Machine Learning: Science and Technology

Reinforcement learning enhanced quantum-inspired algorithm for combinatorial optimization

IOP Publishing Mach. Learn.: Sci. Technol. 2 (2021) 025009 https://doi.org/10.1088/2632-2153/abc328

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
23 June 2020

REVISED
2 October 2020

ACCEPTED FOR PUBLICATION
20 October 2020

PUBLISHED
29 December 2020

PAPER

Reinforcement learning enhanced quantum-inspired algorithm for combinatorial optimization

Dmitrii Beloborodov[1], A E Ulanov[1], Jakob N Foerster[2], Shimon Whiteson[2] and A I Lvovsky[1,3]

[1] Russian Quantum Center, Moscow, Russia
[2] Department of Computer Science, University of Oxford, Oxford, United Kingdom
[3] Department of Physics, University of Oxford, Oxford, United Kingdom

E-mail: dmitribeloborodov@yandex.ru

Keywords: combinatorial optimization, reinforcement learning, ising model, maximum cut

Abstract

Quantum hardware and quantum-inspired algorithms are becoming increasingly popular for combinatorial optimization. However, these algorithms may require careful hyperparameter tuning for each problem instance. We use a reinforcement learning agent in conjunction with a quantum-inspired algorithm to solve the Ising energy minimization problem, which is equivalent to the Maximum Cut problem. The agent controls the algorithm by tuning one of its parameters with the goal of improving recently seen solutions. We propose a new Rescaled Ranked Reward (R3) method that enables a stable single-player version of self-play training and helps the agent escape local optima. The training on any problem instance can be accelerated by applying transfer learning from an agent trained on randomly generated problems. Our approach allows sampling high quality solutions to the Ising problem with high probability and outperforms both baseline heuristics and a black-box hyperparameter optimization approach.

1. Introduction

Many important real-world combinatorial problems can be mapped to the Ising model, ranging from portfolio optimization (Venturelli and Kondratyev 2019, Marzec 2016) to protein folding (Perdomo-Ortiz *et al* 2012). The Ising model describes the pairwise interaction of binary particles and assigns some cost function (energy) to each particle configuration. The Ising problem consists in finding binary strings that minimize the energy. It is a quadratic unconstrained optimization task over the discrete $\{\pm 1\}^n$ domain and equivalent to the Max-Cut problem from graph theory.

There are multiple methods for solving the Ising and Max-Cut problems. Classic algorithms include heuristics performing local search in the solution space, like breakout local search (Benlic and Hao 2013) and simulated annealing (Kirkpatrick *et al* 1983). For many combinatorial problems, commercial solvers are available, including Gurobi (2019) and CPLEX (2019).

An entirely different approach is to use a quantum physical system with its energy function similar to the optimization objective, and then anneal this system towards its ground state—the lowest energy state. Devices utilizing this method include the coherent Ising machine (CIM) (Inagaki *et al* 2016, McMahon *et al* 2016) and the quantum superconducting annealer manufactured by D-Wave Systems (McGeoch *et al* 2019). For example, in CIM, pulses of light circulate in a lossy optical fiber loop containing a parametric amplifier. In each round trip, a classical controller modulates these pulses according to the parameters of the Ising problem and the measured amplitudes of other pulses.

Quantum technology does not yet compete with classical computation systems in terms of both problem size and solution quality. However, it has inspired a family of new classical optimisation algorithms that perform well in comparison with existing ones (King *et al* 2018, Leleu *et al* 2019). An example is a simulation of CIM, known as the SimCIM algorithm (Tiunov *et al* 2019). SimCIM reformulates the Ising model as a continuous constrained optimization problem and solves it with iterative gradient-based optimization, with each iteration corresponding to a roundtrip of the optical pulses through the fiber loop. SimCIM was

implemented on computers equipped with consumer GPUs and outperformed CIM in both solution quality and computation time (Tiunov *et al* 2019). It has since been applied as a Boltzmann sampler to train general Boltzmann machines and for applications in statistical physics (Ulanov *et al* 2019). However, both SimCIM and CIM require parameter tuning for each problem instance to obtain the best results. One of the SimCIM parameters, the combined coefficients of linear gain and loss (which can be interpreted as a dynamic regularization coefficient), needs to be varied as a function of the iteration number. As a result, the use of classic hyperparameter optimization approaches (Feurer and Hutter 2018) is limited, since most methods assume a small number of continuous or discrete parameters.

To automate parameter tuning in a flexible way, we use a reinforcement learning (RL) agent to control the regularization (gain-loss) function of SimCIM during the optimization process. An important feature of the Ising problem is the presence of multiple local optima whose energy is only slightly higher than the global minimum, but the associated bit configuration is significantly different. To address this issue, we propose Rescaled Ranked Reward (R3), a modification of Ranked Reward (R2) (Laterre *et al* 2018). In this approach, we assign reward to the agent depending on how its current score compares to scores obtained in recent trials, and thus enable self-play training for a single-player environment. Rescaled Ranked Reward ensures the agent is motivated to keep discovering better solutions, without destabilizing the training process.

We demonstrate that the convergence speed noticeably improves if we apply policy transfer from an agent pre-trained on randomly generated problems to the unseen target problem. This transfer learning is facilitated by feature-wise linear modulation (FiLM) (Dumoulin *et al* 2016) with the features extracted from the general parameters of the problem at hand.

Our approach allows us to find the best solutions with higher probability than SimCIM with a regularization function that changes linearly or according to a hyperbolic tangent function with manually tuned parameters (which is our benchmark for the human level performance). It also outperforms CMA-ES (Hansen *et al* 2003), one of the most powerful black-box algorithms for hyperparameter optimization[4].

