Exploratory Combinatorial Optimization with Reinforcement Learning

Thomas D. Barrett,[1] **William R. Clements,**[2] **Jakob N. Foerster,**[3] **A. I. Lvovsky**[1,4]
[1]University of Oxford, Oxford, UK
[2]indust.ai, Paris, France
[3]Facebook AI Research
[4]Russian Quantum Center, Moscow, Russia
{thomas.barrett, alex.lvovsky}@physics.ox.ac.uk william.clements@indust.ai, jnf@fb.com

Abstract

Many real-world problems can be reduced to combinatorial optimization on a graph, where the subset or ordering of vertices that maximize some objective function must be found. With such tasks often NP-hard and analytically intractable, reinforcement learning (RL) has shown promise as a framework with which efficient heuristic methods to tackle these problems can be learned. Previous works construct the solution subset incrementally, adding one element at a time, however, the irreversible nature of this approach prevents the agent from revising its earlier decisions, which may be necessary given the complexity of the optimization task. We instead propose that the agent should seek to continuously improve the solution by learning to *explore at test time*. Our approach of exploratory combinatorial optimization (ECO-DQN) is, in principle, applicable to any combinatorial problem that can be defined on a graph. Experimentally, we show our method to produce state-of-the-art RL performance on the Maximum Cut problem. Moreover, because ECO-DQN can start from any arbitrary configuration, it can be combined with other search methods to further improve performance, which we demonstrate using a simple random search.

1 Introduction

NP-hard combinatorial problems – such as Travelling Salesman (Papadimitriou 1977), Minimum Vertex Cover (Dinur and Safra 2005) and Maximum Cut (Goemans and Williamson 1995) – are canonical challenges in computer science. With practical applications ranging from fundamental science to industry, efficient methods for approaching combinatorial optimization are of great interest. However, as no known algorithms are able to solve NP-hard problems in polynomial time, exact methods rapidly become intractable. Approximation algorithms guarantee a worst-case solution quality, but sufficiently strong bounds may not exist and, even if they do, these algorithms can have limited scalability (Williamson and Shmoys 2011). Instead, heuristics are often deployed that, despite offering no theoretical guarantees, are chosen for high performance.

There are numerous heuristic methods, ranging from search-based (Benlic and Hao 2013; Banks, Vincent, and Anyakoha 2008) to physical systems that utilise both quantum and classical effects (Johnson et al. 2011; Yamamoto et al. 2017) and their simulated counterparts (Kirkpatrick, Gelatt, and Vecchi 1983; Clements et al. 2017; Tiunov, Ulanov, and Lvovsky 2019). However, the effectiveness of general algorithms is dependent on the problem being considered, and high levels of performance often require extensive tailoring and domain-specific knowledge. Machine learning offers a route to addressing these challenges, which led to the demonstration of a meta-algorithm, S2V-DQN (Khalil et al. 2017), that utilises reinforcement learning (RL) and a deep graph network to automatically learn good heuristics for various combinatorial problems.

A solution to a combinatorial problem defined on a graph consists of a subset of vertices that satisfies the desired optimality criteria. Approaches following S2V-DQN's framework incrementally construct solutions one element at a time – reducing the problem to predicting the value of adding any vertex not currently in the solution to this subset. However, due to the inherent complexity of many combinatorial problems, learning a policy that directly produces a single, optimal solution is often impractical, as evidenced by the sub-optimal performance of such approaches. Instead, we propose that a natural reformulation is for the agent to explore the solution space at test time, rather than producing only a single "best-guess". Concretely, this means the agent can add or remove vertices from the solution subset and is tasked with searching for ever-improving solutions at test time. In this work we present ECO-DQN (Exploratory Combinatorial Optimization DQN), a framework combining RL and deep graph networks to realise this approach.

Our experimental work considers the Maximum Cut (Max-Cut) problem as it is a fundamental combinatorial challenge – in fact over half of the 21 NP-complete problems enumerated in Karp's seminal work (Karp 1972) can be reduced to Max-Cut – with numerous real-world applications. The framework we propose can, however, be readily applied to any graph-based combinatorial problem where solutions correspond to a subset of vertices and the goal is to optimize some objective function.

By comparing ECO-DQN to S2V-DQN as a baseline, we demonstrate that our approach improves on the state-of-the-art for applying RL to the Max-Cut problem. Suitable ablations show that this performance gap is dependent on both allowing the agent to reverse its earlier decisions and providing suitable information and rewards to exploit this free-

dom. Moreover, ECO-DQN can be initialised in any state (i.e. will look to improve on any proposed solution) and, as a consequence, has the flexibility to be either deployed independently or combined with other search heuristics. For example, we achieve significant performance improvements by simply taking the best solution found across multiple randomly initialised episodes. ECO-DQN also generalises well to graphs from unseen distributions. We obtain very strong performance on known benchmarks of up to 2000 vertices, even when the agent is trained on graphs an order of magnitude smaller and with a different structure.

Related Work

A formative demonstration of neural networks for combinatorial optimization (CO) was the application of Hopfield networks to the Travelling Salesman Problem (TSP) by Hopfield and Tank (1985). They mapped N-city problems to $N \times N$ graphs (networks) with each vertex, v_{ij}, a binary variable denoting whether city i is the j-th to be visited, and the edges connecting vertices proportional to the distance between cities. Although the results of Hopfield and Tank were contested by Wilson and Pawley (1988), there followed a period active research into neural networks for CO that lasted for over a decade (Smith 1999). During this time, RL techniques were first by applied to CO by Zhang and Diettench (1995), who considerer the NP-hard job-shop problem (of which the TSP is a specific case).

