# Exploratory Combinatorial Optimization with Reinforcement Learning

**Thomas D. Barrett,**[1] **William R. Clements,**[2] **Jakob N. Foerster,**[3] **Alex 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@physics.ox.ac.uk, william.clements@indust.ai, jnf@fb.com, alex.lvovsky@physics.ox.ac.uk

**Abstract**

Many real-world problems can be reduced to combinatorial optimization on a graph. While such tasks are often NP-hard and analytically intractable, reinforcement learning (RL) has shown promise as a framework with which efficient heuristics methods to tackle these problems can be learned. Previous works construct the solution incrementally, adding one element at a time, however, this precludes the agent from revising its earlier decisions, which may be necessary for complex optimization tasks. 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 defined on a graph. Experimentally, our method produces state-of-the-art RL performance for 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.

## Introduction

NP-hard combinatorial optimization (CO) 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 approaches to CO are of great interest. However, as no known algorithms are able to solve NP-hard problems in polynomial time, exact methods rapidly become intractable. Instead, heuristics are often deployed that, despite offering no theoretical guarantees, can be chosen for high performance.

There are numerous heuristic methods – from search-based (Banks, Vincent, and Anyakoha 2008; Benlic and Hao 2013) to both simulated (Clements et al. 2017; Tiunov, Ulanov, and Lvovsky 2019) and physically-implemented annealing (Johnson et al. 2011; Yamamoto et al. 2017) – however, their effectiveness 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 combinatorial problems, learning a policy that directly produces a single, optimal solution is often impractical. Instead, we propose that the agent should 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.

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 Maximum Cut (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 freedom. Moreover, as ECO-DQN can be initialised in any state (i.e. will look to improve on any proposed solution) it can, in principle, be 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 unseen graphs. 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.

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

We consider the optimization task as a Markov decision process 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 is then $Q^{\pi}(s,a) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t)\big]$, where $s_0{=}s{\in}\mathcal{S}$ and $a{\in}\mathcal{A}$ is the first action taken with future actions chosen according to the policy, $\pi$. A deep Q-network (Mnih et al. 2015), with parameters $\theta$, provides a function $Q(s,a;\theta)$ which is trained to approximate $Q^*(s,a) \equiv \max_{\pi} Q^{\pi}(s,a)$, the Q-values 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.

Our choice of deep Q-network is a message passing neural network (MPNN) (Gilmer et al. 2017). Each vertex in the graph, $v{\in}V$, is represented with an embedding, $\mu_v^k{\in}\mathbb{R}^n$, 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

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

$$\mu_v^{k+1} = U_k\big(\mu_v^k, m_v^{k+1}\big), \qquad (2)$$

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 $S$, $\{Q_v\}_{v\in V} = R(\{\mu_u^K\}_{u\in V})$.

## ECO-DQN

A straightforward application of Q-learning to graph-based CO is to attempt to directly learn the value 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. However, the complexity of NP-hard combinatorial problems means it is challenging to learn a single function approximation of $Q^*(s,a)$ that generalises well across the many possible graphs, ultimately resulting in sub-optimal polices. Here, we present an alternative approach where the agent is trained to explore the solution space at test time. The fundamental change distinguishing our approach, ECO-DQN, from previous works can then be summarised as: *instead of learning to construct a single good solution, learn to explore for improving solutions*. ECO-DQN requires several novel features to explore effectively, which we now discuss.

*(i) Reversible actions:* ECO-DQN allows for any vertex in the graph to be added or removed from the solution set at every time-step. The objective of an exploring agent – to find the best solution (highest cut-value) at any point within an episode – is reflected in the reward structure 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 (note that we implicitly assume the graph, $G$, and solution subset, $S$, to be included in the state). The normalisation by the total number of vertices, $|V|$, mitigates 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.

