nature machine intelligence

ARTICLES

https://doi.org/10.1038/s42256-022-00468-6

Check for updates

Combinatorial optimization with physics-inspired graph neural networks

Martin J. A. Schuetz[1,2,3] ✉, J. Kyle Brubaker[2] ✉ and Helmut G. Katzgraber[1,2,3] ✉

Combinatorial optimization problems are pervasive across science and industry. Modern deep learning tools are poised to solve these problems at unprecedented scales, but a unifying framework that incorporates insights from statistical physics is still outstanding. Here we demonstrate how graph neural networks can be used to solve combinatorial optimization problems. Our approach is broadly applicable to canonical NP-hard problems in the form of quadratic unconstrained binary optimization problems, such as maximum cut, minimum vertex cover, maximum independent set, as well as Ising spin glasses and higher-order generalizations thereof in the form of polynomial unconstrained binary optimization problems. We apply a relaxation strategy to the problem Hamiltonian to generate a differentiable loss function with which we train the graph neural network and apply a simple projection to integer variables once the unsupervised training process has completed. We showcase our approach with numerical results for the canonical maximum cut and maximum independent set problems. We find that the graph neural network optimizer performs on par or outperforms existing solvers, with the ability to scale beyond the state of the art to problems with millions of variables.

Optimization is ubiquitous across science and industry. Specifically, the field of combinatorial optimization—the search for the minimum of an objective function within a finite but often large set of candidate solutions—is one of the most important areas in the field of optimization, with practical (yet notoriously challenging) applications found in virtually every industry, including both the private and public sectors, as well as in areas such as transportation and logistics, telecommunications and finance[1–5]. Although efficient specialized algorithms exist for specific use cases, most optimization problems remain intractable, especially in real-world applications where problems are more structured and thus require additional steps to make them amenable to traditional optimization techniques. Despite remarkable advances in both algorithms and computing power, substantial yet generic improvements have remained elusive, generating an increased interest in new optimization approaches that are broadly applicable and radically different from traditional operations research tools.

In the broader physics community, the advent of quantum annealing devices such as the D-Wave Systems Inc. quantum annealers[6–9] has spawned a renewed interest in the development of heuristic approaches to solve discrete optimization problems. On the one hand, recent advances in quantum science and technology have inspired the development of novel classical algorithms, sometimes dubbed nature-inspired or physics-inspired algorithms (for example, simulated quantum annealing[10,11] running on conventional CMOS hardware), that have raised the bar for emerging quantum annealing hardware (for example, refs. [12–15]). On the other hand, in parallel to these algorithmic developments, substantial progress has been made in recent years in the development of programmable special-purpose devices based on alternative technologies, such as the coherent Ising machine based on optical parametric oscillators[16,17], digital MemComputing machines based on self-organizing logic gates[18,19], and the ASIC-based Fujitsu digital annealer (ASIC, application-specific integrated circuit)[20–22]. Some of these approaches face severe scalability limitations. For example, in the coherent Ising machine there is a trade-off between precision and the number of variables, and the Fujitsu digital annealer—baked into an ASIC—can currently handle, at most, 8,192 variables. It is thus of great interest to find new alternative approaches to tackle large-scale combinatorial optimization problems, going far beyond what is currently accessible with quantum and nature-inspired approaches alike.

In the deep learning community, graph neural networks (GNNs) have seen a burst in popularity over the past few years[23–30]. In essence, GNNs are deep neural network architectures specifically designed for graph structure data, with the ability to learn effective feature representations of nodes, edges or even entire graphs. Prime examples of GNN applications include classification of users in social networks[31,32], the prediction of future interactions in recommender systems[33] and the prediction of certain properties of molecular graphs[34,35]. As a convenient and general framework to model a variety of real-world complex structural data, GNNs have been applied successfully to a broad set of problems, including recommender systems in social media and e-commerce[36,37], the detection of misinformation (fake news) in social media[38], and various domains of natural sciences including event classification in particle physics[39,40], to name a few. Although several specific implementations of GNNs exist[29,41,42], at their core, GNNs typically iteratively update the features of the nodes of a graph by aggregating the information from their neighbours (often referred to as message passing[43]), thereby iteratively making local updates to the graph structure as the training of the network progresses. Because of their scalability and inherent graph-based design, GNNs present an alternative platform on which to build large-scale combinatorial heuristics.

In this Article we present a highly scalable GNN-based solver to (approximately) solve combinatorial optimization problems with up to millions of variables. The approach is schematically depicted in Fig. 1, and works as follows. First, we identify the Hamiltonian (cost function) H that encodes the optimization problem in terms of binary decision variables $x_v \in \{0, 1\}$ and we associate this variable

[1]Amazon Quantum Solutions Lab, Seattle, WA, USA. [2]AWS Intelligent and Advanced Compute Technologies, Professional Services, Seattle, WA, USA. [3]AWS Center for Quantum Computing, Pasadena, CA, USA. ✉e-mail: maschuet@amazon.com; johbruba@amazon.com; katzgrab@amazon.com

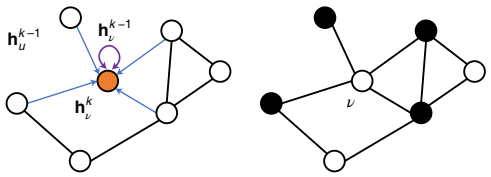

Fig. 1 | Schematic of the GNN approach for combinatorial optimization presented in this work. Following a recursive neighbourhood aggregation scheme, the graph neural network is iteratively trained against a custom loss function that encodes the specific optimization problem (for example, maximum cut). At training completion, we project the final values for the soft node assignments at the final graph neural network layer back to binary variables $x_i = 0, 1$, providing the solution bit string $\mathbf{x} = (x_1, x_2, \ldots)$. Further details are given in the main text.

with a vertex $v \in \mathcal{V}$ for an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{1, 2, \ldots, n\}$ and the edge set $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}\}$ capturing interactions between the decision variables. We then apply a relaxation strategy to the problem Hamiltonian to generate a differentiable loss function, with which we perform unsupervised training on the node representations of the GNN. The GNN follows a standard recursive neighbourhood aggregation scheme[43,44], where each node $v = 1, 2, \ldots, n$ collects information (encoded as feature vectors) of its neighbours to compute its new feature vector $\mathbf{h}_v^k$ at layer $k = 0, 1, \ldots, K$. After k iterations of aggregation, a node is represented by its transformed feature vector $\mathbf{h}_v^k$, which captures the structural information within the node's k-hop neighbourhood[28]. For binary classification tasks we typically use convolutional aggregation steps, followed by the application of a nonlinear softmax activation function to shrink down the final embeddings $\mathbf{h}_v^K$ to one-dimensional soft (probabilistic) node assignments $p_v = \mathbf{h}_v^K \in [0, 1]$. Finally, once the unsupervised training process has completed, we apply a projection heuristic to map these soft assignments p_v back to integer variables $x_v \in \{0, 1\}$, using, for example, $x_v = \mathrm{int}(p_v)$. We numerically showcase our approach with results for canonical NP-hard optimization problems such as maximum cut (MaxCut) and maximum independent set (MIS), showing that our GNN-based approach can perform on par or even better than existing well-established solvers, while being broadly applicable to a large class of optimization problems. The scalability of our approach also opens up the possibility of studying unprecedented problem sizes with hundreds of millions of nodes when leveraging distributed training in a mini-batch fashion on a cluster of machines, as demonstrated recently in ref. [45].

The Article is structured as follows. In the next section we provide some context for our work, discussing recent developments at the cross-section between machine learning and combinatorial optimization. We then summarize the basic concepts underlying our approach, as well as information on the class of problems that this approach can solve. An outline of the implementation of the proposed GNN-based optimizer follows, then numerical experiments. We next discuss potential real-world applications in industry. Finally, we draw conclusions and give an outlook on future directions of research.

Related work

In this section we briefly review the relevant existing literature, with the goal to provide a detailed context for our work. Broadly speaking, our work makes a physics-inspired contribution to the emerging cross-fertilization between combinatorial optimization and machine learning, where the development of novel deep learning architectures has sparked a renewed interest in heuristics for solving NP-hard combinatorial optimization problems using neural networks, as extensively reviewed in refs. [46,47], for example. Leaving alternative, non-graph-based approaches aside (as presented, for example, in ref. [48]), in the following short survey we focus on graph-based optimization problems—where modern deep learning architectures such as sequence models, attention mechanisms and GNNs provide a natural tool set[46]—and we primarily distinguish between approaches based on supervised learning, reinforcement learning or unsupervised learning. This categorization can be refined further with respect to the typical size of a problem solved by a specific approach and the scope of the solver (special-purpose versus general-purpose).

Supervised learning. The majority of neural-network-based approaches to combinatorial optimization are based on supervised learning, with the goal to approximate some (typically complex, nonlinear) mapping from an input representation of the problem to the target solution, based on the minimization of some empirical, handcrafted loss function. Early work was based on pointer networks, which leverage sequence-to-sequence models to produce permutations over inputs of variable size, as is relevant for the canonical travelling salesman problem (TSP)[49]. Since then, numerous studies have fused GNNs with various heuristics and search procedures to solve specific combinatorial optimization problems, such as quadratic assignment[50], graph matching[51], graph colouring[52] and the TSP[53,54]. As pointed out in ref. [55], however, the viability and performance of supervised approaches critically depends on the existence of large, labelled training datasets with previously optimized hard problem instances, resulting in a problematic chicken-and-egg scenario that is further amplified by the fact that it is hard to efficiently sample unbiased and representative labelled instances of NP-hard problems[56].