More recently, Bello *et al.* (2016) used policy gradients to train pointer networks (Vinyals, Fortunato, and Jaitly 2015), a recurrent architecture that produces a softmax attention mechanism (a "pointer") to select a member of the input sequence as an output. However, this architecture did not reflect the structure of problems defined over a graph, which Khalil *et al.* (2017) addressed with S2V-DQN, a general RL-based framework for CO that uses a combined graph embedding network and deep Q-network. Mittal *et al.* (2019) developed these ideas further by modifying the training process: first training an embedding graph convolution network (GCN), and then training a Q-network to predict the vertex (action) values. This is orthogonal to our proposal which considers the framework itself, rather than the training procedure, and, in principle, appears to be compatible with ECO-DQN.

Another current direction is applying graph networks for CO in combination with a tree search. Li *et al.* (2018) combined a GCN with a guided tree-search in a supervised setting, i.e. requiring large numbers of pre-solved instances for training. Very recently, Abe *et al.* (2019) trained a GCN using Monte-Carlo tree search as a policy improvement operator, in a similar manner to AlphaGo Zero (Silver et al. 2017), however, this work does not consider the Max-Cut problem.

2 Background

Max-Cut Problem

The Max-Cut problem is to find a subset of vertices on a graph that maximises the total number of edges connecting vertices within this subset to vertices not in this subset (the cut value). In this work we consider the more general weighted version of this problem, where each edge in the graph is assigned a weight and the objective is to maximise the total value of cut edges. Formally, for a graph, $G(V, W)$, with vertices V connected by edges W, the Max-Cut problem is to find the subset of vertices $S \subset V$ that maximises $C(S, G) = \sum_{i \subset S, j \subset V \setminus S} w_{ij}$ where $w_{ij} \in W$ is the weight of the edge connecting vertices i and j.

This is not simply a mathematical challenge as many real world applications can be reduced to the Max-Cut problem, including protein folding (Perdomo-Ortiz et al. 2012), investment portfolio optimization (Elsokkary et al. 2017; Venturelli and Kondratyev 2018) (specifically using the Markowitz (1952) formulation), and finding the ground state of the Ising Hamiltonian in physics (Barahona 1982).

Q-learning

As is standard for RL, we consider the optimization task as a Markov decision process (MDP) defined by the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ the reward function and $\gamma \in [0, 1]$ is the discount factor. A policy, $\pi : \mathcal{S} \to [0, 1]$, maps a state to a probability distribution over actions. The Q-value of a given state-action pair, $(s \in \mathcal{S}, a \in \mathcal{A})$, is then given by the discounted sum of immediate and future rewards

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t) \;\middle|\; s_0 = s, a_0 = a, \pi\right], \quad (1)$$

where s_0 and a_0 correspond to the initial state and action taken, with future actions chosen according to the policy, π.

A deep Q-network (Mnih et al. 2015) (DQN) provides a function $Q(s, a; \theta)$, where θ parameterises the network, which is trained to approximate $Q^*(s, a) \equiv \max_{\pi} Q^{\pi}(s, a)$, the Q-values of each state-action pair when following the optimal policy. Once trained, an approximation of the optimal policy can be obtained simply by acting greedily with respect to the predicted Q-values, $\pi(s; \theta) = \text{argmax}_{a'} Q(s, a'; \theta)$.

Message Passing Neural Networks

Our choice of deep Q-network is a message passing neural network (MPNN) (Gilmer et al. 2017). This is a general framework of which many common graph networks are specific implementations. Each vertex in the graph, $v \in V$, is represented with an n-dimensional embedding, μ_v^k, where k labels the current iteration (network layer). These are initialised, by some function I, from an input vector of observations, x_v, as $\mu_v^0 = I(x_v)$. During the message-passing phase, the embeddings are repeatedly updated with information from neighbouring vertices, $N(v)$, according to

$$m_v^{k+1} = M_k\big(\mu_v^k, \{\mu_u^k\}_{u \in N(v)}, \{w_{uv}\}_{u \in N(v)}\big), \quad (2)$$

$$\mu_v^{k+1} = U_k\big(\mu_v^k, m_v^{k+1}\big), \quad (3)$$

where M_k and U_k are message and update functions, respectively. After K rounds of message passing, a prediction is produced by some readout function, R. In our case this prediction is the set of Q-values of the actions corresponding to "flipping" each vertex, i.e. adding or removing it from the solution subset S, $\{Q_v\}_{v \in V} = R(\{\mu_u^K\}_{u \in V})$.

3 Exploiting Exploration

One straightforward application of Q-learning to CO over a graph is to attempt to directly learn the utility of adding any given vertex to the solution subset. This formalism, which is followed by S2V-DQN and related works, incrementally constructs a solution by adding one vertex at a time to the subset, until no further improvement can be made. However, the complexity of NP-hard combinatorial problems means it is challenging to learn a single function approximation of $Q^*(s, a)$ that generalises across the vast number of possible graphs. Therefore, as vertices can only be added to the solution set, policies derived from the learnt Q-values, such as a typical greedy policy, will likely be sub-optimal.

In this work we present an alternative approach where the agent is trained to explore the solution space at test time, seeking ever-improving states. As such, the Q-value of either adding or removing a vertex from the solution is continually re-evaluated in the context of the episode's history. Additionally, as all actions can be reversed, the challenge of predicting the true value of a vertex "flip" does not necessarily result in sub-optimal performance. The fundamental change distinguishing our approach, ECO-DQN, from previous works can then be summarised as follows: *instead of learning to construct a single good solution, learn to explore for improving solutions*.