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

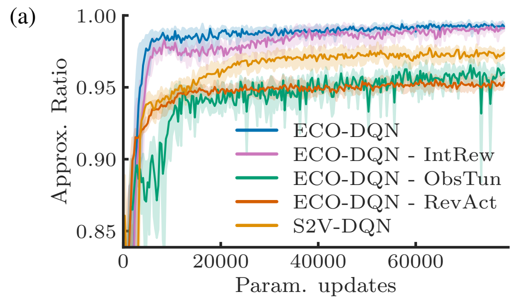
*(ii) Intermediate rewards:* As our environment only provides a reward when a new best solution is found, after an initial period of exploration, these extrinsic rewards can be sparse, or absent, for the remainder of the episode. We therefore also provide a small intermediate reward of $1/|V|$ whenever 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 effects 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, most 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 are typically close to each other, the agent learns to "hop" between nearby local optima, thereby performing an in-depth search of the most promising subspace of the state space (see figure 2c).

*(iii) Observation tuning:* A Q-value for flipping each vertex is calculated using seven observations derived from the current state ($x_v{\in}\mathbb{R}^7$). Three of these are local, which is to say they can be different for each vertex considered: (1) vertex state, i.e. if $v$ is currently in the solution set, $S$; (2) immediate cut change if vertex is "flipped"; (3) steps since vertex was last "flipped". The remaining global observations describe the state of the graph and the context of the episode: (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. 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; (7) accounts for the finite episode duration.

## Experiments

We train and test agents on Erdős-Rényi (Erdős and Rényi 1960) (ER) graphs with a connection probability of 0.15 and $w_{ij}{\in}\{0,\pm1\}$. Equivalent experiments on Barabasi-Albert (Albert and Barabási 2002) (BA) graphs are detailed in the Supplemental Material, with the results in both cases being qualitatively similar and supporting the same conclusions. Training is performed on random graphs with each episode considering a freshly generated instance. The performance over training (i.e. learning curves) is evaluated on a fixed set of 50 held-out graphs. Once trained, the agents are tested on a separate set of 100 held-out validation graphs.

Within an episode every action taken demarks a time-step and the best solution obtained at any time within the



(b)

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

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

episode is taken as the final result. Agents that are allowed to take the same action multiple times (*reversible* agents, e.g. ECO-DQN) have episode lengths set to twice the number of vertices in the graph, $2|V|$, and each episode 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, e.g. S2V-DQN) are initialised with an empty solution subset. These agents greedily select actions until no more are available as this will result in an equal or better policy than terminating the episode when all remaining actions are predicted to have negative Q-values. To facilitate direct comparison, ECO-DQN and S2V-DQN are implemented with the same MPNN architecture. Details are provided in the Supplemental Material, however, many different MPNN implementations capture relevant information about the local neighbourhood of a vertex and can be used with good success.

We compare ECO-DQN to S2V-DQN and individually ablate the novel features of ECO-DQN – reversible actions (RevAct), observation tuning (ObsTun) and intermediate rewards (IntRew) – which together fully account for the differences between these approaches (ECO-DQN $\equiv$ S2V-DQN + RevAct + ObsTun + IntRew). Irreversible agents follow S2V-DQN and use $\gamma$=1. In the absence of observation tuning (i.e. observations (2-7)), we modify the rewards to be $\mathcal{R}(s_t)=(C(s_t)-C(s_{t-1}))/|V|$, which is necessary as without observations (4-6) the ECO-DQN reward structure is non-Markovian. As additional benchmarks we also implement the MaxCutApprox algorithm in both the *reversible* and *irreversible* frameworks (MCA-rev and MCA-irrev, respectively). 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 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 graphs 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" (see Supplemental Material for details).

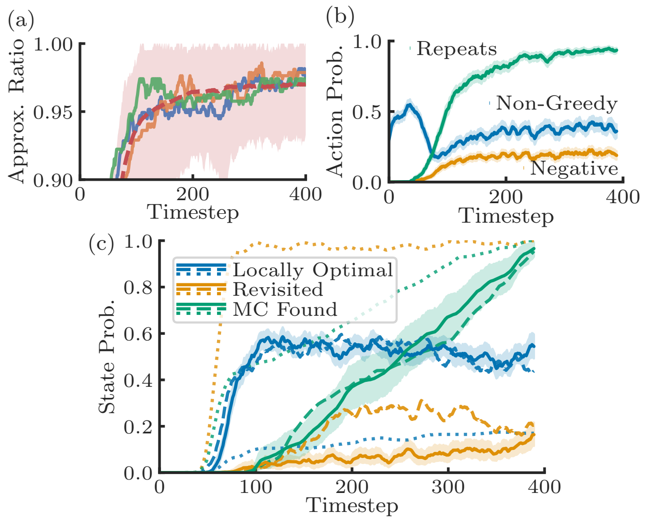
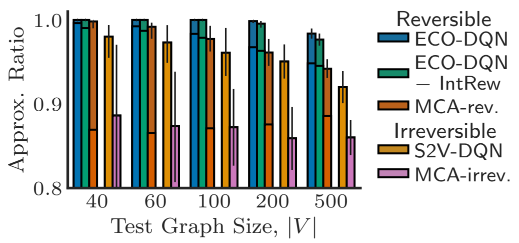