2. Background

2.1. Combinatorial optimization

The Ising problem is to find the vector $\boldsymbol{x} \in \{\pm 1\}^n$ of n binary variables that minimize the 'energy' functional

$$H = -\frac{1}{2}\sum_{i,j} J_{i,j} x_i x_j. \tag{1}$$

In physics, the individual values x_i correspond to quantized electron spins in a lattice that interact with each other via magnetic field. The symmetric matrix J defines this pairwise interaction. This problem is NP-hard (Barahona 1982) and is equivalent to the Max-cut problem of dividing a set of n nodes of a weighted graph into two subsets, such that the sum $\mathcal{C}(J,\boldsymbol{x}) = \frac{1}{4}(\boldsymbol{x}^T J \boldsymbol{x} - \sum_{ij} J_{ij})$ of edge weights connecting these subsets is maximized. In this interpretation, the problem matrix J is the adjacency matrix of the graph, and binary variables $\boldsymbol{x}$ denote the choice of the subset for each node. The optimization objective $\mathcal{C}(J,\boldsymbol{x})$ is called the *cut value* (higher is better); in this paper we use it to evaluate our algorithm and compare it to benchmarks.

2.2. The coherent Ising machine

We now briefly recap the physics of the CIM. While we hope that some readers will find this description useful, it is not essential for the understanding of our RL algorithm.

The CIM is a quantum optical device specially designed to solve the problem of finding the ground state of the Ising Hamiltonian. It consists of an optical parametric amplifier (OPA) with its output fed back to the input via an optical fiber loop (figure 1). An essential feature of the OPA is that it coherently amplifies optical fields at phases 0 and π (with the phase reference determined by the pump field) while the signals with orthogonal phases are deamplified. The length of the feedback loop contains an integer number of intervals between pump pulses, which results in well-defined pulsed modes circulating in it. After multiple roundtrips, these modes, seeded by vacuum electromagnetic field fluctuations, are amplified to microscopic amplitudes with the phases tending to either $\varphi_i = 0$ or π. This growth is not infinite: at sufficiently high amplitudes, the OPA gain will saturate due to two-photon absorption.

A small fraction of each pulse exiting the OPA is directed to an optical homodyne detector which measures the amplitude (quadrature) c_i of the field corresponding to zero phase and the measurement result is stored for the duration of one roundtrip. Each (jth) pulse entering the OPA is then subjected to optical

[4] The code is available at https://github.com/BeloborodovDS/SIMCIM-RL.

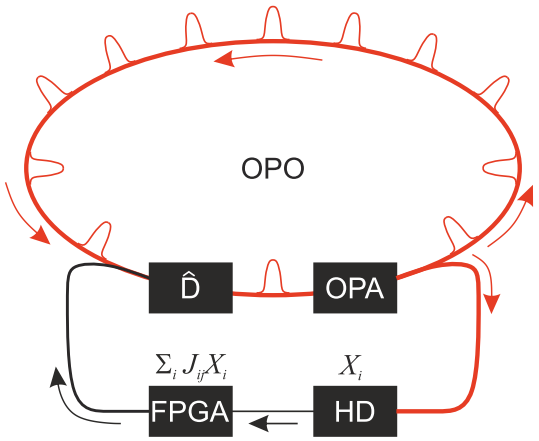

Figure 1. Concept of the CIM. OPA, OPO stand for optical parametric amplifier and oscillator, respectively, HD—homodyne detector, FPGA—field-programmable gated array, $\hat{D}$ — phase-space displacement.

phase-space displacement proportional to

$$F_j = \sum_{i \neq j} J_{ij} c_i, \tag{2}$$

where c_i are the measurement results obtained from all other pulses. This procedure breaks the symmetry between the phases of 0 or π acquired by each pulsed mode as a result of the amplification, such that the sequence $\varphi_i = 0$ or π of the phases, interpreted as spin values $x_i = \pm 1$, respectively, corresponds to the solution of the Ising problem for a given matrix J.

This can be intuitively understood as follows. If the spins x_i in the Ising energy are replaced by continuous variables c_i, the gradient $\partial H/\partial c_j$ can be interpreted as the force vector that is directed towards the energy minimum. But the displacement value F_j that we apply to our pulses is exactly proportional to that gradient. This displacement is therefore equivalent to applying a force that drives the system towards the energy minimum.

2.3. SimCIM algorithm

SimCIM (Tiunov *et al* 2019) is a numerical algorithm simulating by the physics of the CIM. In this algorithm, we characterize each pulse by its continuous amplitude c_j and take into account the following effects that modify this amplitude during the pulse's roundtrip through the system.

- The linear gain inside the OPA and a subsequent loss inside the fiber loop multiply the amplitude by a constant factor.
- The displacement signal (2) is added to it.
- Quantum noise is added, which is assumed to be a sample from a normal distribution.
- The OPA gain is lower for high amplitudes due to the amplifier saturation, which prevents infinite growth of the amplitude. In SimCIM, this is modeled by restricting each amplitude to the interval $[-1,+1]$.

For each roundtrip t, we therefore compute the vector

$$\boldsymbol{g}_t = \mu(J\boldsymbol{c}_t - p_t \boldsymbol{c}_t) + \sigma \boldsymbol{\varepsilon}_t, \tag{3}$$

where $\boldsymbol{c} = \{c_j\}_{j=1}^n$ is the vector comprised by the amplitudes, μ is a constant proportionality coefficient, $\boldsymbol{\varepsilon}_t$ is a vector of random samples from the standard normal distribution and σ the noise amplitude. The scalar p_t defines the combined effects of gain and loss; its value is allowed to change as a function of t because one can control the pump power. Then we calculate the update vector $\boldsymbol{m}_t = \eta \boldsymbol{m}_{t-1} + (1-\eta)\boldsymbol{g}_t$ by applying momentum η and update the amplitude vector as follows:

$$\boldsymbol{c}_{t+1} = \boldsymbol{c}_t + \boldsymbol{m}_t \odot \mathbb{I}[|\boldsymbol{c}_t + \boldsymbol{m}_t| \leq 1] \tag{4}$$

where $\mathbb{I}$ denotes the indicator function and $\odot$ element-wise product[5]. In other words, an update is applied to each element of $\boldsymbol{c}_t$ only if it does not cause it to to exceed the boundary of $[-1, 1]$. Both the amplitude vector and momentum vector are initialized to zero at $t = 1$.