Reinforcement learning. The critical need for training labels can be circumvented with reinforcement learning (RL) techniques that aim to learn a policy with the goal of maximizing some expected reward function. Specifically, optimization problems can typically be described with a native objective function that can then serve as a reward function in an RL approach[46]. Motivated by the challenges associated with the need for optimal target solutions, Bello and colleagues extended the pointer network architecture[49] to an actor-critic RL framework to train an approximate TSP solver, using a recurrent neural network encoder scheme and the expected tour length as a reward signal[57]. Using a general RL framework based on a graph attention network architecture[42], notable improvements in accuracy on a two-dimensional Euclidean TSP have subsequently been presented in ref. [58], getting close to optimal results for problems with up to 100 nodes. Moreover, TSP variants with hard constraints have been analysed in ref. [59], with the help of a multi-level RL framework in which each layer of a hierarchy learns a different policy, and from which actions can then be sampled. Finally, while the majority of RL-based approaches have focused on the TSP or variants thereof, ref. [60] proposes a combination of RL and graph embedding to learn efficient greedy meta-heuristics to incrementally construct a solution, showcasing this approach with numerical results for minimum vertex cover, MaxCut and TSP as test problems, for graphs with up to ~1,000–1,200 nodes.

Unsupervised learning. Conceptually, our work is most similar to those that aim to train neural networks in an unsupervised, end-to-end fashion, without the need for labelled training sets[55]. Specifically, Toenshoff and colleagues have recently used a recurrent GNN architecture—dubbed RUN-CSP—to solve optimization problems that can be framed as maximum constraint satisfaction problems[61]. For other types of problem, such as the maximum independent set problem, the model relies on empirically selected handcrafted loss functions. Using the language of constraint

satisfaction problems, where the system size is expressed in terms of both the number of variables and the number of constraints, the authors solve problem instances of Maximum 2-satisfiability, 3-colourability, MaxCut and maximum independent set with up to 5,000 nodes, showing that RUN-CSP can compete with traditional approaches like greedy heuristics or semidefinite programming. Finally, by either optimizing a smooth relaxation of the cut objective or applying a policy gradient, Yao and colleagues trained a GNN to specifically solve the MaxCut problem, albeit at relatively small system sizes with up to 500 nodes[62], and without any details on runtime.

In this Article we present a highly scalable, physics-inspired framework that uses deep-learning tools in the form of GNNs to approximate solutions to hard combinatorial optimization problems with up to millions of variables. Our GNN optimizer is based on a direct mathematical relation between prototypical Ising spin Hamiltonians[63], the quadratic binary unconstrained optimization (QUBO) and polynomial binary unconstrained optimization (PUBO) formalism and the differentiable loss function with which we train the GNN, thereby providing one unifying framework for a broad class of combinatorial optimization problems, and opening up the powerful toolbox of statistical physics to modern deep-learning approaches. Fusing concepts from statistical physics with modern machine learning tooling, we propose a simple, generic and robust solver that does not rely on handcrafted loss functions. Specifically, we show that the same GNN optimizer can solve different QUBO problems, without any need to change the architecture or loss function, while scaling to problem instances that are orders of magnitude larger than what many traditional QUBO solvers can handle[6,12,64,65].

Preliminaries

To set up our notation and terminology we start out with a brief review of both combinatorial optimization and graph neural networks.

Combinatorial optimization. The field of combinatorial optimization is concerned with settings where a large number of yes/no decisions must be made and each set of decisions yields a corresponding objective function value, like a cost or profit value, that is to be optimized[1]. Canonical combinatorial optimization problems include, among others, the maximum cut problem (MaxCut), the maximum independent set problem (MIS), the minimum vertex cover problem, the maximum clique problem and the set cover problem. In all cases, exact solutions are not feasible for sufficiently large systems due to the exponential growth of the solution space as the number of variables n increases. Bespoke (approximate) algorithms to solve these problems can typically be identified, at the cost of limited scope and generalizability. Conversely, in recent years, the QUBO framework has resulted in a powerful approach that unifies a rich variety of these NP-hard combinatorial optimization problems[1–3,66]. The cost function for a QUBO problem can be expressed in compact form with the Hamiltonian

$$H_{\mathrm{QUBO}} = \mathbf{x}^{\top} Q \mathbf{x} = \sum_{i,j} x_i Q_{ij} x_j, \tag{1}$$

where $\mathbf{x}=(x_1, x_2, \ldots)$ is a vector of binary decision variables and the QUBO matrix Q is a square matrix of constant numbers that encodes the actual problem to solve. Without loss of generality, the Q matrix can be assumed to be symmetric or in an upper triangular form[1]. We have omitted any irrelevant constant terms, as well as any linear terms as these can always be absorbed into the Q matrix because $x_i^2 = x_i$ for binary variables $x_i \in \{0, 1\}$. Problem constraints, which are relevant for many real-world optimization problems, can be accounted for with the help of penalty terms entering the objective function (rather than being explicitly imposed), as detailed in ref. [1]. The significance of QUBO problems is further illustrated by the close relation to the famous Ising model, which is known to provide mathematical formulations for many NP-complete and NP-hard problems, including all of Karp's 21 NP-complete problems[66]. As opposed to QUBO problems, Ising problems are described in terms of binary spin variables $z_i \in \{-1, 1\}$, which can be mapped straightforwardly to their equivalent QUBO form, and vice versa, using $z_i = 2x_i - 1$. By definition, both the QUBO and the Ising models are quadratic, but can be naturally generalized to higher-order PUBO problems, as described by the N-local Hamiltonian

$$H_{\mathrm{PUBO}} = \sum_{k=0}^{N} \sum_{\langle i_1, i_2, \ldots, i_k \rangle} Q_{i_1 i_2 \cdots i_k} x_{i_1} x_{i_2} \cdots x_{i_k}, \tag{2}$$

with real-numbered coefficients $Q_{i_1 i_2 \cdots i_k}$ for some $N \geq 3$, and $\langle i_1, i_2, \ldots, i_k \rangle$ indicating a group of k binary variables (or spins in the Ising formulation). Terms containing a product of k variables, of the form $Q_{i_1 i_2 \cdots i_k} x_{i_1} x_{i_2} \cdots x_{i_k}$, are commonly referred to as k-local interactions, with $Q_{i_1 i_2 \cdots i_k}$ being the coupling constant. As we exemplify below for some canonical problems, graph (hypergraph) problems can be naturally framed as QUBO (PUBO) problems. To this end, given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we simply associate a binary variable x_i with every vertex $i \in V$, and then express the (node classification) objective as a QUBO problem, where the specific assignment $\mathbf{x}$ can be visualized as a specific two-tone (for example, light and dark) colouring of the graph (with colouring of the graph we refer to a specific node classification as given by the assignment vector $\mathbf{x}$, taking for example $x_i = 0$ as red node colouring and $x_i = 1$ as blue node colouring; we do not refer to the well-known vertex colouring problem, which seeks to colour the vertices of a graph such that no two adjacent vertices are of the same colour) (Fig. 1).

GNNs. On a high level, GNNs are a family of neural networks capable of learning how to aggregate information in graphs for the purpose of representation learning. Typically, a GNN layer is composed of three functions[35]: (1) a message passing function that permits information exchange between nodes over edges, (2) an aggregation function that combines the collection of received messages into a single, fixed-length representation and (3) a (typically nonlinear) update activation function that produces node-level representations given the previous layer representation and the aggregated information. Although a single-layer GNN encapsulates a node's features based on its immediate or one-hop neighbourhood, by stacking multiple layers, the model can propagate each node's features through intermediate nodes, analogous to the broadening of the receptive field in downstream layers of convolutional neural networks. Formally, at layer $k=0$, each node $\nu \in \mathcal{V}$ is represented by some initial representation $\mathbf{h}_\nu^0 \in \mathbb{R}^{d_0}$, usually derived from the node's label or given input features of dimensionality d_0 (ref. [67]). Following a recursive neighbourhood aggregation scheme, the GNN then iteratively updates each node's representation, in general described by some parametric function f_θ^k, resulting in

$$\mathbf{h}_\nu^k = f_\theta^k \left(\mathbf{h}_\nu^{k-1}, \{ \mathbf{h}_u^{k-1} | u \in \mathcal{N}_\nu \} \right), \tag{3}$$

for the layers $k = 1, \ldots, K$, with $\mathcal{N}_\nu = \{u \in \mathcal{V} | (u, \nu) \in \mathcal{E}\}$ referring to the local neighbourhood of node ν, that is, the set of nodes that share edges with node ν. The total number of layers K is usually determined empirically as a hyperparameter, as is the intermediate representation dimensionality d_k. Both can be optimized in an outer

