Learning Combinatorial Optimization Algorithms over Graphs

Hanjun Dai, Elias B. Khalil, Yuyu Zhang,
Bistra Dilkina, and Le Song

NIPS 2017

Repeated optimization creates training data

A delivery company solves a new routing instance every day.

Instances differ, but may share a useful distribution D.

Learn a heuristic on graphs sampled from D.

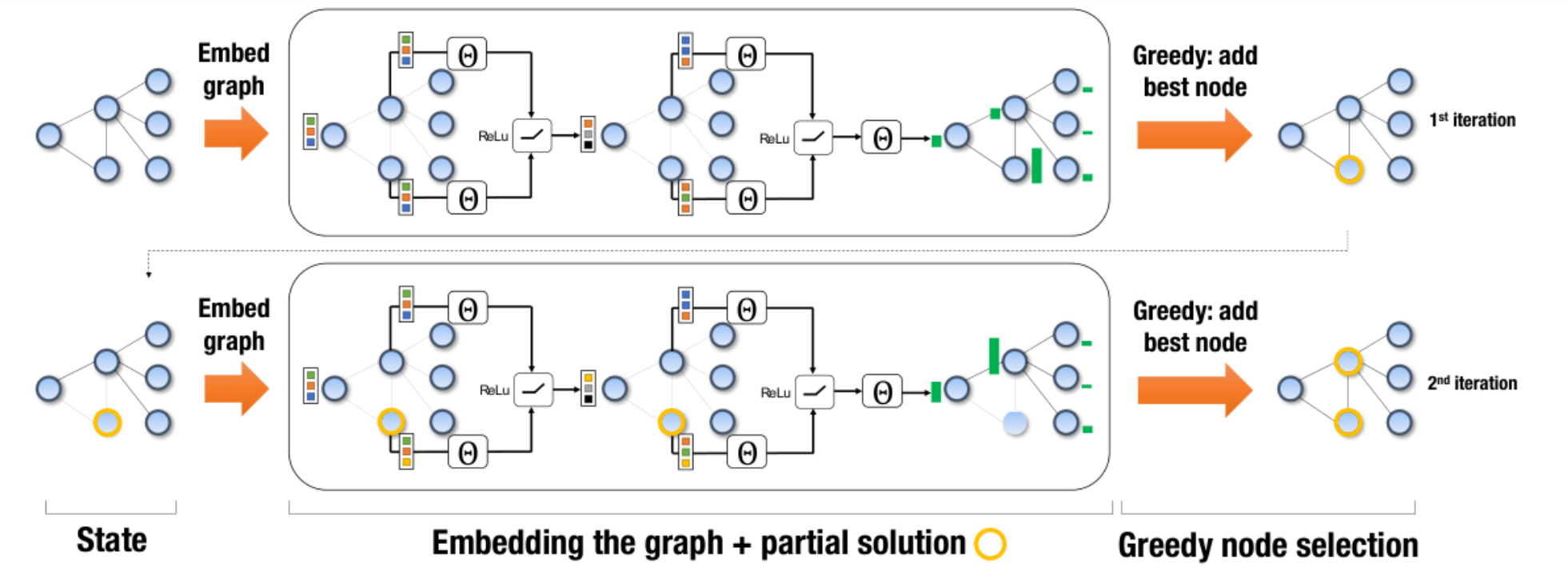
Evaluate solution quality and runtime on unseen graphs.

Three optimization problems

Problem	Solution	Objective
Vertex cover	Node subset covering all edges	Minimize selected nodes
Max-Cut	Partition of the node set	Maximize crossing weight
Euclidean TSP	Tour through every city	Minimize tour length

Common decision: which node should we add next?

A learned construction rule



Graph and partial solution determine the next node score.