These iterations are repeated N times. Subsequently, the solution of the original discrete problem (1) is calculated as its elementwise *sign*. SimCIM is reminiscent of the Hopfield-Tank simulated annealer (Hopfield and Tank 1986), but differs from it in the shape of the activation function.

SimCIM can be thought of a method to solve the optimization problem

$$\begin{cases} L = -\boldsymbol{c}^T J \boldsymbol{c} + p\boldsymbol{c}^T \boldsymbol{c} \to \min_{\boldsymbol{c}}, \\ J \in \mathbb{R}^{n \times n},\ J^T = J,\ \boldsymbol{c} \in [-1, 1]^n \end{cases} \tag{5}$$

The vector $\boldsymbol{g}_t$ in equation (3) can then be interpreted as the antigradient of the loss function L, the constant μ as the learning rate and the quantity p_t as the regularization coefficient. The hyperparameters μ, η, σ are scalar values and relatively easy to tune. In contrast, p_t is a discretized function of time, which poses a challenge to common hyperparameter optimization techniques due to the large dimensionality.

2.3.1. Eigenvalue decomposition

Since the matrix J is real and symmetric, we can construct an eigenvalue decomposition $J = Q\Lambda Q^T$, where Q is an orthogonal matrix with the eigenvectors of matrix J as its columns, and Λ is a diagonal matrix with the eigenvalues of J as diagonal elements Λ_{ii}.

With some simplifications ($\eta = \sigma = 0,\ \boldsymbol{c}_t \ll \mathbf{1}$) the dynamics (4) of the system can be described by the equation $\boldsymbol{c}_{t+1} = \boldsymbol{c}_t + \mu(J\boldsymbol{c}_t - p_t\boldsymbol{c}_t)$. By performing eigenvalue decomposition and the change of variable $\boldsymbol{e} = Q^T\boldsymbol{c}$, the update equation simplifies to $e_{t+1,i} = e_{t,i} + \mu(\Lambda_{ii}e_{t,i} - p_t e_{t,i})$, i.e. the update is applied to individual elements of the vector $\boldsymbol{e}$. Thus, when p_t is greater than the highest eigenvalue of J, both $\boldsymbol{e}$ and $\boldsymbol{c} = Q\boldsymbol{e}$ exponentially decay. Also, setting p_t lower than all Λ_{ii} leads to exponential growth of all amplitudes e_i, and subsequent poor conversion of the iterations. Using these observations, we reparameterize the regularization function by introducing a normalized regularization function $\bar{p}_t$, which, as a rule, is restricted to the interval $[0, 1]$:

$$p_t = \bar{p}_t \left(\max_i \Lambda_{ii} - \min_i \Lambda_{ii} \right) + \min_i \Lambda_{ii}. \tag{6}$$

2.3.2. Choosing a learning rate

To select the learning rate μ for each problem instance, we use an automatic procedure similar to the learning rate range test proposed in Smith (2017). One optimization cycle of SimCIM (4) with momentum $\eta = 0$ is performed with an exponentially decaying μ, starting from some high value. The learning rate is chosen at an iteration where the l_1 norm of gradient $\|\boldsymbol{g}_t\|_1$ starts to converge.

2.4. Reinforcement learning

In RL, an agent at each step t interacts with some environment $\mathcal{E}$ by observing its state s_t, performing an action a_t sampled from its policy $\pi(a|s)$, and obtaining a reward $r_t(s_t, a_t, s_{t+1})$. One interaction session, called an episode, usually lasts until the agent reaches a terminal state or until the limit on the number of steps T is reached. The goal of the agent is to maximize the expected sum of discounted rewards during the episode: $\mathbb{E}_{\tau(\pi)} \sum_{t=0}^{T} \gamma^t r_t(s_t, a_t, s_{t+1})$, where $\tau(\pi)$ is a trajectory generated by the agent in the environment, and $\gamma \in [0, 1)$ is the discount factor.

In actor-critic learning, an agent consists of two components. The actor, using observations from the environment, predicts the agent's action a according to its policy $\pi(a|s)$. The actor's parameters are updated in the direction of improvement, which is estimated using sampled trajectories. The critic predicts the value of each observation, which is then used to reduce the variance of actor's gradient. In the case of deep RL, both actor and critic are usually implemented as deep neural networks.

3. Our approach

Observations, actions, and rewards from each SimCIM optimization cycle constitute one agent rollout. Every m iterations of SimCIM, the neural network based RL agent observes the state of the optimization process

[5] The update (4) is slighly different from that originally published in Tiunov *et al* (2019), where the vector $\boldsymbol{c}$ was clipped to within $[-1, 1]$ after adding the update vector. We found empirically that this approach injects additional randomness into the optimisation process and reduces the chances of getting stuck in local optima.

and chooses the action that determines p_t in the next m SimCIM iterations. In an episode of N SimCIM steps, the agent performs N/m actions. The calculation of a single SimCIM rollout is summarized in algorithm 1.

3.1. Actions

The agent chooses one of the three discrete actions: $a \in \{-1, 0, 1\}$. The value of $\bar{p}_t$ during each of the next m SimCIM iterations is modified by $a\frac{p_\Delta}{m}$, where p_Δ is a hyperparameter. In addition, $\bar{p}_t$ is decreased by $\frac{1}{N}$ at each iteration, so $a \equiv 0$ corresponds to a linear decrease of $\bar{p}_t$ from 1 to 0 during a rollout. The value of $\bar{p}_t$ is clipped to the interval $[0, 1.05]$ to limit the exploration area.