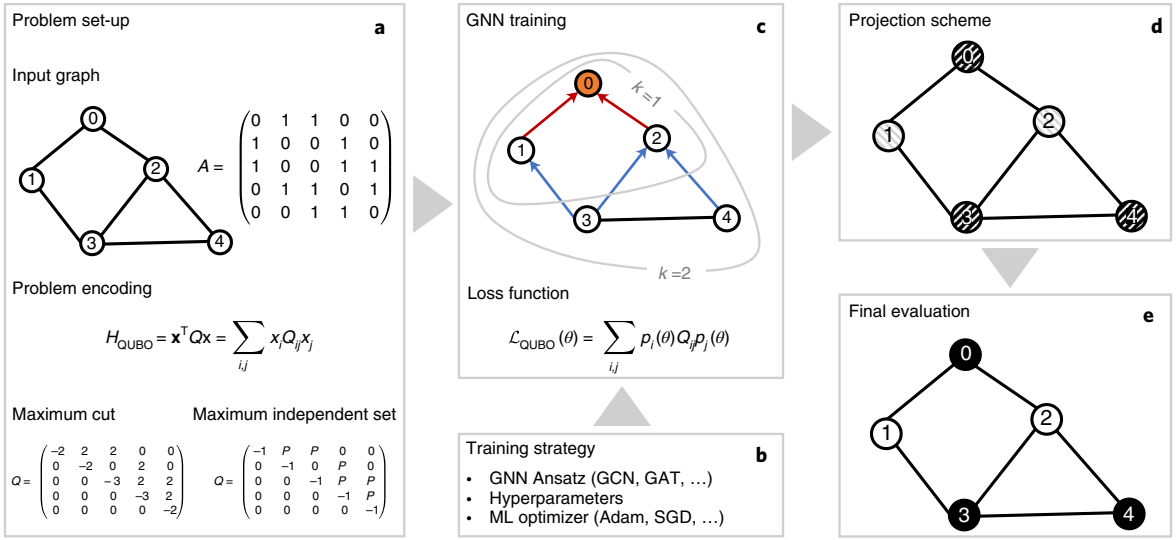

Fig. 2 | Flow chart illustrating the end-to-end workflow for the proposed physics-inspired GNN optimizer. a, The problem is specified by a graph $\mathcal{G}$ with associated adjacency matrix A, and a cost function as described (for example) by the QUBO Hamiltonian H_{QUBO}. Within the QUBO framework the cost function is fully captured by the QUBO matrix Q, as illustrated for both MaxCut and MIS for a sample (undirected) graph with five vertices and six edges. **b**, The problem set-up is complemented by a training strategy that specifies the GNN Ansatz, a choice of hyperparameters and a specific ML optimizer (for example, Adam, stochastic gradient descent (SGD)). **c**, The GNN is iteratively trained against a custom loss function $\mathcal{L}_{\text{QUBO}}(\theta)$ that encodes a relaxed version of the underlying optimization problem as specified by the cost function H_{QUBO}. Typically, a GNN layer operates by aggregating information within the local one-hop neighbourhood (as illustrated by the $k=1$ circle for the top node with label 0). By stacking layers one can extend the receptive field of each node, thereby allowing distant propagation of information (as illustrated by the $k=2$ circle for the top node with label 0). **d**,**e**, The GNN generates soft node assignments, which can be viewed as class probabilities. Using some projection scheme (**d**), we then project the soft node assignments back to (hard) binary variables $x_i=0,1$ (as indicated by the binary black/white node colouring), providing the final solution bit string $\mathbf{x}$ (**e**).

loop. Although a growing number of possible implementations for GNN architectures[30] exists, here we use a graph convolutional network (GCN)[29] for which equation (3) reads explicitly as

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u\in\mathcal{N}(v)} \frac{\mathbf{h}_u^{k-1}}{|\mathcal{N}(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \tag{4}$$

with $\mathbf{W}_k$ and $\mathbf{B}_k$ being (shared) trainable weight matrices, the denominator $|\mathcal{N}(v)|$ serving as a normalization factor (with other choices available as well) and $\sigma(\cdot)$ being some (component-wise) nonlinear activation function such as sigmoid or ReLU. Although GNNs can be used for various prediction tasks (including node classification, link prediction, community detection, network similarity or graph classification), here we focus on node classification, where, usually, the last (Kth) layer's output is used to predict a label y_v for every node $v \in \mathcal{V}$. To this end, we feed the (parametrized) final node embeddings $\mathbf{z}_v = \mathbf{h}_v^K(\theta)$ into a problem-specific loss function and run stochastic gradient descent to train the weight parameters.

Combinatorial optimization with GNNs

We now detail how to use GNNs to solve combinatorial optimization problems, as schematically outlined in Fig. 2. To this end, we frame combinatorial optimization problems as unsupervised node classification tasks, without the need for any labelled data. Because the nodes do not carry any inherent features, in our set-up the node embeddings $\mathbf{h}_v^0$ are initialized randomly. Warm-starting the training process with pre-training (transfer learning) will be left for future research. The Hamiltonians belonging to the class described above are not differentiable and cannot be used straightforwardly within the GNN training process. Therefore, for a given problem Hamiltonian H and graph $\mathcal{G}$, we generate a differentiable loss function $\mathcal{L}(\theta)$, as required for standard backpropagation, by promoting the binary decision variables $x_i \in \{0,1\}$ to continuous (parametrized) probability parameters $p_i(\theta)$ with the following (heuristic) relaxation approach:

$$x_i \longrightarrow p_i(\theta) \in [0,\ 1]. \tag{5}$$

The soft assignments p_i can be viewed as class probabilities. They are generated by our GNN Ansatz as final node embeddings $p_i = \mathbf{h}_i^K \in [0,\ 1]$ at layer K, after the application of a nonlinear softmax activation function. They are then used as input for the loss function $\mathcal{L}(\theta)$. In particular, for QUBO-type problems

$$H_{\text{QUBO}} \longrightarrow \mathcal{L}_{\text{QUBO}}(\theta) = \sum_{i,j} p_i(\theta) Q_{ij} p_j(\theta), \tag{6}$$

which is differentiable with respect to the parameters of the GNN model, θ, and similarly for PUBO problems on hypergraphs with higher-order terms of the form $p_i p_j p_k$ and so on, thereby establishing a straightforward, general connection between combinatorial optimization problems, Ising Hamiltonians and GNNs. For training with gradient descent, standard ML optimizers such as Adam can be used. Once the (unsupervised) training process has completed, we apply projection heuristics to map these soft assignments p_i back to integer variables $x_i=0,1$, using, for example, simply $x_i=\text{int}(p_i)$. The application of other, more sophisticated projection schemes will be left for future research. Note that any projection heuristics can be applied throughout training after every epoch, thereby increasing the pool of solution candidates, at no additional computational cost. With the GNN guiding the search through the solution space, one can then book-keep all solution candidates identified throughout training and simply pick the best solution found.

Our general GNN approach features several hyperparameters, including the number of layers K, the dimensionality of the

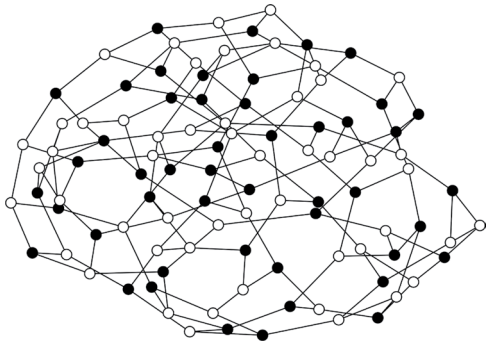

Fig. 3 | Example solution to MaxCut for a random 3-regular graph with $n=100$ nodes. After training completion, the GNN provides a binary bit string $\mathbf{x}$ that assigns one of two possible colours (for example, black or white) to each vertex. An edge is said to be cut when it connects two vertices of different colours. For a given graph, the optimization problem is to assign the colours in such a way that as many edges as possible can be cut at the same time (corresponding to the antiferromagnetic ground state of the system).

embedding vectors $\mathbf{h}_i^k$ and the learning rate β, with details depending on the specific architecture and optimizer used. These can be fine-tuned and optimized in an outer loop, using, for example, standard techniques such as grid search or more advanced Bayesian optimization methods.

Our GNN-based approach can be readily implemented with open-source libraries such as PyTorch Geometric[68] or the Deep Graph Library[69]. The core of the corresponding code is displayed in the Supplementary Information for a GCN with two layers and a loss function for any QUBO problem. For illustration, an example solution to the archetypal MaxCut problem (as implemented with this Ansatz) for a 3-regular graph with $n=100$ vertices is shown in Fig. 3. Here, the cut size achieved with our GNN method amounts to 132. Further details are provided in the following.

Numerical experiments

We perform numerical experiments using MaxCut and MIS benchmark problems. Before providing details on these numerical experiments, we first describe our GNN model architecture as it is consistent across the d-regular MaxCut and MIS problem instances described below. It is certainly possible that better solutions can be found by fine-tuning the hyperparameters for every given problem instance. However, one of our goals is to design a robust and scalable solver that is able to solve a large sample of instances efficiently without the need for hand-tuning the parameters on an instance-by-instance basis.