Figure 2: Intra-episode behaviour on 200-vertex graphs. (a) Mean (dashed) and range (shaded) of all trajectories, with three examples (solid) shown for reference. (b) The probability that the chosen action has already been taken within the episode (Repeats), does not provide the greatest immediate reward (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 agent's 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 100 graphs (trajectories) in the validation set.

**Single-episode optimization.** 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 that state-of-the-art performance requires an agent to not only be able to reverse previous actions, but also to 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 agents generalise to larger graphs (figure 3a). Figure 1b shows the performance of agents trained and tested on graphs with up to 200 vertices. We see that ECO-DQN has superior performance in all cases.

We now consider how this strong performance is achieved by examining the intra-episode behaviour of an agent trained

(a)



(b)

| Dataset ($|V|$) | ECO | S2V | MCA-(rev, irrev) |
|---|---|---|---|
| Physics (125) | 1.000 | 0.928 | 0.879, 0.855 |
| G1-G10 (800) | 0.996 | 0.950 | 0.947, 0.913 |
| G22-G32 (2000) | 0.971 | 0.919 | 0.883, 0.893 |

Figure 3: (a) The performance of agents trained on graphs with $|V|$=40 tested on graphs of up to $|V|$=500. Reversible agents use 50 randomly initialised episodes for each of the 100 validation graphs of a given size. The first marking on each bar is the average across every episode (the expected 'single-try' performance). The upper limit extends to the average performance across different graphs with error bars denoting the 68 % confidence interval. Irreversible approaches are initialised with empty solution sets, and so only use 1 episode per graph. (b) Average performance on known benchmarks of agents trained on ER graphs with $|V|$=200.

and tested on 200-vertex ER graphs. Figure 2a highlights 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. 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, vertices are moved 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 differ 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). We see that this behaviour is learnt by comparing the agent's behaviour at three points during training. Weaker agents from earlier in training revisit 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, simply increasing the number of time-steps in an episode from $2|V|$=400 to $4|V|$ increase the average approximation ratio from $0.98^{+0.01}_{-0.01}$ to $0.99^{+0.01}_{-0.01}$.

**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 different initialisations, and selecting the best result from across this set. As such, we now optimize every graph using 50 randomly initialised episodes. We also 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.

Figure 3a show the generalisation of agents trained on 40 vertices to graphs with up to 500 vertices. ECO-DQN is compared to multiple benchmarks, however there are three important observations to emphasise. Firstly, reversible agents outperform the irreversible benchmarks on all tests, with the performance gap increasing with graph size. Secondly, using multiple randomly initialised episodes provides a significant advantage. Indeed, even the simple MCA-rev algorithm, using only 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. Thirdly, we see that the small intermediate rewards (IntRew) provided during training improve performance, although, due to the near optimal performance on small graphs within a single optimisation episode, this only becomes noticeable when generalising to large graphs at test time.

Finally, we test ECO-DQN, trained on ER graphs with $|V|$=200, on publicly available datasets. The "Physics" dataset consists of ten graphs – with $|V|$=125, exactly 6 connections per vertex and $w_{ij}\in\{0,\pm1\}$ – 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. The results in figure 3b show that ECO-DQN significantly outperforms other approaches, even when restricted to use only a single episode per graph.

Despite the structure of graphs in the "Physics" dataset being distinct from the 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, $w_{ij}\in\{0,1\}$.

## Summary

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

## References

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.

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.

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.

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.

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.

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.

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

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.