Algorithm 1 SimCIM rollout

Initialize $\bar{p}_0 = 1.0$
for $t = 0$ **to** $N - 1$ **step** m **do**
  if $t \mod m = 0$ **then**
    Use the agent to decide on action $a \in \{-1, 0, 1\}$;
  end if
  $\bar{p}_{t+1} := \bar{p}_t - 1/N + a\frac{p_\Delta}{m}$;
  clip $\bar{p}_{t+1}$ to $[0, 1.05]$;
  Use equation (6) to calculate p_{t+1} from $\bar{p}_{t+1}$;
  Apply a SimCIM iteration via equation (4) using p_{t+1};
end for

3.2. Observations

The agent observes the current state of optimization variables in the eigenbasis of the problem matrix J, i.e. it is supplied with the set of amplitudes $\boldsymbol{e}_{t,i}$ (listed in the order of decreasing corresponding eigenvalues Λ_{ii}), as well as the elapsed time t/N and the regularization function $\bar{p}_{t-1}$ from the previous step. The benefit of using $\boldsymbol{e}_t$, rather than the actual amplitudes $\boldsymbol{c}_t$, as the state component, is that the former have a natural ordering according to the corresponding eigenvalues of J, while the components of $\boldsymbol{c}_t$ can be arbitrary permuted along with the rows and columns of J. This representation of the state therefore facilitates the transferability of the agent across problems.

To provide the agent with the information about the current problem instance for the purpose of transfer learning, we calculate problem features as $\phi_j = \frac{1}{n}\sum_{i=1}^{n} |Q_{ij}|$. This means that ϕ_j are scaled l_1 norms of the problem matrix eigenvectors. These features are static observations that are fixed during the entire episode. Features ϕ_j are provided to the agent at each step as a part of the observation.

3.3. Rewards

In the case of combinatorial optimization, we are interested in finding solutions with the best quality (e.g. cut value) for each instance, while the path in which it has been reached is less important. Also, solutions with slightly different cut values may correspond to completely different bit configurations $\boldsymbol{x}$. Thus the current cut value or its difference between steps is not the best choice for the reward.

To address this issue, the Ranked Reward (R2) method was proposed in Laterre *et al* (2018). In R2, the environment maintains a list of discovered cut values C_j for the last P episodes (a 'leaderboard'), the q-th percentile C^q is calculated over this list, and the new solution with the cut value C is rewarded at the last step only according to the rule

$$r_{R2} = \begin{cases} +1, & C > C^q \\ -1, & C < C^q \\ \pm 1 \text{ randomly}, & C = C^q \end{cases}, \tag{7}$$

where q and P are hyperparameters. This scheme implies that the reward depends not only on the agent's performance in the current episode, but also in P previous ones: rewards for the same state-action pair may differ significantly in the beginning and the end of the training process. Such artificially non-stationary nature of the environment appears to be a natural approach in settings with unknown best result, where the only reliable way to measure the agent's performance is to compare it to the performance of other agents. Another application of this approach can be found in multi-player self-play RL (Silver *et al* 2017). This kind of reward ensures that the agent constantly improves its performance in search of better solutions. In the language of self-play, the agent is rewarded for beating most of its last results in a single-player game (being at the top of the leaderboard) and punished otherwise.

We propose a modification of this method that we dub Rescaled Ranked Rewards (R3) to account for imbalanced reward distributions:

$$r_{R3} = \begin{cases} +\frac{q}{100}, & C > C^q \\ -(1 - \frac{q}{100}), & C < C^q \\ \bar{r}, & C = C^q \end{cases}, \tag{8}$$

where $\bar{r}$ is calculated in such a way that the average reward over the last P episodes is equal to zero. This modification ensures that negative and positive rewards are balanced. It also ensures that solutions with $C > C^q$ are clearly distinguishable from those with $C = C^q$, and hence discourages the agent from getting stuck in a local optimum.

The reward calculation and agent update are outlined in algorithm 2. The batching logic is omitted for simplicity, however this procedure can be generalized for a batch of agents to improve efficiency.

Algorithm 2 Reward calculation and agent update

Initialize an empty queue $\mathcal{L}$ of maximum size **P**;
for *episode* = 1 **to** *numEpisodes* **do**
  Calculate one rollout of SimCIM controlled by the agent using algorithm 1;
  Calculate cut value C;
  Push C into $\mathcal{L}$;
  Calculate C_q as the q-th percentile of values in $\mathcal{L}$;
  Calculate reward for the last step according to equation (8);
  Update the agent according to PPO algorithm (Schulman *et al* 2017);
end for

3.4. Transfer learning

The approach we propose requires training the agent for each problem instance separately. However it is possible to accelerate this process significantly by pre-training the agent on randomly generated problem instances.

We pre-train the agent on random adjacency matrices from the Erdös–Rényi distribution (Erdös and Rényi 1960) with a fixed connection probability of 0.06. We select this value so that the pre-training distribution is close to that for the target set of problems. However, we observe that transfer works reliably for matrices with different structure, too.

At each step of the pre-training process, the environment samples a new matrix J, and the agent uses it to generate a batch of episodes and perform a gradient update. This is repeated a fixed number of times. Note that this procedure does not require any costly data labeling or using previously known solutions.

Once the training is complete, the agent is fine-tuned to the specific problem of interest. This fine-tuning is performed in a similar manner: at each step the agent generates a batch of episodes using the matrix J of the problem and performs a gradient update.

3.5. Implementation details

The agent is implemented as two separate fully-connected networks (actor and critic) with two hidden layers of size 256 and tanh activation functions. These two networks take environment observation as input and produce policy and value function, respectively.

The static features of the problem matrix ϕ_j are not included in the network inputs; instead, they are used to calculate a set of parameters to perform FiLM (Dumoulin *et al* 2016) of the last hidden layer in the actor network. The FiLM module is a linear layer that predicts a set of weights and biases that are used to scale and shift the activations of the actor's hidden layer element-wise.

