ metric is calculated as a supervision signal using solutions from *greedy algorithm* $\rightarrow$ suboptimal
- Getting optimal solutions for large graphs is not tractable $\rightarrow$ the very problem we try to solve
- But this is just training set; so, we prepare the questions ourselves. Can we cheat?

A discussion on suboptimality of supervision

- In GCOMB work, a $score(v)$ metric is calculated as a supervision signal using solutions from *greedy algorithm* $\rightarrow$ suboptimal
- Getting optimal solutions for large graphs is not tractable $\rightarrow$ the very problem we try to solve
- But this is just training set; so, we prepare the questions ourselves. Can we cheat?
- **An idea:** Instead of random ones, we can try to generate graphs that we know exact answers to.

Graphs with known answers

Existing work for TSP: (Arthur and Frendewey, 1988)

1. Uses 0-1 integer LP formulation of a generic TSP instance
2. Finds closed form solutions in terms of free parameters of the generic problem
3. Varies the parameters to generate graph-solution pairs

Graphs with known answers

- Advantages:
  - It generates TSP problems of many types: Symmetric, asymmetric, triangle-equality, hamiltonian-cycle etc.
  - It is empirically shown that generated problems are as hard as random ones
  - LP formulation and duality based methods are general enough to be applicable to other problems

Graphs with known answers

- Advantages:
  - It generates TSP problems of many types: Symmetric, asymmetric, triangle-equality, hamiltonian-cycle etc.
  - It is empirically shown that generated problems are as hard as random ones
  - LP formulation and duality based methods are general enough to be applicable to other problems
- Disadvantages:
  - The method needs to be written again for each new problem
  - A general framework that works for all problems might be very unlikely

The End

Table of Contents

References I

Amarbayasgalan, Tsatsral, Bilguun Jargalsaikhan, and Keun Ho Ryu (2018). "Unsupervised novelty detection using deep autoencoders with density based clustering". In: *Applied Sciences* 8.9, p. 1468.

Arthur, Jeffrey L. and James O. Frendewey (1988). "Generating Travelling-Salesman Problems with Known Optimal Tours". In: *The Journal of the Operational Research Society* 39.2, pp. 153–159. ISSN: 01605682, 14769360. URL: http://www.jstor.org/stable/2582378.

Dai, Hanjun et al. (2017). *Learning Combinatorial Optimization Algorithms over Graphs*. arXiv: 1704.01665 [cs.LG].

Mittal, Akash et al. (2019). *Learning Heuristics over Large Graphs via Deep Reinforcement Learning*. arXiv: 1903.03332 [cs.LG].

Simmons-Edler, Riley et al. (2019). *Reward Prediction Error as an Exploration Objective in Deep RL*. arXiv: 1906.08189 [cs.LG].

References II

Song, Hyungseok et al. (2019). "Solving Continual Combinatorial Selection via Deep Reinforcement Learning". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. DOI: 10.24963/ijcai.2019/481. URL: http://dx.doi.org/10.24963/ijcai.2019/481.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press. URL: http://incompleteideas.net/book/the-book-2nd.html.