What the designer supplies

Supplied by the designer	Learned from experience
Problem distribution D	Parameters of the node scoring function
Helper h(S) and legal candidates	How graph context affects node value
Objective c and stopping rule	A policy for sequential construction

$$v^* = \arg\max_{v \notin S} \hat{Q}(h(S), v)$$

Minimum vertex cover as an MDP

Component	Minimum vertex cover
State	Graph G and selected node set S
Action and transition	Choose $v \notin S$, then add v to S
Reward	−1 for each added node
Termination	Every edge has an endpoint in S

Worked example: the star graph

E = {(c,a), (c,b), (c,d), (c,e)}

Construction	Total return	Cover size
Choose c, then stop	−1	1
Choose a, b, d, e	−4	4

Every first action gives reward −1. Future consequences differ.

Q-values account for later decisions

$Q^*(\emptyset, c) = -1$ $Q^*(\emptyset, a) = -2$

$Q^*(S,v)$ measures return after choosing v and then acting optimally.

The greedy choice uses the largest Q-value: −1 is better than −2.

A learned $\hat{Q}$ approximates this ranking across many graph states.

Reward design matches the objective

$r(S,v) = c(h(S \cup \{v\}), G) - c(h(S), G)$

Σ rewards $= c(h(S_{final}), G) - c(h(\emptyset), G)$

MVC: $c = -|S|$ Max-Cut: c = cut weight
TSP: c = −tour length

Exact telescoping uses undiscounted return.

Max-Cut uses changes in crossing weight

r(S,v) = weight(v, outside S except v)
− weight(v, inside S)

Adding a node can remove edges from the current cut.

The selected set grows, but the cut weight need not grow.

Table 1 stops when the cut weight cannot improve.

TSP combines learning with insertion

$$\Delta L(i,v,j) = d(i,v) + d(v,j) - d(i,j)$$

The policy chooses the next unvisited city v.

The helper chooses the insertion position with minimum ΔL.

Reward is −ΔL. Stop after every city joins the tour.

What should the graph representation preserve?

Node features record which nodes the algorithm already selected.

Edges describe interactions among candidate decisions.

Shared parameters allow a variable number of nodes.

Relabeling nodes should relabel their scores consistently.

Structure2vec performs message passing

$\mu_v^{(0)} = 0$

$\mu_v^{(t+1)} = F(x_v, \{\mu_u^{(t)} : u \in N(v)\}, \text{edge weights})$

x_v = 1 if v has been selected, and 0 otherwise.

T rounds give each node information from its T-hop neighborhood.

The paper's embedding update

$$\mu_v^{(t+1)} = \mathrm{ReLU}\left(\theta_1 x_v + \theta_2 \sum_{u \in N(v)} \mu_u^{(t)} + \theta_3 \sum_{u \in N(v)} \mathrm{ReLU}(\theta_4 w_{vu}) \right)$$

Term	Information
$\theta_1 x_v$	Selected status of this node
$\theta_2 \Sigma \mu_u$	Context from neighboring nodes
$\theta_3 \Sigma \mathrm{ReLU}(\theta_4 w_{vu})$	Incident edge weights

Node context and graph context produce Q

$$g = \Sigma_{u \in V} \mu_u^{(T)}$$

$$Q(S, v; \Theta) = \theta_5^T \, \mathrm{ReLU}([\, \theta_6 g \, , \, \theta_7 \mu_v^{(T)} \,])$$

g summarizes the current tagged graph.

μ_v describes the candidate node in its local context.

One inference episode

1. Initialize the partial solution and task-specific helper.

2. Compute embeddings and score legal candidate nodes.

3. Select the highest-scoring node and update the solution.

4. Repeat until the task’s stopping rule holds.

Weights stay fixed during testing. The graph state changes.

Q-learning supplies its own training target

$$y = r(S,v) + \gamma \max_{v'} \hat{Q}(S',v')$$

$$\text{Loss} = [\, y - \hat{Q}(S,v;\Theta) \,]^2$$

The environment provides the reward and next state.