We train the agent using PPO (Schulman *et al* 2017) with 4 epochs. The discount factor γ is equal to 1.0. SimCIM performs $N = 1000$ iterations per episode, and the agent acts every $m = 10$ iterations, corresponding to 100 steps per episode. The SimCIM algorithm allows efficient parallel implementation on a GPU, so we train the agent in batches of size 256 (both for pre-training and fine-tuning). We use $q = 99$ to calculate rewards in R2 and R3 methods; the leaderboard size P is equal to 5 batch sizes for fine-tuning and one batch size for pre-training (since each problem instance is used to generate only one batch of episodes). The pre-training is performed for 30 000 problem instances.

The SimCIM hyperparameters are chosen as follows. The momentum is set to $\eta = 0.9$ and noise level to $\sigma = 0.03$. The learning rate μ is tuned automatically for each problem instance, including the random instances used for pre-training. The regularization function increment p_Δ is equal to 0.04.

4. Related Work

In addition to classic heuristic methods for combinatorial optimization that can be found in industrial-scale packages like Gurobi (gur 2019) and CPLEX (cpl 2019), many RL-based algorithms are emerging. Early works (Vinyals *et al* 2015, Mirhoseini *et al* 2017) use RL to train recurrent neural networks with attention mechanisms to construct the solution iteratively. In later papers (Khalil *et al* 2017, Li *et al* 2018, Kool *et al* 2018, Mittal *et al* 2019, Abe *et al* 2019, Barrett *et al* 2019) different kinds of graph neural networks are used in conjunction with RL to solve combinatorial problems on graphs by iteratively flipping bit values.

In Laterre *et al* (2018), a permutation-invariant network was used as a RL agent to solve the bin packing problem. This work introduced Ranked Reward to automatically control the learning curriculum of the agent.

Combining RL with heuristics was explored in Xinyun and Yuandong (2018): one agent was used to select a subset of problem components, and another selected an heuristic algorithm to process them.

In Khairy *et al* (2019), a RL agent was used to tune the parameters of a simulated quantum approximate optimization algorithm (QAOA) (Farhi *et al* 2014) to solve the Max-Cut problem and showed strong advantage over black-box parameter optimization methods on graphs with up to 22 nodes. QAOA was designed with near-term noisy quantum hardware in mind but, given the current state of technology, the problem size is limited both in hardware and simulation.

To the best of our knowledge, combining quantum-inspired algorithms with RL for combinatorial optimization in the context of practically significant problem sizes was not explored before.

5. Experiments

To evaluate our method, we use problem instances from Gset (Ye 2003), which is a set of graphs (represented by adjacency matrices J) that is commonly used to benchmark Max-Cut solvers. Gset contains problems of practically significant sizes, from hundreds to thousands of variables from several different distributions.

We concentrate on graphs G1–10. Of these, G1–G5 appear to belong to the Erdös and Rényi (1960) model with the connection probability approximately equal to 0.06, while G6–G10 are weighted graphs with the same adjacency structure, but with approximately half of the edges having weights equal to -1. All of these graphs have 800 nodes.

For all our experiments, we use a single machine with a GeForce RTX 2060 GPU.

5.1. Performance

The agent, pre-trained and fine-tuned as described in section 3.4, is used to generate a batch of solutions, for which we calculate the maximum and median cut value. We also report the fraction of solved instances: the problem is considered solved if the maximum cut over the batch is equal to the best known value reported in Benlic and Hao (2013).

The results are presented in table 1. The obtained maximum and median are normalized by this best known value; the normalized values are further averaged over instances G1–G10 and over three random seeds for each instance (for each random seed we pre-train a new agent). Problem instances G6–G10 belong to a distribution never seen by the agent during the pre-training.

We compare our method to two baseline approaches to tuning the regularization function of SimCIM. In the first approach (labelled 'Linear'), the scaled regularization function $\bar{p}_t$ decays linearly from 1 to 0 during the N SimCIM iterations; in our RL setting, this is equivalent to the agent that always chooses zero increment as the action. In the second approach (labelled 'Manual'), which has been used in the original SimCIM paper (Tiunov *et al* 2019), the regularization function is a parameterized hyperbolic tangent function:

$$p_t = J_m O(\tanh(S(t/N - 0.5)) + D), \tag{9}$$

where $J_m = \max_i \sum_j |J_{ij}|$; t/N is a normalized iteration number and O, S, D are the scale and shift parameters. These parameters are tuned manually for all instances G1–G10 at once: the same hyperparameter set is used for all problem instances. If manually tuned in this fashion, SimCIM solves 8 of G1–G10 instances but the result is stochastic and the probability of solving each instance is different (Tiunov *et al* 2019). We evaluate the baselines by sampling 30 batches of solutions (batch size 256) for each instance and averaging the statistics (maximum, median, fraction of solved) over all batches of all instances.

We also compare our approach to a well-known evolutionary algorithm CMA-ES (Hansen *et al* 2003) (population size 10). We parameterize the regularization function for iteration t according to equation (9), and CMA-ES is used to tune $D \in [-3, 3]$ and $O, S \in [0.01, 10]$ (exponential scale) for at most 500 SimCIM evaluations in batches of size 256 each. We maximize $\mathcal{C}_{\max} + q_{\max}$, where $\mathcal{C}_{\max}$ is the maximum cut over the

Table 1. Performance on Gset: maximum and median normalized cut values are averaged over the instances (G1–G10); Agent-K denotes an agent fine-tuned for K episodes; Agent-0 is not fine-tuned. Standard deviation over three random seeds is reported in brackets for each value.