GNN architecture. We use a simple two-layer GCN architecture based on PyTorch GraphConv units. The first convolutional layer is fed the node embeddings of dimension d_0 and outputs a representation of size d_1. Next, we apply a component-wise, nonlinear ReLU transformation. The second convolutional layer is then fed this intermediate representation and outputs the output layer of size d_2, which is then fed through the component-wise sigmoid transformation to provide a soft probability $p_i \in [0,1]$ for every node $i \in \mathcal{V}$. We find that the following simple heuristic for determining the hyperparameters d_0 and d_1 works well: if the number of nodes is large ($n \geq 10^5$), then we set $d_0 = \mathrm{int}(\sqrt{n})$, else we set $d_0 = \mathrm{int}(\sqrt[3]{n})$, and we take $d_1 = \mathrm{int}(d_0/2)$. Because we solve for binary classification tasks, we set the final output dimension as $d_2 = 1$. However, for multi-colour problems this could be extended to $C > 2$ classes by passing the output layer through a softmax transformation (instead of a sigmoid) and taking the argmax. Note that as the graph size scales beyond ~10^5 nodes, memory becomes a concern, so we further reduce the representations to allow the GNN to be trained on a single graphics processing unit (GPU). Distributed training leveraging a whole cluster of machines will be discussed in the 'Conclusion and outlook' section. With the GNN's output depending on the random initialization of the hidden feature vectors, there is a risk of becoming stuck in a local optimum where the GNN stops learning. To counter this issue, one can take multiple shots (that is, run the GNN training multiple times for different random seeds and choose the best solution), thereby boosting the performance at the cost of extended runtime. In our numerical experiments we limited the number of shots per instance to five, only re-running the training when an obviously sub-optimal solution was detected. Finally, we set the learning rate to $\beta = 10^{-4}$ and allow the model to train for up to ~10^5 epochs, with a simple early stopping rule set to an absolute tolerance of 10^{-4} and a patience of 10^3.

Maximum cut. MaxCut is an NP-hard combinatorial optimization problem with practical applications in machine scheduling[70], image recognition[71] and electronic circuit layout design[72]. In the current era of noisy intermediate-scale quantum devices, with the advent of novel hybrid quantum-classical algorithms such as the quantum approximate optimization algorithm (QAOA)[73], the MaxCut problem has recently attracted considerable attention as a potential use case of pre-error-corrected quantum devices[74–79]. MaxCut is a graph partitioning problem defined as follows: given a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, we seek a partition of V into two subsets with maximum cut, where a cut refers to edges connecting two nodes from different vertex sets. Intuitively, this means we score a point whenever an edge connects two nodes of different colours. To formulate MaxCut mathematically, we introduce binary variables satisfying $x_i = 1$ if vertex i is in one set and $x_i = 0$ if it is in the other set. It is then easy to verify that the quantity $x_i + x_j - 2x_ix_j = 1$ if the edge (i, j) has been cut, and 0 otherwise. With the help of the adjacency matrix A_{ij} with $A_{ij} = 0$ if edge (i, j) does not exist and $A_{ij} > 0$ if a (possibly weighted) edge connects node i with j, the MaxCut problem is described by the quadratic Hamiltonian

$$H_{\mathrm{MaxCut}} = \sum_{i<j} A_{ij}(2x_ix_j - x_i - x_j), \tag{7}$$

which falls into the broader class of QUBO problems described by equation (1). We provide the explicit Q-matrix for a sample MaxCut problem in Fig. 2. Up to an irrelevant constant, the MaxCut problem can equivalently by described by the compact Ising Hamiltonian $H_{\mathrm{MaxCut}} = \sum_{i<j} J_{ij} z_i z_j$ with $J_{ij} = A_{ij}/2$, favouring antiferromagnetic ordering of the spins for $J_{ij} > 0$, as expected intuitively based on the problem definition. As our figure of merit, we denote the largest cut found as $\mathrm{cut}^\star = -H_{\mathrm{MaxCut}}(\mathbf{x}^\star)$, with $\mathbf{x}^\star$ referring to the corresponding bit string.

The complexity of MaxCut depends on the regularity and connectivity of the underlying graph. Following an existing trend in the community[75], we first consider the MaxCut problem on random (unweighted) d-regular graphs, where every vertex is connected to exactly d other vertices. We perform the benchmarks as follows. For graphs with up to a few hundred nodes, we compare our GNN-based solver to the (approximate) polynomial-time Goemans–Williamson (GW) algorithm[80], which provides the current record for an approximate answer within some fixed multiplicative factor of the optimum (referred to as approximation ratio α), using semidefinite programming and randomized rounding. Specifically, the GW algorithm achieves a guaranteed approximation ratio of $\alpha \sim 0.878$ for generic graphs. This lower bound can be raised for specific graphs such as unweighted 3-regular graphs where $\alpha \sim 0.9326$ (ref. [81]). Our implementation of the GW algorithm is based on the open-source CVXOPT solver, with CVXPY as the modelling interface.

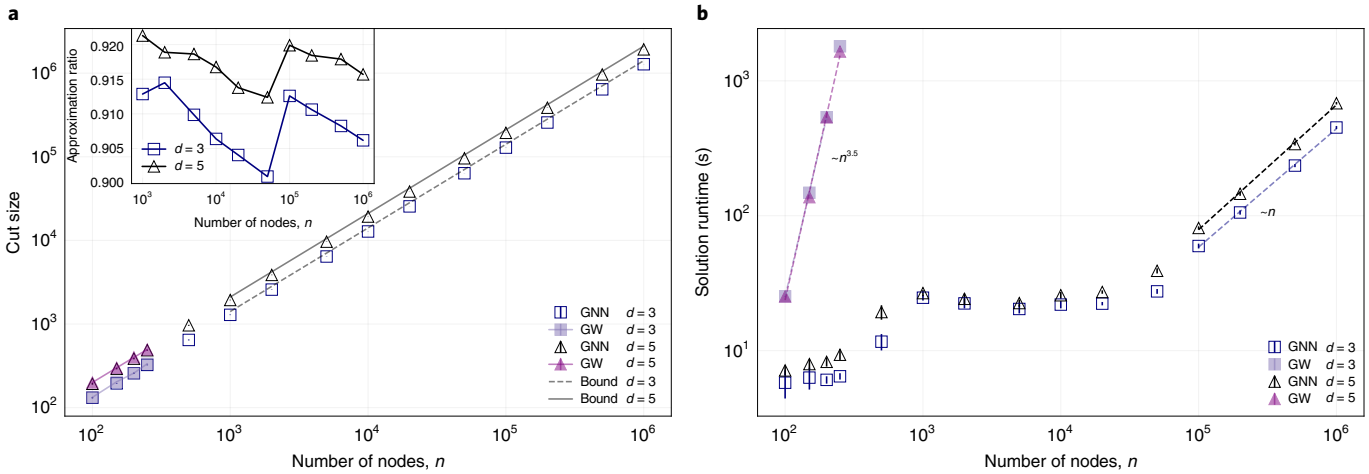

Fig. 4 | Numerical results for MaxCut. a, Average cut size for d-regular graphs with $d=3$ and $d=5$ as a function of the number of vertices n, bootstrap-averaged over 20 random graph instances, for both the GNN-based method and the GW algorithm. On each graph instance, the GNN solver is allowed up to five shots, and the GW algorithm takes 100 shots. Solid lines for $n \geq 10^3$ represent theoretical upper bounds, as described in the main text. Inset: the estimated relative approximation ratio, defined as $\mathrm{cut}^\star/\mathrm{cut}_{\mathrm{ub}}$, showing that our approach consistently achieves high-quality solutions. **b**, Algorithm runtime in seconds for both the GNN solver and the GW algorithm. Error bars refer to twice the bootstrapped standard deviations, sampled across 20 random graph instances for every data point.

For very large graphs with up to a million nodes, numerical benchmarks are not available, but we can compare our best solution $\mathrm{cut}^\star$ to an analytical result derived in ref. [82], where it was shown that with high probability (in the limit $n \to \infty$) the size of the maximum cut for random d-regular graphs with n nodes is given by $\mathrm{cut}^\star = (d/4 + P_* \sqrt{d/4} + \mathcal{O}(\sqrt{d}))n + \mathcal{O}(n)$. Here, $P_* \approx 0.7632$ refers to a universal constant related to the ground-state energy of the Sherrington–Kirkpatrick model[83,84], which can be expressed analytically via Parisi's formula[82]. We thus take $\mathrm{cut}_{\mathrm{ub}} = (d/4 + P_* \sqrt{d/4})n$ as an upper-bound estimate for the maximum cut size in the large-n limit. We complement this upper bound with a lower bound as achieved by a simple, randomized 0.5-approximation algorithm that (on average) cuts half of the edges, yielding a cut size of $\mathrm{cut}_{\mathrm{rnd}} \approx (d/4)n$ for a d-regular graph with $|\mathcal{E}| = (d/2)n$. Our results for the achieved cut size as a function of the number of vertices n are shown in Fig. 4. All results are bootstrapped estimates of the mean, with error bars denoting twice the bootstrapped standard deviations, sampled across 20 random d-regular graphs for every data point. For graphs with up to a few hundred nodes, we find that a simple two-layer GCN architecture can perform on par with the GW algorithm, while showing a runtime advantage compared to GW starting at around $n \approx 100$ nodes. For large graphs with $n \approx 10^4$ to 10^6 nodes, we find that our approach consistently achieves high-quality solutions with $\mathrm{cut}^\star \gtrsim 0.9 \times \mathrm{cut}_{\mathrm{ub}}$ for both $d=3$ and $d=5$, respectively (that is, much better than any naive randomized algorithm). As expected for d-regular graphs, we find $\mathrm{cut}^\star$ to scale linearly with the number of nodes n, that is, $\mathrm{cut}^\star \approx \gamma_d n$, with $\gamma_3 \approx 1.28$ and $\gamma_5 \approx 1.93$ for $d=3$ and $d=5$, respectively. Moreover, utilizing modern GPU hardware, we observe a favourable runtime scaling at intermediate and large system sizes that allows us to solve instances with $n=10^6$ nodes in ~10 min (which includes both GNN model training and post-processing steps). Specifically, as shown in Fig. 4, we observe an approximately linear scaling of total runtime with $\sim n$ for large d-regular graphs with $10^5 \leq n \leq 10^6$, contrasting with the observed GW algorithm scaling as $\sim n^{3.5}$ for problem sizes in the range $n \lesssim 250$, thereby showing the (expected) time complexity $\tilde{O}(n^{3.5})$ of the interior-point method (as commonly used for solving the semidefinite program underlying the GW algorithm) that dominates the GW algorithm runtime[85,86].