However, simply allowing for revisiting the previously flipped vertices does not automatically improve performance. The agent is not immediately able to make more informed decisions, nor can it reach previously unobtainable solutions. Instead, further modifications are required to leverage this freedom for improved performance, which we now discuss.

Reward Shaping. The objective of our exploring agent is to find the best solution (highest cut-value) at any point within an episode. Formally, the reward at state $s_t \in \mathcal{S}$ is given by $\mathcal{R}(s_t) = \max(C(s_t) - C(s^*),\, 0)/|V|$, where $s^* \in \mathcal{S}$ is the state corresponding to the highest cut value previously seen within the episode, $C(s^*)$ (note that we implicitly assume the graph, G, and solution subset, S, to be included in the state). As continued exploration is desired, even after a good solution is found, there is no punishment if a chosen action reduces the cut-value. The reward is normalised by the total number of vertices, $|V|$, to mitigate the impact of different reward scales across different graph sizes. We use a discount factor of $\gamma = 0.95$ to ensure the agent actively pursues rewards within a finite time horizon.

As our environment only provides a reward when a new best solution is found, after an initial period of exploration, these extrinsic rewards can be sparse, or absent, for the remainder of the episode. We therefore also provide a small intermediate reward of $1/|V|$ whenever the agent reaches a locally optimal state (one where no action will immediately increase the cut value) previously unseen within the episode. In addition to mitigating the effect of sparse extrinsic rewards, these intrinsic rewards also shape the exploratory behaviour at test time. There are far more states than could be visited within our finite episodes, the vast majority of which are significantly sub-optimal, and so it is useful to focus on a subset of states known to include the global optimum. As local optima in combinatorial problems are typically close to each other, the agent learns to "hop" between nearby local optima, thereby performing a in-depth local search of the most promising subspace of the state space (see figure 2c).

Observations A Q-value for flipping each vertex is calculated using seven observations, ($x_v \in \mathbb{R}^7$), derived from the current state, with a state corresponding to both the target graph, $G(V, W)$, and the current subset of vertices assigned to the solution set, $S \subset V$. These observations are:

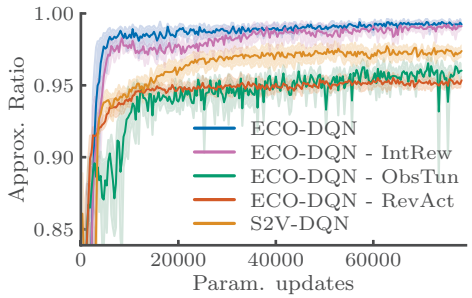
1. Vertex state, i.e. if v is currently in the solution set, S.
2. Immediate cut change if vertex state is changed.
3. Steps since the vertex state was last changed.
4. Difference of current cut-value from the best observed.
5. Distance of current solution set from the best observed.
6. Number of available actions that immediately increase the cut-value.
7. Steps remaining in the episode.

Observations (1-3) are local, which is to say they can be different for each vertex considered, whereas (4-7) are global, describing the overall state of the graph and the context of the episode. The general purposes of each of the observations are: (1-2) provide useful information for determining the value of selecting an action; (3) provides a simple history to prevent short looping trajectories; (4-6) ensure the extrinsic and intrinsic rewards are Markovian; and (7) accounts for the finite episode duration.

Experiments

Experimental details. In this work we train and test the agent on both Erdős-Rényi (Erdős and Rényi 1960) and Barabasi-Albert (Albert and Barabási 2002) graphs with edges $w_{ij} \in \{0, \pm 1\}$, which we refer to as ER and BA graphs, respectively. Training is performed on randomly generated graphs from either distribution, with each episode considering a freshly generated instance. The performance over training (i.e. all learning curves) is evaluated as the mean of a fixed set of 50 held-out graphs from the same distribution. Once trained, the agents are tested on a separate set of 100 held-out validation graphs from a given distribution.

During training and testing, every action taken demarks a timestep, t. For agents that are allowed to take the same action multiple times (i.e. ECO-DQN and selected ablations), which for convenience we will refer to as *reversible* agents, the episode lengths are set to twice the number of vertices in the graph, $t = 1, 2, \ldots, 2|V|$. Each episode for such agents is initialised with a random subset of vertices in the solution set. By contrast, agents that can only add vertices to the solution set (*irreversible* agents, i.e. S2V-DQN and selected ablations) are initialised with an empty solution subset. These agents keep selecting actions greedily even if no positive Q-values are available until $t = |V|$, to account for possible incorrect predictions of the Q-values. In both cases the best solution obtained at any timestep within the episode is taken as the final result. To facilitate direct comparison,



(a) Learning curves: ER graphs with $|V| = 40$

Agent	$\|V\|$=20	$\|V\|$=40	$\|V\|$=60	$\|V\|$=100	$\|V\|$=200
ECO-DQN	$0.97^{+0.03}_{-0.03}$	$1.00^{+0.00}_{-0.00}$	$0.99^{+0.01}_{-0.01}$	$0.99^{+0.01}_{-0.01}$	$0.98^{+0.01}_{-0.01}$
S2V-DQN	$0.97^{+0.03}_{-0.03}$	$0.98^{+0.01}_{-0.02}$	$0.98^{+0.01}_{-0.02}$	$0.92^{+0.02}_{-0.02}$	$0.95^{+0.02}_{-0.02}$
MCA-irrev	$0.89^{+0.06}_{-0.11}$	$0.89^{+0.04}_{-0.05}$	$0.87^{+0.05}_{-0.05}$	$0.87^{+0.03}_{-0.04}$	$0.86^{+0.03}_{-0.03}$