	Linear	Manual	CMA-ES	Agent-0	Agent-100	Agent-200	Agent-500
Maximum	0.9993 (2×10^{-05})	0.9997 (2×10^{-05})	0.9995 (8×10^{-05})	0.9990 (2×10^{-04})	0.9996 (8×10^{-05})	0.9997 (1×10^{-05})	**0.9998** (0e+00)
Median	0.9942 (5×10^{-05})	0.9946 (3×10^{-04})	0.9933 (4×10^{-04})	0.9901 (2×10^{-03})	0.9901 (1×10^{-04})	0.9925 (2×10^{-03})	**0.9979** (4×10^{-04})
Solved	0.2000 (0e+00)	0.6667 (5×10^{-02})	0.6000 (0e+00)	0.1333 (5×10^{-02})	0.6000 (8×10^{-02})	0.7333 (5×10^{-02})	**0.8000** (0e+00)

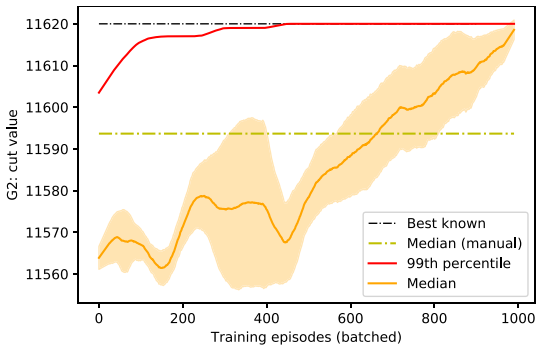

Figure 2. Example: dynamics of the cut value obtained on G2 during fine-tuning; standard deviation is calculated over three random seeds (smoothed with Savitzky–Golay filter).

batch, and $q_{\max}$ is the fraction of values in the batch equal to $\mathcal{C}_{\max}$. Since all elements of J are integer, so is the cut value, while $0 < q_{\max} \leq 1$. As a result, this objective orders batches first by the maximum cut value, and then by the probability to obtain it. After the optimization is finished, the best parameters are selected, and a new batch of solutions is sampled with these parameters. We report results from the batches obtained in this manner, averaged over three random seeds and over all instances.

Though the pre-trained agent without fine-tuning (Agent-0) is even worse than the baselines, fine-tuning rapidly improves the performance of the agent. The fine-tuned agent does not solve all instances in G1–G10, however it discovers high-quality solutions more reliably than the benchmarks.

CMA-ES can solve each of the G1–G10 instances: we observed that the best known value appeared at least once for each instance during several trials with different seeds. However, for some instances this result is not reproducible due to the stochastic nature of SimCIM: a new batch of solutions generated with the best parameters found by CMA-ES may yield a lower maximum cut. In this sense, the results for CMA-ES are worse than for the manually tuned baseline.

Figure 2 demonstrates the dynamics of the maximum and median cut values for the G2 instance during the process of fine-tuning. The median value continues to improve, even after the agent has found the best known value, and eventually surpasses the manually tuned baseline. This means that the agent still finds new ways to reach solutions with the best known cut, thereby increasing the probability to find the best solution. A further advantage of our agent is that it adaptively optimizes the regularization hyperparameter during the test run by taking the current trajectories $\boldsymbol{c}_t$ into account.

The exact maximum cut values after fine-tuning and best known solutions for specific instances G1–G10 are presented in table 2. The agent stably finds the best known solutions for G1–G8 and closely lying solutions for G9–G10. The reason it fails to solve G9 and G10 is that the policy found by the agent corresponds to a deep local optimum that the agent is unable to escape by gradient descent. In contrast, CMA-ES does not use gradient descent and is focused on exploratory search in a broad range of parameters, and hence is sometimes able to solve these graphs. However, even with CMA-ES, the solution probability is vanishingly small: 1.3×10^{-5} for G9 and 9.8×10^{-5} for G10.

The numbers of samples used by the automatic methods—our agent and CMA-ES—differ compared to the manual hyperparameter tuning and the linear variation of the hyperparameter. In the former case, the total number of samples consumed including both training (fine-tuning) and at test equalled $\sim 256 \times 500 = 128,000$. On the other hand, the manual tuning required much fewer samples (tens of thousands), while the linear setting did not involve any tuning at all. Hence it is fair to say that the linear and manual methods are much more sample efficient. While this may be perceived as a weakness of our method,

Table 2. Results for specific Gset instances: best known cut value, best value obtained by the agent, their difference and the probability for the fully trained agent to find a solution corresponding to its best value.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Best (Benlic and Hao 2013)	11 624	11 620	11 622	11 646	11 631	2178	2006	2005	2054	2000
Agent	11 624	11 620	11 622	11 646	11 631	2178	2006	2005	2050	1999
Difference	0	0	0	0	0	0	0	0	–4	–1
Probability	0.87	0.49	0.81	0.93	0.34	0.53	0.82	0.92	0.61	0.46

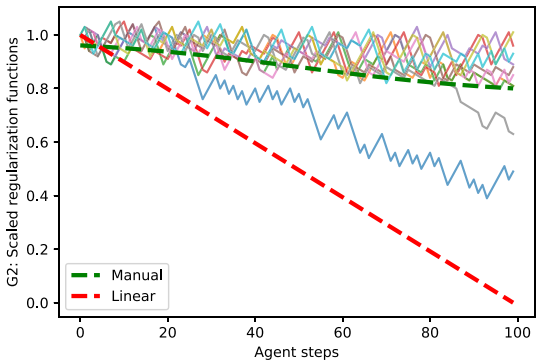

Figure 3. Examples of regularization functions discovered by the agent on instance G2 compared to regularization functions of Linear and Manual approaches.

Table 3. Ablation study, fraction of problems solved. Agent-K denotes an agent fine-tuned for K episodes. Standard deviation over three random seeds is reported in brackets for each value.