To complement our work on random d-regular graphs, we performed additional experiments on standard MaxCut benchmark instances, with published results, based on the publicly available Gset dataset[87] that is commonly used for testing MaxCut algorithms. We provide benchmark results for seven different graphs, with thousands of nodes, including (1) two Erdös–Renyi graphs with uniform edge probability, (2) two graphs where the connectivity gradually decays from node 1 to node n, (3) two 4-regular toroidal graphs and (4) one of the largest Gset instances with $n=10^4$. The results are displayed in Table 1. Here we report cut sizes achieved with our physics-inspired GNN solver (PI-GNN), together with results sourced from refs. [61,88–90]; the latter include an SDP solver using dual scaling (DSDP)[90], a combination of local search and adaptive perturbation referred to as breakout local search (BLS)[89] (providing the best known solutions for the Gset dataset), a Tabu Search metaheuristic (KHLWG)[88] and a recurrent GNN architecture for maximum constraint satisfaction problems (RUN-CSP)[61]. We assess the solution quality achieved with PI-GNN with the relative error $\epsilon = (\mathrm{cut}_{\mathrm{best}} - \mathrm{cut}^\star)/|\mathcal{E}|$ quantifying the gap to the best known solution, normalized by the number of edges $|\mathcal{E}|$, thereby giving the fraction of uncut edges as compared to the best known solution. We find that our general-purpose approach is competitive with other solvers and typically within ~1% of the best published results.

Maximum independent set. The MIS problem is a prominent combinatorial optimization problem with practical applications in network design[91] and finance[92], and is closely related to the maximum clique, minimum vertex cover and set packing problems. In the quantum community, the MIS problem has recently attracted much interest[93] as a potential target use case for novel experimental platforms based on neutral atom arrays[94]. The MIS problem reads as follows. Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an independent set is a subset of vertices that are not connected with each other. The MIS problem is then the task to find the largest independent set, with its (maximum) cardinality typically denoted as the independence number α. To formulate the MIS problem mathematically, for a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, one first associates a binary variable $x_i \in \{0, 1\}$ with every vertex $i \in V$, with $x_i = 1$ if vertex i belongs to the

Table 1 | Numerical results for MaxCut on Gset instances

Graph	Nodes	Edges	BLS	DSDP	KHLWG	RUN-CSP	PI-GNN	Relative error, ϵ (%)
G14	800	4,694	**3,064**	2,922	3,061	2,943	3,026	0.81
G15	800	4,661	**3,050**	2,938	**3,050**	2,928	2,990	1.29
G22	2,000	19,990	**13,359**	12,960	**13,359**	13,028	13,181	0.89
G49	3,000	6,000	**6,000**	**6,000**	**6,000**	**6,000**	5,918	1.37
G50	3,000	6,000	**5,880**	**5,880**	**5,880**	**5,880**	5,820	1.00
G55	5,000	12,468	**10,294**	9,960	10,236	10,116	10,138	1.25
G70	10,000	9,999	**9,541**	9,456	9,458	—	9,421	1.20

The table reports cut sizes achieved with our physics-inspired GNN solver (PI-GNN), together with results sourced from refs. [61,88–90]. Best known results are shown in bold. The last column specifies the relative error ϵ comparing PI-GNN to the best known cut size. Further details are provided in the main text. GNN model configurations are detailed in the Supplementary Information.

independent set, and $x_i = 0$ otherwise. The MIS can then be formulated in terms of a Hamiltonian that counts the number of marked (coloured) vertices and adds a penalty to non-independent configurations (when two vertices in this set are connected by an edge). It is given by

$$H_{\mathrm{MIS}} = -\sum_{i\in\mathcal{V}} x_i + P\sum_{(i,j)\in\mathcal{E}} x_i x_j, \tag{8}$$

with a negative pre-factor to the first term (because we solve for the largest independent set within a minimization problem) and the penalty parameter $P>0$ enforcing the constraints. Note that the numerical value for P is typically set as $P=2$ (ref. [95]), but can be further optimized in an outer loop. Energetically, the Hamiltonian H_{MIS} favours each variable to be in the state $x_i=1$ unless a pair of these are connected by an edge. Again, the Hamiltonian H_{MIS} is quadratic and falls into the broader class of QUBO problems described by equation (1); again, we provide the explicit Q-matrix for a sample MIS problem in Fig. 2.

The MIS problem is known to be strongly NP-hard, making the existence of an efficient algorithm for finding the maximum independent set on generic graphs unlikely. In addition, the MIS problem is even hard to approximate. In general, the MIS problem cannot be approximated to a constant factor in polynomial time (unless P = NP). Again we study the MIS problem on random unweighted d-regular graphs. Because in our approach the independence constraint is enforced with soft penalty terms $\sim P$ (just like in any QUBO-based model), the predicted set may violate the independence condition (that is, the set may contain nodes connected by an edge). Setting $P=2$, we have observed these violations only in very few cases. If present, as part of our post-processing, we have enforced the independence constraint by greedily removing one of the nodes of each induced edge from the set, and only reporting results after this correction. For small graphs with up to a few hundred nodes, we compare the GNN-based results to results obtained with the Boppana–Halldórsson algorithm built into the Python NetworkX library[96]. For very large graphs with up to a million nodes (where benchmarks are not available) we resort to analytical upper bounds for random d-regular graphs, as presented in ref. [97]. Here, the best known bounds on the ratio α_d/n are reported as $\alpha_3/n = 0.45537$ and $\alpha_5/n = 0.38443$ for $d=3$ and $d=5$, respectively, as derived using refined versions of Markov's inequality[98]. Our results for the achieved independence number as a function of the number of vertices n are shown in Fig. 5. All results are bootstrapped estimates of the mean, with error bars denoting twice the bootstrapped standard deviations, sampled across 20 random d-regular graphs for every data point. Our numerical results for MIS are similar to the observations we have made for MaxCut: for graphs with up to a few hundred nodes, we find that a simple two-layer GCN architecture can perform on par with (or better than) the traditional solver, with the GNN solver showing a favourable runtime scaling. For large graphs with $n\approx 10^4$ to 10^6 nodes we find that our approach consistently achieves high-quality solutions with $\alpha_3/n\approx 0.416$ and $\alpha_5/n\approx 0.338$ for $d=3$ and $d=5$, respectively, resulting in estimated numerical approximation ratios of $0.416/0.45537\approx 0.92$ and $0.338/0.38443\approx 0.88$, respectively. Finally, as shown in Fig. 5, we observe a moderate, super-linear scaling of the total runtime as $\sim n^{1.7}$ for large d-regular graphs with $n\gtrsim 10^5$, as opposed to the Boppana–Halldórsson solver with a runtime scaling of $\sim n^{2.9}$ in the range $n\lesssim 500$. Note that the GNN model training alone displays sub-linear runtime scaling as $\sim n^{0.8}$, in line with our MaxCut results, and the aggregate runtime (including post-processing to enforce the independence condition) scales as $\sim n^{1.7}$ in the regime $n\approx 10^5$ to 10^6.

Applications in industry

Although our previous analysis has focused on canonical graph optimization problems such as maximum cut and maximum independent set, in this section we discuss real-world applications in industry for which our solver could provide solutions, in particular at potentially unprecedented problem scales. We focus on applications of the QUBO formalism, even though our methodology is not limited to this modelling framework. We first review the existing literature, providing relevant references across a wide stack of problem domains. Thereafter, we explicitly show how to distil combinatorial QUBO problems for a few select real-world use cases, from risk diversification in finance to sensor placement problems in water distribution networks (WDNs). Once in QUBO format, the problem can be plugged into our general-purpose physics-inspired GNN solver, as outlined above.

As extensively reviewed in refs. [1,2,66], the QUBO (or, equivalently, Ising) formalism provides a comprehensive modelling framework encompassing a vast array of optimization problems, including knapsack problems, task (resource) allocation problems and capital budgeting problems, among others. Specifically, the applicability of the QUBO representation has been reported for problem settings involving circuit board layouts[99], capital budgeting in financial analysis[100], computer aided design (CAD)[101], electronic traffic management[102,103], cellular radio channel allocation[104], molecular conformation[105] and the prediction of epileptic seizures[106], among others. As mentioned earlier, practical applications of the MaxCut problem can be found in machine scheduling[70], image recognition[71] and electronic circuit layout design[72]. Similarly, in what follows we discuss in detail three select use cases and how they can be cast in QUBO form and thus made amenable to our solver.