(b) Single episode performance: ER graphs

Agent	$\|V\|$=20	$\|V\|$=40	$\|V\|$=60	$\|V\|$=100	$\|V\|$=200
ECO-DQN	$0.99^{+0.01}_{-0.01}$	$0.99^{+0.01}_{-0.01}$	$0.98^{+0.00}_{-0.02}$	$0.97^{+0.02}_{-0.03}$	$0.93^{+0.02}_{-0.03}$
S2V-DQN	$0.97^{+0.01}_{-0.03}$	$0.96^{+0.03}_{-0.04}$	$0.94^{+0.02}_{-0.04}$	$0.95^{+0.02}_{-0.03}$	$0.94^{+0.02}_{-0.02}$
MCA-irrev	$0.92^{+0.05}_{-0.08}$	$0.89^{+0.05}_{-0.06}$	$0.88^{+0.04}_{-0.05}$	$0.87^{+0.03}_{-0.04}$	$0.87^{+0.03}_{-0.03}$

(c) Single episode performance: BA graphs

Figure 1: Performance comparison of ECO-DQN and baselines. (a) Learning curves, averaged over 5 seeds, when training on 40-vertex ER graphs. (b-c) Approximation ratios for ER and BA graphs with different numbers of vertices, $|V|$. We report the mean approximation ratios over the 100 validation graphs, along with the distance to the upper and lower quartiles.

ECO-DQN and S2V-DQN are implemented with the same MPNN architecture, with details provided in the Appendix.

Benchmarking details We compare the performance of ECO-DQN to a leading RL-based heuristic, S2V-DQN. To interpret the performance gap, we also consider the following ablations, which together fully account for the differences between our approach and the baseline (ECO-DQN $\equiv$ S2V-DQN+RevAct+ObsTun+IntRew).

- *Reversible Actions* (RevAct): Whether the agent is allowed to flip a vertex more than once. For irreversible agents we follow S2V-DQN and use γ=1.
- *Observation Tuning* (ObsTun): Observations (2-7) from the list above that allow the agent to exploit having reversible actions. Also, in the absence of ObsTun, the rewards used are simply the (normalised) immediate change in cut value, $\mathcal{R}(s_t) = (C(s_t) - C(s_{t-1}))/|V|$, which is necessary as without observations (4-5) the ECO-DQN reward structure is non-Markovian.
- *Intermediate Rewards* (IntRew): Whether the agent is provided with the small intermediate rewards for reaching new locally optimal solutions.

As an additional benchmark we also implement the MaxCutApprox (MCA) algorithm. This is a greedy algorithm, choosing the action (vertex) that provides the greatest immediate increase in cut value until no further improvements can be made. We consider two modifications of MCA. The standard application, which we denote MCA-irrev, is irreversible and begins with an empty solution set. The alternative algorithm, MCA-rev, starts with a random solution set and allows reversible actions.

We use the approximation ratio – $C(s^*)/C(s_{\text{opt}})$, where $C(s_{\text{opt}})$ is the cut-value of the true optimum solution – of each approach as a metric of solution quality. Exact methods are intractable for many of the graph sizes we use, therefore we apply a battery of optimization approaches to each graph and take the best solution found by any of them as the "optimum" solution. Specifically, in addition to ECO-DQN, S2V-DQN and the MCA algorithms, we use CPLEX, an industry standard integer programming solver, and a pair of recently developed simulated annealing heuristics by Tiunov *et al.* (2019) and Leleu *et al.* (2019). Details of these implementations and a comparison of their efficacy can be found in the Supplemental Material.

Performance benchmarking. Figure 1a shows learning curves of agents trained on ER graphs of size $|V| = 40$, where it can be seen that ECO-DQN reaches a significantly higher average cut than S2V-DQN. Removing either reversible actions (RevAct) or the additional observations (ObsTun) reduces the performance below that of S2V-DQN, underlining our previous assertion that obtaining state-of-the-art performance requires not only that the agent be allowed to reverse its previous actions, but also that it be suitably informed and rewarded to do so effectively. Intermediate rewards (IntRew) are seen to speed up and stabilise training. They also result in a small performance improvement, however this effect becomes clearer when considering how the agents generalise to larger graphs (see figures 3a and 3b).

Figures 1b and 1c show the performance of agents trained and tested on graphs with up to 200 vertices. We see that ECO-DQN has superior performance across most considered graph sizes and structures. Both ECO-DQN and S2V-DQN have similar computational costs per action, with extended performance comparisons provided in the appendix.

Intra-episode behaviour. We now consider how this strong performance is achieved by examining the intra-episode behaviour of an agent trained and tested on 200-vertex ER graphs. The larger graph size is chosen as it provides greater scope for the agent to exhibit non-trivial behaviour. Figure 2a highlights the trajectories taken by the trained agent on graphs from the validation set. Whilst the overall trend is towards higher cut-values, the fluctuations show that the agent has learnt to search for improving solutions even when this requires sacrificing cut-value in the short-term. Further analysis of the agent's behaviour is presented in figures 2b and 2c which show the action preferences and the types of states visited, respectively, over the course of an optimization episode.