	Pre-training			No pre-training		
Solved	R3	R3, no FiLM	R2	R3	R3, no FiLM	R2
Solved (100 it.)	0.60 (8×10^{-02})	**0.63** (5×10^{-02})	0.60 (8×10^{-02})	0.40 (0e+00)	0.37 (5×10^{-02})	0.10 (8×10^{-02})
Solved (200 it.)	**0.73** (5×10^{-02})	0.70 (0e+00)	0.67 (5×10^{-02})	0.47 (5×10^{-02})	0.53 (9×10^{-02})	0.33 (5×10^{-02})
Solved (500 it.)	**0.80** (0e+00)	0.77 (5×10^{-02})	0.70 (0e+00)	0.73 (5×10^{-02})	0.73 (5×10^{-02})	0.53 (5×10^{-02})

we reiterate that its purpose is not to compete with the human expert in sample efficiency but relieve them of the tedious task of parameter tuning.

Figure 3 demonstrates regularization functions discovered by the agent on instance G2, as well as regularization functions of Linear and Manual approaches. Compared to baselines, functions controlled by the agent oscillate during the optimization process, reacting to the change in agent's observations.

5.2. Ablation study

We study the effect of the three main components of our approach: transfer learning from random problems, Rescaled Ranked Rewards (R3) scheme, and FiLM of the actor network with the problem features.

- To study the effect of the policy transfer, we train pairs of agents with the same hyperparameters, architecture and reward type, but with and without pre-training on randomly sampled problems. In the latter case, the parameters of the agent are initialized randomly.
- We compare our R3 method with the original R2 method both with and without pre-training.
- We study the effect of FiLM by removing the static observations extracted from the problem matrix J from the observation and the FiLM layer from the agent.

We report the fraction of solved problems, averaged over instances G1–G10 and over three random seeds for each instance. The results are presented in table 3 and figure 4.

According to the results, all of the above listed features are essential for the agent's performance. We see, in particular, that the pre-trained agent with both FiLM and R3 rewards experiences a slightly slower start, but eventually finds better optima faster than ablated agents.

5.3. Rescaled ranked rewards

The analysis of specific problem instances helps to demonstrate the advantage of the R3 method. We analyze the behavior of the 99th percentile of the solution cut values (the one used to distribute rewards in R2 and

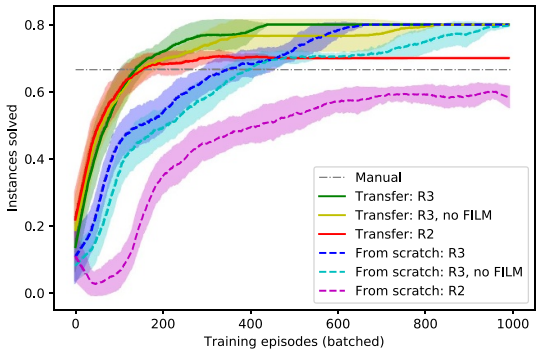

Figure 4. Ablation study: averaged fraction of solved problem instances versus the number of episodes of fine-tuning for each instance (smoothed with Savitzky–Golay filter). Standard deviation is calculated over three random seeds. 'Transfer' and 'From scratch' are used to denote the agent with and without pre-training, respectively.

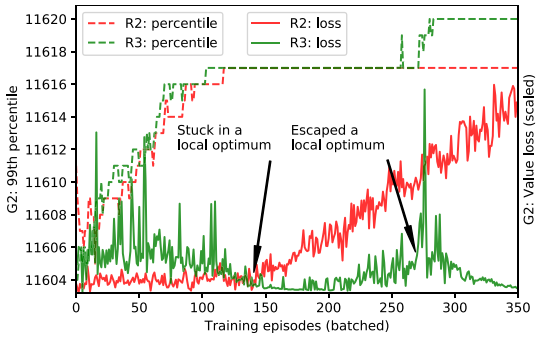

Figure 5. Value loss and 99th percentile during fine-tuning on G2 for R2 and R3 when dealing with local optima.

R3) on the G2 instance from Gset in figure 5. G2 has several local optima with the same cut value 11 617, which are relatively easy to reach. When the agent is stuck in a local optimum, many solutions generated by the agent are likely to have their cut values equal to the percentile, while solutions with higher cut values may appear infrequently.

In the R2 scheme (7), the agent gets random ± 1 rewards for local-optimum solutions and $+1$ for better ones. Thus infrequent solutions with higher cut values become almost indistinguishable from the local-optimum solutions. Furthermore, the fraction of episodes with local-optimum solutions increases, which results in a large fraction of random rewards, thereby preventing the efficient training of the critic network. This is evident from the monotonic growth of the value loss function in figure 5.

In our R3 scheme (8), in contrast, the rewards for the local-optimum solutions are deterministic and dependent on the frequency of such solutions. The more often the agent reaches them, the lower the reward, while the reward for solutions with higher cut values is fixed. Eventually, better solutions outweigh sub-optimal ones, and the agent escapes the local optimum. This moment is indicated by a significant increase of the value loss: the agent starts exploring new, more promising states.

6. Discussion and future work

One of the benefits of our approach is the lightweight architecture of our agent, which allows efficient GPU implementation along with the SimCIM algorithm itself. This allows us to rapidly fine-tune the agent for each problem instance. However, the fully-connected architecture makes it harder to apply our pre-trained agent to problems of various sizes, since the size of the network input layer depends on the problem size. Hence it would be interesting to explore using size-agnostic architectures for the agent, like graph neural networks.

Another future research direction is to train the agent to vary more SimCIM hyperparameters, such as the scaling of the adjacency matrix or the noise level. Additionally, it would be interesting to explore using meta-learning at the pre-training step to accelerate the fine-tuning process.

Lastly, with our approach, each novel instance requires a new run of fine-tuning, leading to a large number of required samples compared with simple instance-agnostic heuristics. In order to make our

approach viable from a practical point of view, we hope to address generalization across different, novel, problem instances more efficiently.