Risk diversification. Graphs offer a convenient framework to model portfolio management problems in finance. Specifically, we outline a risk diversification strategy, but similar considerations

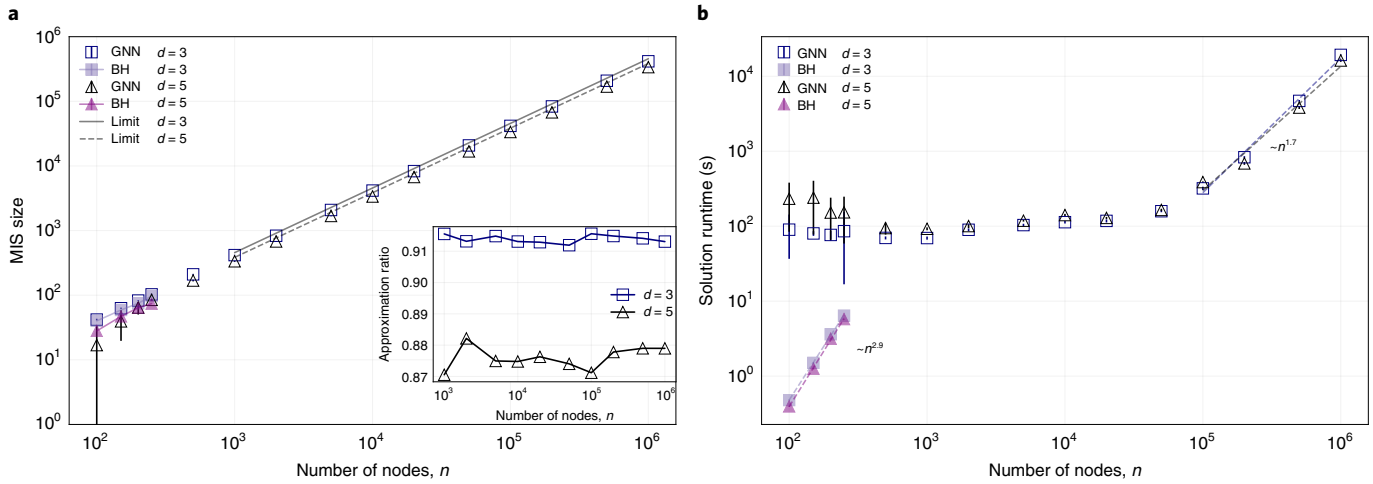

Fig. 5 | Numerical results for the MIS problem. a, Average independence number α for d-regular graphs with $d=3$ and $d=5$ as a function of the number of vertices n, (bootstrap-)averaged over 20 random graph instances, for both the GNN-based method and a traditional MIS algorithm[96]. Solid lines for $n \geq 10^3$ refer to theoretical upper bounds as described in the main text. Inset: the estimated relative approximation ratio comparing the achieved independence number α against known theoretical upper bounds shows that our approach consistently achieves high-quality solutions. **b**, Algorithm runtime, in seconds, for both the GNN solver and the Boppana–Halldórsson algorithm. Error bars refer to twice the bootstrapped standard deviations, sampled across 20 random graph instances for every data point.

apply for the implementation of hedging strategies[107]. We consider a (potentially very large) universe of n assets, for which we are given a vector $\mu \in \mathbb{R}^n$ describing expected future returns, and the covariance matrix $\Sigma \in \mathbb{R}^{n\times n}$ capturing volatility through the correlations among assets. To minimize the volatility of returns of our portfolio, our goal is to select a subset of uncorrelated assets with the largest possible diversified portfolio. To this end we consider a graph $\mathcal{G}$ with n nodes, with every node representing one asset. Correlations can be described in graph form, either by directly taking the cross-correlation matrix as a weighted adjacency matrix, or by creating a binary adjacency matrix A through thresholding. We set $A_{i,j}=1$ if and only if the absolute value of the correlation between assets i and j is greater than some user-specific threshold parameter λ, and $A_{i,j}=0$ otherwise[107]. Accordingly, in our model, pairs of assets are classified as correlated or uncorrelated, based on whether or not the corresponding correlation coefficient exceeds a minimum level. Overall, the risk diversification strategy outlined above can be cast as an optimization problem in QUBO form, with the Hamiltonian

$$H_{\mathrm{wMIS}} = -\sum_{i\in\mathcal{V}} \mu_i x_i + P \sum_{(i,j)\in\mathcal{E}} x_i x_j, \tag{9}$$

in a straightforward extension of equation (8) from the standard MIS problem to the weighted MIS problem. Accordingly, by minimizing the first term of H_{wMIS} we pick large-return assets subject to the independent set constraint (as captured by the second term). This diversification model is reminiscent of the mean-variance Markowitz model[108], albeit in discretized form. Given our results for the standard MIS problem, with small tweaks to the loss function, such a problem could be readily plugged into our solver, for example, as part of a larger, two-stage portfolio management pipeline where first a subset of assets is selected from a larger universe of assets using our solver, and then capital is allocated within a smaller, sparsified basket of assets using off-the-shelf solvers.

Interval scheduling. Here we consider a scenario involving the scheduling of tasks with given start and end times, as relevant, for example, in the context of algorithm design in computer science[109]. Specifically, we face n resource requests, each represented by an interval specifying the time in which it needs to be processed by some machine. Typically, some requests will overlap in time, leading to request clashes that cannot be satisfied by the same machine (resource). Conversely, a subset of intervals is deemed compatible if no two intervals overlap on the machine. As is commonly done in resource allocation problems and scheduling theory, this situation can conveniently be described with the help of an undirected interval graph $\mathcal{G}$ in which we introduce a vertex for each request and edges between vertices whose requests overlap. With the goal to maximize the throughput (that is, to execute as many tasks as possible on a single machine), the interval scheduling maximization problem is then to find the largest compatible set, that is, a set of non-overlapping intervals of maximum size. This use case is equivalent to finding the maximum independent set in the corresponding interval graph $\mathcal{G}$ with n nodes. Although inexpensive (special-purpose) algorithms exist for interval graphs[110], we can then solve the underlying MIS problem, as described by equation (8), on this interval graph, in the same way as any other QUBO problem, using our general-purpose GNN-based approach, following the methodology outlined above.

Sensor placement in WDNs. Optimal sensor placement is key to the detection and isolation of fault events—such as water leaks—in WDNs[111]. As detailed in ref. [111], the problem of optimally placing pressure sensors on a WDN can be efficiently cast as a QUBO problem. Specifically, a WDN can be readily mapped to a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$, with nodes $i\in\mathcal{V}$ referring to tanks or junctions, and edges $(i,j)\in\mathcal{E}$ representing pipes. We then associate a binary variable $x_i=0,1$ with every node, and set $x_i=1$ if node i hosts a sensor and $x_i=0$ otherwise. The problem of covering the WDN with the smallest possible number of pressure sensors then maps onto the minimum vertex cover problem[111], as described by the Hamiltonian

$$H_{\mathrm{MVC}} = \sum_{i\in\mathcal{V}} c_i x_i + P \sum_{(i,j)\in\mathcal{E}} (1 - x_i - x_j + x_i x_j). \tag{10}$$

Here $c_i \geq 0$ denotes the cost of node i hosting a sensor, the first term describes the overall cost of any potential sensor placement strategy, and the second (penalty) term with $P>0$ ensures the constraint

$x_i + x_j \geq 1$ for all edges $(i, j) \in \mathcal{E}$ (that is, at least one of the endpoints of each edge will be in the cover[1]). Potential tweaks to this model are detailed in ref. [111]; however, variations can all be represented as a QUBO problem. Along the lines of our previous analysis for the MaxCut or MIS problem, the Hamiltonian H_{MVC} (or some variation thereof) can be straightforwardly mapped to a relaxed loss function with which we can train our solver and then solve the corresponding sensor placement use case.

Conclusion and outlook

In summary, we have proposed and analysed a versatile and scalable general-purpose solver that is powered by GNNs and draws from concepts in statistical physics. Our approach is applicable to any k-local Ising model, including canonical NP-hard combinatorial optimization problems such as the maximum cut, maximum clique, minimum vertex cover or maximum independent set problems, among others[66]. Starting from a problem formulation in Ising form, we apply a relaxation strategy to the problem Hamiltonian by dropping integrality constraints on the decision variables to generate a differentiable loss function with which we perform unsupervised training on the node representations of the GNN. The GNN is then trained to generate soft assignments to predict the likelihood of belonging in one of two classes, for each vertex in the graph. To find a binary (two-colour) labelling consistent with the original problem formulation, simple projection heuristics are applied. Overall, we find that this approach can compete with existing special-purpose solvers, such as the Goemans–Williamson algorithm designed to solve the maximum cut problem, with the potential to tap into the rich toolbox of statistical physics, including, for example, the study of phase transitions. In the current noisy intermediate-scale quantum era, our approach could be used as a broadly applicable, scalable benchmark for emerging quantum technologies, including special-purpose quantum[6] and quantum-inspired annealers[20], while not being resource-constrained nor being limited to problem instances in the QUBO form, as is also the case for coherent Ising machines[112].