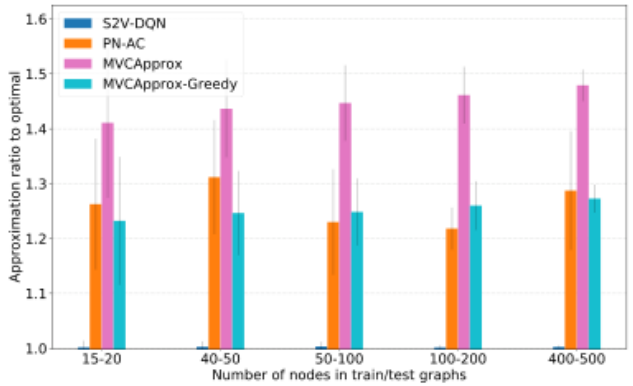
Backpropagation updates both embeddings and the scoring head.

Experimental design

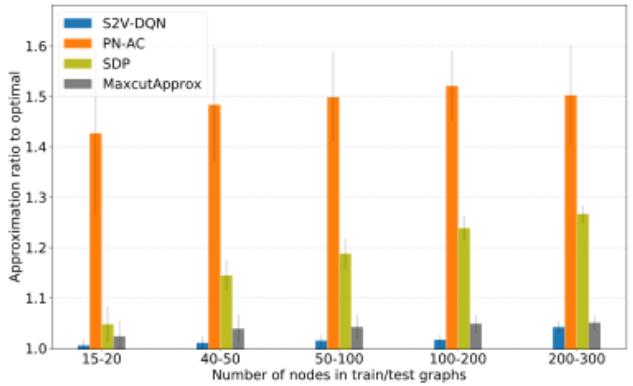
Task	Instances	Comparison methods
MVC	Erdős–Rényi and Barabási–Albert	Cover heuristics, PN-AC, CPLEX
Max-Cut	Weighted ER and BA graphs	Local search, SDP, PN-AC, CPLEX
TSP	Random or clustered 2D points	Insertion, 2-opt, PN-AC, Concorde

100 validation graphs and 1,000 test graphs
Solver references use a one-hour time limit.

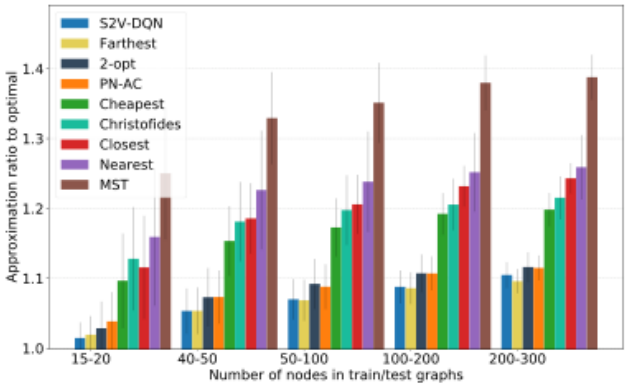
Solution quality on matched distributions



(a) MVC BA



(b) MAXCUT BA



(c) TSP random

Strong MVC and Max-Cut results against these baselines.
Classical TSP heuristics remain competitive.

Reported ratios compare with best solver solutions, not always proven optima.

Generalization to larger graphs

Task	50–100 nodes	400–500	1,000–1,200
MVC, BA	1.0033	1.0045	1.0062
Max-Cut, BA	1.0150	1.0123	1.0038
TSP, clustered	1.0730	1.0944	1.1065

Training size: 50–100 nodes for every row.
A fixed architecture permits transfer. The data determine its success.

Runtime and the cost of learning

Reported inference scaling: O(k |E|)

k is the number of node additions, at most |V|.

The paper reports MVC on 1,200 nodes within 11 seconds on a GPU.

Training cost must be amortized across repeated instances.

The scaling expression holds embedding settings fixed and omits helper details.

Takeaways and reference

A learned node score can guide a conventional construction algorithm.

Message passing represents graph structure and the partial solution.

Reinforcement learning trains the score without optimal action labels.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2017).
Learning Combinatorial Optimization Algorithms over Graphs. NIPS 30.
Attached version: arXiv:1704.01665v4, 21 February 2018.