From figure 2b, we see that the fully trained agent regularly chooses actions that do not correspond to the greatest immediate increase in the cut-value (Non-Greedy), or even that decrease the cut value (Negative). Moreover, the agent

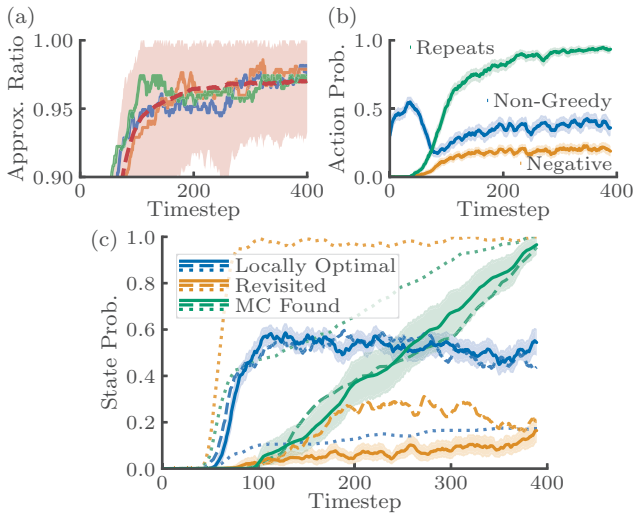


Figure 2: Intra-episode behaviour averaged across all 100 instances from the validation set for ER graphs with $|V| = 200$. (a) Mean (dashed) and range (shaded) of all trajectories, with three examples (solid) shown for reference. (b) The probability that the chosen action has already been taken within the episode (Repeats), does not provide the greatest immediate reward (Non-Greedy) or reduces the cut-value (Negative). (c) The probability that the current state is locally optimal (Locally Optimal), has already been visited within the episode (Revisited), and that the best solution that will be found within the episode has already been seen (MC found). The behaviour is shown at three points during training: when performance is equivalent to that of MCA-irrev (dotted) or S2V-DQN (dashed), and when fully trained (solid). (b-c) use a 10-step moving average over all graphs (trajectories) in the validation set, however, the shaded errors are only shown for the fully trained agent in (c).

also moves the same vertex in or out of the solution set multiple times within an episode (Repeats), which suggests the agent has learnt to explore multiple possible solutions that may be different from those obtained initially.

This is further emphasised in figure 2c where we see that, after an initial period of exploration, the agent searches through the solution space, repeatedly moving in and out of locally optimal (Locally Optimal) solutions whilst minimising the probability that it revisits states (Revisited). By comparing the agent at three points during training (fully trained and when performance level is equivalent to either MCA-irrev or S2V-DQN), we see that this behaviour is learnt. Weaker agents from earlier in training revisit the same states far more often, yet find fewer locally optimal states. The probability that the fully-trained agent has already found the best solution it will see in the episode (MC found) grows monotonically, implying that the agent finds ever better solutions while exploring. Indeed, for this agent, simply increasing the number of timesteps in an episode from $2|V|$=400 to $4|V|$ is seen to increase the average approximation ratio from $0.98^{+0.01}_{-0.01}$ to $0.99^{+0.01}_{-0.01}$. The range quoted for these approximation ratios corresponds to the upper and lower quartiles of the performance across all 100 validation graphs.

4 Leveraging Variance

Changing the initial subset of vertices selected to be in the solution set can result in very different trajectories over the course of an episode. An immediate result of this stochasticity is that performance can be further improved by running multiple episodes with a distribution of initialisations, and selecting the best result from across this set.

Experiments

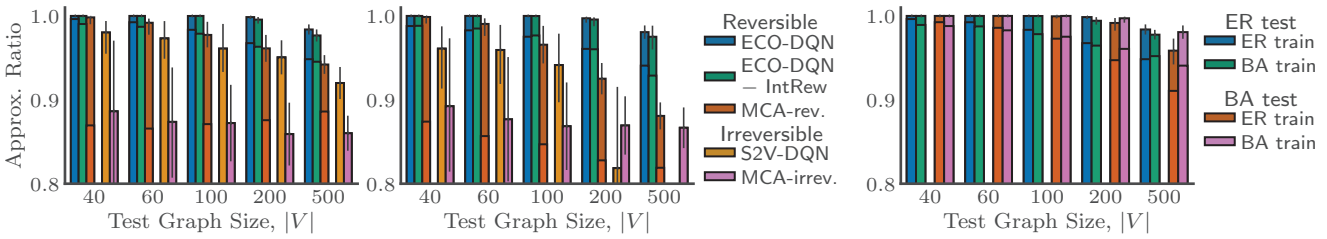
We optimize every graph using 50 randomly initialised episodes. At the same time, we make the task more challenging by testing on graphs that are larger, or that have a different structure, from those on which the agent was trained. This ability to generalise to unseen challenges is important for the real-world applicability of RL agents to combinatorial problems where the distribution of optimization tasks may be unknown or even change over time.

Generalisation to unseen graph types. Figures 3a and 3b show the generalisation of agents trained on 40 vertices to systems with up to 500 vertices for ER and BA graphs, respectively. (Generalisation data for agents trained on graphs of sizes ranging from $|V| = 20$ to $|V| = 200$ can be found in the Appendix.) ECO-DQN is compared to multiple benchmarks, with details provided in the caption, however there are three important observations to emphasise. Firstly, reversible agents outperform the irreversible benchmarks on all tests, with the performance gap widening with increasing graph size. This is particularly noticeable for BA graphs, for which the degrees of each vertex in the graph tend to be distributed over a greater range than for ER graphs, where S2V-DQN fails to generalise in any meaningful way to $|V|\geq$200.