Finally, we highlight possible extensions of research going beyond our present work. First, to better understand the limitations of GNNs in the context of combinatorial optimization, further studies are in order, systematically benchmarking GNNs against state-of-the-art solvers for a large class of optimization problems while leveraging the entire zoo of GNN implementations including, for example, GraphSAGE[27] or graph attention networks (GATs)[42] to potentially boost the GNN Ansatz with an attention mechanism enabling vertices to weigh neighbour representations during the aggregation steps. Second, the presented GNN approach should be able to accommodate problems sizes with hundreds of millions of nodes when leveraging distributed training in a mini-batch fashion on a cluster of machines[45], thereby challenging the capabilities of several existing solvers. Although we have solved individual problem instances from scratch, using a random initialization process for the initial node embeddings, in the future, warm-starting the training process with pre-trained weights (transfer learning) could boost the time to solution. Moreover, one could potentially boost the performance of our optimizer by implementing randomized projection schemes (as opposed to the simple deterministic approach used here) or augment these strategies with simple greedy post-processing routines that check for local optimality with a sequence of local bit flips. Finally, as discussed in the main text, our approach can be generalized to PUBO problems on hypergraphs where so-called hyper-edges may contain more than just two nodes, with no need for (typically) resource-intensive degree reduction schemes, as opposed to resource-constrained QUBO solvers. Potential applications cover many real-world optimization problems involving multi-body interactions, as found in scheduling problems[113] or chemistry[114,115]. In conclusion, the proposed cross-fertilization between machine learning, operations research and physics opens up a number of interesting research directions, with the ultimate goal to further advance our ability to solve hard combinatorial optimization problems.

Data availability

The data necessary to reproduce our numerical benchmark results are publicly available at https://web.stanford.edu/~yyye/yyye/Gset/. Random d-regular graphs have been generated using the open-source networkx library (https://networkx.org).

Code availability

An end-to-end open source demo version of the code implementing our approach has been made publicly available at https://github.com/amazon-research/co-with-gnns-example[116].

Received: 9 July 2021; Accepted: 21 February 2022;
Published online: 21 April 2022

References

1. Glover, F., Kochenberger, G. & Du, Y. Quantum bridge analytics I: a tutorial on formulating and using QUBO models. *4OR* **17**, 335 (2019).
2. Kochenberger, G. et al. The unconstrained binary quadratic programming problem: a survey. *J. Comb. Optim.* **28**, 58–81 (2014).
3. Anthony, M., Boros, E., Crama, Y. & Gruber, A. Quadratic reformulations of nonlinear binary optimization problems. *Math. Program.* **162**, 115–144 (2017).
4. Papadimitriou, C. H. & Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity* (Courier Corporation, 1998).
5. Korte, B. & Vygen, J. *Combinatorial Optimization* Vol. 2 (Springer, 2012).
6. Johnson, M. W. et al. Quantum annealing with manufactured spins. *Nature* **473**, 194–198 (2011).
7. Bunyk, P. et al. Architectural considerations in the design of a superconducting quantum annealing processor. *IEEE Trans. Appl. Supercond.* **24**, 1–10 (2014).
8. Katzgraber, H. G. Viewing vanilla quantum annealing through spin glasses. *Quantum Sci. Technol.* **3**, 030505 (2018).
9. Hauke, P., Katzgraber, H. G., Lechner, W., Nishimori, H. & Oliver, W. Perspectives of quantum annealing: methods and implementations. *Rep. Prog. Phys.* **83**, 054401 (2020).
10. Kadowaki, T. & Nishimori, H. Quantum annealing in the transverse Ising model. *Phys. Rev. E* **58**, 5355 (1998).
11. Farhi, E. et al. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science* **292**, 472–475 (2001).
12. Mandrà, S., Zhu, Z., Wang, W., Perdomo-Ortiz, A. & Katzgraber, H. G. Strengths and weaknesses of weak-strong cluster problems: a detailed overview of state-of-the-art classical heuristics versus quantum approaches. *Phys. Rev. A* **94**, 022337 (2016).
13. Mandrà, S. & Katzgraber, H. G. A deceptive step towards quantum speedup detection. *Quantum Sci. Technol.* **3**, 04LT01 (2018).
14. Barzegar, A., Pattison, C., Wang, W. & Katzgraber, H. G. Optimization of population annealing Monte Carlo for large-scale spin-glass simulations. *Phys. Rev. E* **98**, 053308 (2018).
15. Hibat-Allah, M. et al. Variational neural annealing. *Nat. Mach. Intell.* **3**, 952–961 (2021).
16. Wang, Z., Marandi, A., Wen, K., Byer, R. L. & Yamamoto, Y. Coherent Ising machine based on degenerate optical parametric oscillators. *Phys. Rev. A* **88**, 063853 (2013).
17. Hamerly, R. et al. Scaling advantages of all-to-all connectivity in physical annealers: the coherent Ising machine vs. D-wave 2000Q. *Sci. Adv.* **5**, eaau0823 (2019).
18. Di Ventra, M. & Traversa, F. L. Perspective: memcomputing: leveraging memory and physics to compute efficiently. *J. Appl. Phys.* **123**, 180901 (2018).
19. Traversa, F. L., Ramella, C., Bonani, F. & Di Ventra, M. Memcomputing NP-complete problems in polynomial time using polynomial resources and collective states. *Sci. Adv.* **1**, e1500031 (2015).
20. Matsubara, S. et al. in *Complex, Intelligent and Software Intensive Systems (CISIS-2017)* (eds Terzo, O. & Barolli, L.) 432–438 (Springer, 2017).
21. Tsukamoto, S., Takatsu, M., Matsubara, S. & Tamura, H. An accelerator architecture for combinatorial optimization problems. *FUJITSU Sci. Tech. J.* **53**, 8–13 (2017).
22. Aramon, M., Rosenberg, G., Miyazawa, T., Tamura, H. & Katzgraber, H. G. Physics-inspired optimization for constraint-satisfaction problems using a digital annealer. *Front. Phys.* **7**, 48 (2019).