7. Conclusion

In this work we proposed an RL-based approach to tuning the regularization function of SimCIM, a quantum-inspired algorithm, to robustly solve the Ising problem. Our hybrid approach shows a strong advantage over heuristics and a black-box approach, and allows us to sample high-quality solutions with high probability.

We proposed an improvement over the Ranked Reward (R2) scheme, called Rescaled Ranked Reward (R3), which allows the agent to constantly improve the current solution while avoiding local optima. We also demonstrated that our algorithm may be accelerated significantly by pre-training the agent on randomly generated problem instances, and that it generalizes to out-of-distribution problems.

Importantly, our approach is not limited to SimCIM or even the Ising problem, but can be readily generalized to any algorithm based on a continuous relaxation of discrete optimisation.

Acknowledgments

We would like to thank Egor Tiunov for providing the manual tuning data and William Clements and Vitaly Kurin for helpful discussions. This project has received funding from the Russian Science Foundation (19-71-10092).

ORCID iD

A E Ulanov https://orcid.org/0000-0003-2211-559X

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)

Barahona F 1982 On the computational complexity of Ising spin glass models *J. Phys. A: Math. Gen.* **15** 3241

Barrett T D, Clements W R, Foerster J N and Lvovsky A 2019 Exploratory combinatorial optimization with reinforcement learning (arXiv:1909.04063)

Benlic U and Hao J-K 2013 Breakout local search for the max-cut problem *Eng. Appl. Artif. Intell.* **26** 1162–73

Cplex optimizer 2019 (https://www.ibm.com/analytics/cplex-optimizer)

Dumoulin V, Shlens J and Kudlur M 2016 A learned representation for artistic style (arXiv:1610.07629)

Erdős P and Rényi A 1960 On the evolution of random graphs *Publ. Math. Inst. Hung. Acad. Sci* **5** 17–60

Farhi E, Goldstone J and Gutmann S 2014 A quantum approximate optimization algorithm (arXiv:1411.4028)

Feurer M and Hutter F 2018 Hyperparameter optimization *Automatic Machine Learning: Methods, Systems, Challenges* F Hutter, L Kotthoff and J Vanschoren eds (Berlin: Springer) pp 3–38 In press, available at (http://automl.org/book)

Gurobi optimization 2019 (https://www.gurobi.com/)

Hansen N, Müller S D and Koumoutsakos P 2003 Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es) *Evolutionary Computat.* **11** 1–18

Hopfield J and Tank D 1986 Computing with neural circuits: a model *Science* **233** 625–33

Inagaki T *et al* 2016 A coherent ising machine for 2000-node optimization problems *Science* **354** 603–6

Khairy S, Shaydulin R, Cincio L, Alexeev Y and Balaprakash P 2019 Learning to Optimize Variational Quantum Circuits to Solve Combinatorial Problems *Proc. of the AAAI Conf. on Artificial Intelligence* vol **34**

Khalil E, Dai H, Zhang Y, Dilkina B and Song L 2017 Learning combinatorial optimization algorithms over graphs *Advances in Neural Information Processing Systems* pp 6348–58

King A D, Bernoudy W, King J, Berkley A J and Lanting T 2018 Emulating the coherent ising machine with a mean-field algorithm (arXiv:1806.08422)

Kirkpatrick S, Gelatt C D and Vecchi M P 1983 Optimization by simulated annealing *Science* **220** 671–80

Kool W, van Hoof H and Welling M 2018 Attention, learn to solve routing problems! (arXiv:1803.08475)

Laterre A, Fu Y, Jabri M K, Cohen A-S, Kas D, Hajjar K, Dahl T S, Kerkeni A and Beguir K 2018 Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization (arXiv:1807.01672)

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 *Advances in Neural Information Processing Systems* pp 539–48

Marzec M 2016 Portfolio optimization: applications in quantum computing *Handbook of High-Frequency Trading and Modeling in Finance* (New York: Wiley) pp 73–106

McGeoch C C, Harris R, Reinhardt S P and Bunyk P I 2019 Practical annealing-based quantum computing *Computer* **52** 38–46

McMahon P L *et al* 2016 A fully programmable 100-spin coherent ising machine with all-to-all connections *Science* **354** 614–17

Mirhoseini A *et al* 2017 Device placement optimization with reinforcement learning *Proc. of the 34th Int. Conf. on Machine Learning-Volume 70* pp 2430–9 JMLR. org

Mittal A, Dhawan A, Medya S, Ranu S and Singh A 2019 Learning heuristics over large graphs via deep reinforcement learning (arXiv:1903.03332)

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 *Sci. Rep.* **2** 571
Schulman J, Wolski F, Dhariwal P, Radford A and Klimov O 2017 Proximal policy optimization algorithms (arXiv:1707.06347)
Silver D *et al* 2017 Mastering chess and shogi by self-play with a general reinforcement learning algorithm (arXiv:1712.01815)
Smith L N 2017 Cyclical learning rates for training neural networks *2017 IEEE Conf. on Applications of Computer Vision (WACV)* pp 464–72
Tiunov E S, Ulanov A E and Lvovsky A 2019 Annealing by simulating the coherent ising machine *Opt. Express* **27** 10288–95
Ulanov A E, Tiunov E S and Lvovsky A 2019 Quantum-inspired annealers as Boltzmann generators for machine learning and statistical physics (arXiv:1912.08480)
Venturelli D and Kondratyev A 2019 Reverse quantum annealing approach to portfolio optimization problems *Quantum Machine Intell.* **1** 17–30
Vinyals O, Fortunato M and Jaitly N 2015 Pointer networks *Adv. Neural Inf. Process. Syst.* 2692–700
Xinyun C and Yuandong T 2018 Learning to perform local rewriting for combinatorial optimization (arXiv:1810.00337)
Ye Y 2003 Gset max-cut problem set (https://web.stanford.edu/ yyye/yyye/Gset/)