Secondly, for the reversible agents it is clear that using multiple randomly initialised episodes provides a significant advantage. As ECO-DQN provides near-optimal solutions on small graphs within a single episode, it is only on larger graphs that this becomes relevant. However, it is noteworthy that even the simple MCA-rev algorithm, with only a relatively modest budget of 50 random initialisations, outperforms a highly trained irreversible heuristic (S2V-DQN). This further emphasises how stochasticity – which here is provided by the random episode initialisations and ensures many regions of the solution space are considered – is a powerful attribute when combined with local optimization.

Finally, we again observe the effect of small intermediate rewards (IntRew) for finding locally optimal solutions during training upon the final performance. For small graphs the agent performs near-optimally with or without this intrinsic motivation, however the difference becomes noticeable when generalising to larger graphs at test time.

In figure 3c we observe that ECO-DQN performs well across a range of graph structures, even if they were not represented during training, which is a highly desirable characteristic for practical CO. We train the agent on ER graphs with $|V|$=40 and then test it on BA graphs of up to $|V|$=500, and vice versa. The performance is marginally better when



(a) ER train and test (b) BA train and test (c) ER(BA) train, BA(ER) test

Figure 3: Generalisation of agents to unseen graph sizes and structures. (a-b) The performance of agents trained on ER and BA graphs of size $|V|$=40 tested on graphs of up to $|V|$=500 of the same type. ECO-DQN is shown with and without providing the intermediate rewards (IntRew) for finding locally optimal solutions during training, and is compared to the MCA-rev algorithm. Reversible agents are applied with 50 randomly initialised episodes to each of the 100 validation graphs for each type and size. The first marking on each bar is the average across every episode (the expected 'single-try' performance), with the upper limit extending to the average performance across different graphs. The vertical bars denote the 68 % confidence interval of this upper limit. Irreversible approaches are initialised with empty solution sets, and so only use 1 episode per graph. Note that S2V-DQN applied to BA graphs with $|V|$=500 is not visible on these axes but has a value of $0.49^{+0.12}_{-0.11}$. (c) A comparison of how agents trained on only one of either ER or BA graphs with $|V|$=40, perform on larger graphs from both distributions.

testing on graphs from the same distribution as the training data, however this difference is negligible for $|V| \leq 100$. Furthermore, in every case deploying ECO-DQN with 50 randomly initialised episodes outperforms all other benchmarks (S2V-DQN and MCA), even when only ECO-DQN is trained on different graph types to the test data.

Generalization to real-world datasets. Finally, we test ECO-DQN on publicly available datasets. The "Physics" dataset consists of ten graphs – with $|V|$=125, exactly 6 connections per vertex and $w_{ij} \in \{0, \pm 1\}$ – corresponding to Ising models of physical systems. The GSet is a well-investigated benchmark collection of graphs (Benlic and Hao 2013). We separately consider ten graphs, G1-G10, with $|V|$=800, and ten larger graphs, G22-G32, with $|V|$=2000. For G1-G10 we utilise 50 randomly initialised episodes per graph, however for G22-G32 we use only a single episode per graph, due to the increased computational cost. We apply agents trained on ER graphs with $|V|$=200. The results are summarised in table 1, where ECO-DQN is seen to significantly outperform other approaches, even when restricted to use only a single episode per graph.

Dataset	ECO-DQN	S2V-DQN	MCA-(rev, irrev)
Physics	1.000	0.928	0.879, 0.855
G1-10	0.996	0.950	0.947, 0.913
G22-32	0.971	0.919	0.883, 0.893

Table 1: Average performance on known benchmarks.

Despite the structure of graphs in the "Physics" dataset being distinct from the ER graphs on which the agent is trained, every instance in optimally solved. Averaged across all graphs, 37.6 % of episodes find an optimal solution and 90.4 % of these solutions are unique, demonstrating that, in conjunction with random initialisations, the agent is capable of finding many different optimal trajectories. Importantly, the structure of the GSet is distinct from that of the training data, with the first five instances in each tested set have only positive edges, $w_{ij} \in \{0, 1\}$.

5 Summary and Outlook

This work introduces ECO-DQN, a new state-of-the-art RL-based algorithm for the Max-Cut problem that generalises well to unseen graph sizes and structures. We show that treating CO as an ongoing exploratory exercise in surpassing the best observed solution is a powerful approach to this NP-hard problem. In principle, our approach is applicable to any combinatorial problem defined on a graph.

ECO-DQN can initialise a search from any valid state, opening the door to combining it with other search heuristics. We obtained further improved performance with a simple "heuristic" of randomly initialised episodes, however, one could consider combining ECO-DQN with more sophisticated episode initialisation policies. Alternatively, ECO-DQN could also be initialised with solutions found by other optimization methods to further strengthen them. Also, we train our agents with highly discounted future rewards (γ= 0.95), and although this is found to provide strong performance, the relatively short-term reward horizon likely limits exploration to only local regions of the solution space. As such, it would be interesting to investigate longer reward-horizons, particularly when training on larger graphs. A more substantial avenue to explore would be to use a recurrent architecture to learn a useful representation of the episode history, as opposed to the hand-crafted representation that we describe in section 3.

Whilst these are paths towards further developing exploration-based CO, we believe that the strong performance already demonstrated would allow our approach to be applied in numerous practical settings. This is especially true for settings where many graphs of similar structure need to be optimized, such as protein folding (Perdomo-Ortiz et al. 2012) and portfolio optimization (Elsokkary et al. 2017; Venturelli and Kondratyev 2018).

Test → / Train ↓	$\|V\|$=20	$\|V\|$=40	$\|V\|$=60	$\|V\|$=100	$\|V\|$=200	$\|V\|$=500	$\|V\|$=20	$\|V\|$=40	$\|V\|$=60	$\|V\|$=100	$\|V\|$=200	$\|V\|$=500
	ER graphs						BA graphs					
$\|V\|$=20	$0.99^{+0.01}_{-0.01}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.98^{+0.01}_{-0.01}$	$0.95^{+0.01}_{-0.01}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.99^{+0.01}_{-0.01}$	$0.98^{+0.01}_{-0.01}$
$\|V\|$=40	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.98^{+0.01}_{-0.01}$	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.98^{+0.01}_{-0.01}$
$\|V\|$=60	—	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.99^{+0.01}_{-0.01}$	—	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.99^{+0.01}_{-0.01}$
$\|V\|$=100	—	—	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	—	—	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	$0.98^{+0.01}_{-0.01}$
$\|V\|$=200	—	—	—	—	$1.00^{+0.00}_{-0.00}$	$1.00^{+0.00}_{-0.00}$	—	—	—	—	$0.99^{+0.01}_{-0.01}$	$0.98^{+0.01}_{-0.01}$

Table 2: Generalisation performance of ECO-DQN, using 50 randomly initialised episodes per graph.

Acknowledgements

The authors would like to thank D. Chermoshentsev and A. Boev. A.L.'s research is partially supported by Russian Science Foundation (19-71-10092).

Appendix

Code and graph availability. Source code, including experimental scripts and all testing and validation graphs, can be found at https://github.com/tomdbar/eco-dqn.

Graphs were generated with the NetworkX Python package (Hagberg, Schult, and Swart 2008). For ER graphs, a connection probability of 0.15 is used. The BA graphs have an average degree of 4. To produce our target graphs we then randomly set all non-zero edges to ± 1.

MPNN architecture. The initial embedding for each vertex, v, is given by

$$\mu_v^0 = \mathrm{relu}(\theta_1 x_v) \tag{4}$$

where $x_v \in \mathbb{R}^m$ is the input vector of observations and $\theta_1 \in \mathbb{R}^{m \times n}$. We also learn embeddings describing the connections to each vertex v,

$$\xi_v = \mathrm{relu}\Big(\theta_3 \big[\tfrac{1}{|N(v)|} \sum_{u \in N(v)} \mathrm{relu}\big(\theta_2 [w_{uv}, x_u]\big), |N(v)|\big]\Big), \tag{5}$$

where $\theta_2 \in \mathbb{R}^{m+1 \times n-1}$, $\theta_3 \in \mathbb{R}^{n \times n}$ and square bracket denote concatenation. The embeddings at each vertex are then updated according to

$$m_v^{k+1} = \mathrm{relu}\Big(\theta_{4,k} \big[\tfrac{1}{|N(v)|} \sum_{u \in N(v)} w_{uv} \mu_u^k, \xi_v\big]\Big), \tag{6}$$

$$\mu_v^{k+1} = \mathrm{relu}\Big(\theta_{5,k} \big[\mu_v^k, m_v^{k+1}\big]\Big), \tag{7}$$

where $\{\theta_{4,k}, \theta_{5,k}\} \in \mathbb{R}^{2n \times n}$. After K rounds of message passing, the Q-value for a vertex is read out using the final embeddings across the entire graph,

$$Q_v = \theta_7 \Big[\mathrm{relu}\big(\theta_6 \tfrac{1}{|V|} \sum_{u \in V} \mu_u^K\big), \mu_v^K\Big], \tag{8}$$

with $\theta_6 \in \mathbb{R}^{n \times n}$ and $\theta_7 \in \mathbb{R}^{2n}$.

In this work we use n=64 dimensional embedding vectors, and have K=3 rounds of message passing. However, many different MPNN implementations can be used with good success. In general, what is important is that the network be capable of capturing relevant information about the local neighbourhood of a vertex.

Training details. The Q-learning algorithm for ECO-DQN is shown in algorithm 1. All agents are trained with a minibatch sizes of 64 and k=32 actions per step of gradient descent. The learning rate is 10^{-4} and the exploration rate is linearly decreased from ε=1 to ε=0.05 over the first $\sim$10 % of training. We use the same MPNN for both S2V-DQN and ECO-DQN. We verify that this network properly represents S2V-DQN by reproducing its performance on the 'Physics' dataset at the level reported in the original work by Khalil *et al.* (2017). During the training of an irreversible agent's Q-network, the predictions of the Q-values produced by the target network are clipped to be strictly non-negative. This clipping is also used by Khalil *et al.* and is empirically observed to improve and stabilise training.

Algorithm 1: Q-learning for ECO-DQN

```
Initialize experience replay memory M.
for each episode do
    Sample a graph G(V,W) from distribution D
    Initialise a random solution set, S_0 ⊂ V
    for each step t in the episode do
        v_t = { choose random v' ∈ V,            with prob. ε
              { argmax_{v'∈V} Q(S_t, v'; θ),    otherwise
        S_{t+1} := { S_t ∪ {v_t},   if v_t ∉ S_t
                   { S_t \ {v_t},   if v_t ∈ S_t
        Add tuple (S_t, v_t, R_t, S_{t+1}) to M
        if t mod k = 0 then
            Sample minibatch B ~ M
            Update θ by SGD for B
        end
    end
end
return θ
```

Extended data. ECO-DQN's generalisation performance on ER and BA graphs is shown in table 2.

ECO-DQN and S2V-DQN have a similar computational

Graph size	ECO-DQN	MCA-(rev, irrev)
$\|V\|$=20	(0.29 ± 0.06) ms	(0.10 ± 0.02) ms
$\|V\|$=500	(6.64 ± 0.09) ms	(1.38 ± 0.06) ms

Table 3: Time per action for ECO-DQN and the greedy MCA algorithms across different graph sizes. Experiments were performed on NVIDIA Tesla M60 GPUs.

cost per time-step, which is largely determined by computing the MPNN-predicted Q-values, with the overall time to solution depending on the graph size and episode length. This performance is compared to the speed of the simple MCA agents in table 3.

Supplemental material. Supplemental Material can be found in the arXiv release (arxiv:1909.04063).

References

Abe, K.; Xu, Z.; Sato, I.; and Sugiyama, M. 2019. Solving NP-Hard Problems on Graphs by Reinforcement Learning without Domain Knowledge. *arXiv:1905.11623*.

Albert, R., and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74(1):47.

Banks, A.; Vincent, J.; and Anyakoha, C. 2008. A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing* 7(1):109–124.

Barahona, F. 1982. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General* 15(10):3241.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural Combinatorial Optimization with Reinforcement Learning. *arXiv:1611.09940*.

Benlic, U., and Hao, J.-K. 2013. Breakout Local Search for the Max-Cut problem. *Engineering Applications of Artificial Intelligence* 26(3):1162 – 1173.

Clements, W. R.; Renema, J. J.; Wen, Y. H.; Chrzanowski, H. M.; Kolthammer, W. S.; and Walmsley, I. A. 2017. Gaussian Optical Ising Machines. *Phys. Rev. A* 96:043850.

Dinur, I., and Safra, S. 2005. On the Hardness of Approximating Minimum Vertex Cover. *Annals of Mathematics* 439–485.

Elsokkary, N.; Khan, F. S.; La Torre, D.; Humble, T. S.; and Gottlieb, J. 2017. Financial Portfolio Management using D-Wave Quantum Optimizer: The Case of Abu Dhabi Securities Exchange. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).

Erdős, P., and Rényi, A. 1960. On the Evolution of Random Graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5(1):17–60.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1263–1272.

Goemans, M. X., and Williamson, D. P. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems using Semidefinite Programming. *Journal of the ACM (JACM)* 42(6):1115–1145.

Hagberg, A. A.; Schult, D. A.; and Swart, P. J. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *In Proceedings of the 7th Python in Science Conference*.

Hopfield, J. J., and Tank, D. W. 1985. "Neural" Computation of Decisions in Optimization Problems. *Biological Cybernetics* 52(3):141–152.

Johnson, M. W.; Amin, M. H.; Gildert, S.; Lanting, T.; Hamze, F.; Dickson, N.; Harris, R.; Berkley, A. J.; Johansson, J.; Bunyk, P.; et al. 2011. Quantum Annealing with Manufactured Spins. *Nature* 473(7346):194.

Karp, R. M. 1972. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*. Springer. 85–103.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *Advances in Neural Information Processing Systems*, 6348–6358.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by Simulated Annealing. *Science* 220(4598):671–680.

Leleu, T.; Yamamoto, Y.; McMahon, P. L.; and Aihara, K. 2019. Destabilization of local minima in analog spin systems by correction of amplitude heterogeneity. *Phys. Rev. Lett.* 122:040607.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In *Advances in Neural Information Processing Systems*, 539–548.

Markowitz, H. 1952. Portfolio Selection. *The Journal of Finance* 7(1):77–91.

Mittal, A.; Dhawan, A.; Medya, S.; Ranu, S.; and Singh, A. 2019. Learning Heuristics over Large Graphs via Deep Reinforcement Learning. *arXiv:1903.03332*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Papadimitriou, C. H. 1977. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4(3):237–244.

Perdomo-Ortiz, A.; Dickson, N.; Drew-Brook, M.; Rose, G.; and Aspuru-Guzik, A. 2012. Finding low-energy conformations of lattice protein models by quantum annealing. *Scientific Reports* 2:571.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of Go without Human Knowledge. *Nature* 550(7676):354.

Smith, K. A. 1999. Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research. *INFORMS Journal on Computing* 11(1):15–34.

Tiunov, E. S.; Ulanov, A. E.; and Lvovsky, A. 2019. Annealing by simulating the coherent Ising machine. *Optics Express* 27(7):10288–10295.

Venturelli, D., and Kondratyev, A. 2018. Reverse Quantum Annealing Approach to Portfolio Optimization Problems. *Quantum Machine Intelligence* 1–14.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems*, 2692–2700.

Williamson, D. P., and Shmoys, D. B. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.

Wilson, G., and Pawley, G. 1988. On the stability of the Travelling Salesman Problem algorithm of Hopfield and Tank. *Biological Cybernetics* 58(1):63–70.

Yamamoto, Y.; Aihara, K.; Leleu, T.; Kawarabayashi, K.-i.; Kako, S.; Fejer, M.; Inoue, K.; and Takesue, H. 2017. Coherent Ising machines—optical neural networks operating at the quantum limit. *npj Quantum Information* 3(1):49.

Zhang, W., and Dietterich, T. G. 1995. A Reinforcement Learning Approach to Job-Shop Scheduling. In *IJCAI*, volume 95, 1114–1120. Citeseer.