23. Gori, M., Monfardini, G. & Scarselli, F. A new model for learning in graph domains. In *Proc. 2005 IEEE International Joint Conference on Neural Networks* Vol. 2 729–734 (IEEE, 2005).
24. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The graph neural network model. *IEEE Trans. Neural Netw.* **20**, 61–80 (2008).
25. Micheli, A. Neural network for graphs: a contextual constructive approach. *IEEE Trans. Neural Netw.* **20**, 498–511 (2009).
26. Duvenaud, D. K. et al. Convolutional networks on graphs for learning molecular fingerprints. *Adv. Neural Inf. Process. Syst.* **28**, 2224–2232 (2015).
27. Hamilton, W. L., Ying, R. & Leskovec, J. Representation learning on graphs: methods and applications. Preprint at https://arxiv.org/abs/1709.05584 (2017).
28. Xu, K., Hu, W., Leskovec, J. & Jegelka, S. How powerful are graph neural networks? Preprint at https://arxiv.org/abs/1810.00826 (2018).
29. Kipf, T. N. & Welling, M. Semi-supervised classification with graph convolutional networks. Preprint at https://arxiv.org/abs/1609.02907 (2016).
30. Wu, Z. et al. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw.* **32**, 4–24 (2021).
31. Perozzi, B., Al-Rfou, R. & Skiena, S. Deepwalk: online learning of social representations. In *Proc. 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 701–710 (ACM, 2014).
32. Sun, Z., Deng, Z. H., Nie, J. Y. & Tang, J. Rotate: knowledge graph embedding by relational rotation in complex space. Preprint at https://arxiv.org/abs/1902.10197 (2019).
33. Ying, R. et al. Graph convolutional neural networks for web-scale recommender systems. In *Proc. 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)* 974–983 (ACM, 2018).
34. Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A. & Kim, P. M. Fast and flexible protein design using deep graph neural networks. *Cell Syst.* **11**, 402–411 (2020).
35. Gaudelet, T. et al. Utilising graph machine learning within drug discovery and development. Preprint at https://arxiv.org/abs/2012.05716 (2020).
36. Pal, A. et al. Pinnersage: multi-modal user embedding framework for recommendations at Pinterest. In *Proc. 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* 2311–2320 (ACM, 2020).
37. Rossi, E. et al. Temporal graph networks for deep learning on dynamic graphs. Preprint at https://arxiv.org/abs/2006.10637 (2020).
38. Monti, F., Frasca, F., Eynard, D., Mannion, D. & Bronstein, M. Fake news detection on social media using geometric deep learning. Preprint at https://arxiv.org/abs/1902.06673 (2019).
39. Choma, N. et al. Graph neural networks for icecube signal classification. In *17th IEEE International Conference on Machine Learning and Applications (ICMLA)* 386–391 (IEEE, 2018).
40. Shlomi, J., Battaglia, P. & Vlimant, J.-R. Graph neural networks in particle physics. *Mach. Learn. Sci. Technol.* **2**, 021001 (2020).
41. Li, Y., Tarlow, D., Brockschmidt, M. & Zemel, R. Gated graph sequence neural networks. Preprint at https://arxiv.org/abs/1511.05493 (2015).
42. Veličković, P. et al. Graph attention networks. Preprint at https://arxiv.org/abs/1710.10903 (2017).
43. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning* 1263–1272 (PMLR, 2017).
44. Xu, K. et al. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning* 5453–5462 (PMLR, 2018).
45. Zheng, D. et al. DistDGL: distributed graph neural network training for billion-scale graphs. In *IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)* 36–44 (IEEE, 2020).
46. Kotary, J., Fioretto, F., Van Hentenryck, P. & Wilder, B. End-to-end constrained optimization learning: a survey. Preprint at https://arxiv.org/abs/2103.16378 (2021).
47. Cappart, Q. et al. Combinatorial optimization and reasoning with graph neural networks. Preprint at https://arxiv.org/abs/2102.09544 (2021).
48. Mills, K., Ronagh, P. & Tamblyn, I. Finding the ground state of spin Hamiltonians with reinforcement learning. *Nat. Mach. Intell.* **2**, 509–517 (2020).
49. Vinyals, O., Fortunato, M. & Jaitly, N. Pointer networks. *Adv. Neural Inf. Process. Syst.* **28**, 2692–2700 (2015).
50. Nowak, A., Villar, S., Bandeira, A. S. & Bruna, J. Revised note on learning algorithms for quadratic assignment with graph neural networks. Preprint at https://arxiv.org/abs/1706.07450 (2017).
51. Bai, Y. et al. SimGNN: a neural network approach to fast graph similarity computation. Preprint at https://arxiv.org/abs/1808.05689 (2018).
52. Lemos, H., Prates, M., Avelar, P. & Lamb, L. Graph colouring meets deep learning: effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)* 879–885 (IEEE, 2019).
53. Li, Z., Chen, Q. & Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *Proc. NeurIPS* 536–545 (2018).
54. Joshi, C. K., Laurent, T. & Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. Preprint at https://arxiv.org/abs/1906.01227 (2019).
55. Karalias, N. & Loukas, A. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. Preprint at https://arxiv.org/abs/2006.10643 (2020).
56. Yehuda, G., Gabel, M. & Schuster, A. It's not what machines can learn, it's what we cannot teach. Preprint at https://arxiv.org/abs/2002.09398 (2020).
57. Bello, I., Pham, H., Le, Q. V., Norouzi, M. & Bengio, S. Neural combinatorial optimization with reinforcement learning. Preprint at https://arxiv.org/abs/1611.09940 (2017).
58. Kool, W., van Hoof, H. & Welling, M. Attention, learn to solve routing problems! Preprint at https://arxiv.org/abs/1803.08475 (2019).
59. Ma, Q., Ge, S., He, D., Thaker, D. & Drori, I. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. Preprint at https://arxiv.org/abs/1911.04936 (2019).
60. Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. & Song, L. Learning combinatorial optimization algorithms over graphs. In *Annual Conference on Neural Information Processing Systems (NIPS)* 6351–6361 (2017).
61. Toenshoff, J., Ritzert, M., Wolf, H. & Grohe, M. RUN-CSP: unsupervised learning of message passing networks for binary constraint satisfaction problems. Preprint at https://arxiv.org/abs/1909.08387 (2019).
62. Yao, W., Bandeira, A. S. & Villar, S. Experimental performance of graph neural networks on random instances of max-cut. In *Wavelets and Sparsity XVIII* Vol. 11138 111380S (International Society for Optics and Photonics, 2019).
63. Ising, E. Beitrag zur Theorie des Ferromagnetismus. *Z. Phys.* **31**, 253–258 (1925).
64. Matsubara, S., et al. Ising-model optimizer with parallel-trial bit-sieve engine. In *Conference on Complex, Intelligent, and Software Intensive Systems* 432–438 (Springer, 2017).
65. Hamerly, R. et al. Experimental investigation of performance differences between coherent Ising machines and a quantum annealer. *Sci. Adv.* **5**, eaau0823 (2019).
66. Lucas, A. Ising formulations of many NP problems. *Front. Phys.* **2**, 5 (2014).
67. Alon, U. & Yahav, E. On the bottleneck of graph neural networks and its practical implications. Preprint at https://arxiv.org/abs/2006.05205 (2020).
68. Fey, M. & Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. Preprint at https://arxiv.org/abs/1903.02428 (2019).
69. Wang, M. et al. Deep Graph Library: a graph-centric, highly-performant package for graph neural networks. Preprint at https://arxiv.org/abs/1909.01315 (2019).
70. Alidaee, B., Kochenberger, G. A. & Ahmadian, A. 0-1 Quadratic programming approach for optimum solutions of two scheduling problems. *Int. J. Syst. Sci.* **25**, 401–408 (1994).
71. Neven, H., Rose, G. & Macready, W. G. Image recognition with an adiabatic quantum computer I. Mapping to quadratic unconstrained binary optimization. Preprint at https://arxiv.org/abs/0804.4457 (2008).
72. Deza, M. & Laurent, M. Applications of cut polyhedra. *J. Comput. Appl. Math.* **55**, 191–216 (1994).
73. Farhi, E., Goldstone, J. & Gutmann, S. A. A quantum approximate optimization algorithm. Preprint at https://arxiv.org/abs/1411.4028 (2014).
74. Zhou, L., Wang, S.-T., Choi, S., Pichler, H. & Lukin, M. D. Quantum approximate optimization algorithm: performance, mechanism and implementation on near-term devices. *Phys. Rev. X* **10**, 021067 (2020).
75. Guerreschi, G. G. & Y., A. QAOA for Max-Cut requires hundreds of qubits for quantum speed-up. *Nat. Sci. Rep.* **9**, 6903 (2019).
76. Crooks, G. E. Performance of the quantum approximate optimization algorithm on the maximum cut problem. Preprint at https://arxiv.org/abs/1811.08419 (2018).
77. Lotshaw, P. C. et al. Empirical performance bounds for quantum approximate optimization. *Quantum Inf. Process.* **20**, 403 (2021).
78. Patti, T. L., Kossaifi, J., Anandkumar, A. & Yelin, S. F. Variational Quantum Optimization with Multi-Basis Encodings. Preprint at https://arxiv.org/abs/2106.13304 (2021).
79. Zhao, T., Carleo, G., Stokes, J. & Veerapaneni, S. Natural evolution strategies and variational Monte Carlo. *Mach. Learn. Sci. Technol.* **2**, 02LT01 (2020).
80. Goemans, M. X. & Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* **42**, 1115–1145 (1995).
81. Halperin, E., Livnat, D. & Zwick, U. MAX CUT in cubic graphs. *J. Algorithms* **53**, 169–185 (2004).
82. Dembo, A., Montanari, A. & Sen, S. Extremal cuts of sparse random graphs. *Ann. Probab.* **45**, 1190–1217 (2017).
83. Sherrington, D. & Kirkpatrick, S. Solvable model of a spin glass. *Phys. Rev. Lett.* **35**, 1792–1795 (1975).

84. Binder, K. & Young, A. P. Spin glasses: experimental facts, theoretical concepts and open questions. *Rev. Mod. Phys.* **58**, 801–976 (1986).
85. Alizadeh, F. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optimization* **5**, 13 (1995).
86. Haribara, Y. & Utsunomiya, S. in *Principles and Methods of Quantum Information Technologies. Lecture Notes in Physics* Vol. 911 (eds Semba, K. & Yamamoto, Y.) 251–262 (Springer, 2016).
87. Ye, Y. *The Gset Dataset* (Stanford, 2003); https://web.stanford.edu/~yyye/yyye/Gset/
88. Kochenberger, G. A., Hao, J.-K., Lu, Z., Wang, H. & Glover, F. Solving large scale Max Cut problems via tabu search. *J. Heuristics* **19**, 565–571 (2013).
89. Benlic, U. & Hao, J.-K. Breakout local search for the Max-Cut problem. *Eng. Appl. Artif. Intell.* **26**, 1162–1173 (2013).
90. Choi, C. & Ye, Y. *Solving Sparse Semidefinite Programs Using the Dual Scaling Algorithm with an Iterative Solver* Working Paper (Department of Management Sciences, Univ. Iowa, 2000).
91. Hale, W. K. Frequency assignment: theory and applications. *Proc. IEEE* **68**, 1497–1514 (1980).
92. Boginski, V., Butenko, S. & Pardalos, P. M. Statistical analysis of financial networks. *Comput. Stat. Data Anal.* **48**, 431–443 (2005).
93. Yu, H., Wilczek, F. & Wu, B. Quantum algorithm for approximating maximum independent sets. *Chin. Phys. Lett.* **38**, 030304 (2021).
94. Pichler, H., Wang, S.-T., Zhou, L., Choi, S. & Lukin, M. D. Quantum optimization for maximum independent set using Rydberg atom arrays. Preprint at https://arxiv.org/abs/1808.10816 (2018).
95. Djidjev, H. N., Chapuis, G., Hahn, G. & Rizk, G. Efficient combinatorial optimization using quantum annealing. Preprint at https://arxiv.org/abs/1801.08653 (2018).
96. Boppana, R. & Halldórsson, M. M. Approximating maximum independent sets by excluding subgraphs. *BIT Numer. Math.* **32**, 180–196 (1992).
97. Duckworth, W. & Zito, M. Large independent sets in random regular graphs. *Theor. Comput. Sci.* **410**, 5236–5243 (2009).
98. McKay, B. D. Independent sets in regular graphs of high girth. *Ars Combinatoria* **23A**, 179 (1987).
99. Grötschel, M., Jünger, M. & Reinelt, G. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.* **36**, 493–513 (1988).
100. Laughhunn, D. J. Quadratic binary programming with application to capital-budgeting problems. *Oper. Res.* **18**, 454–461 (1970).
101. Krarup, J. & Pruzan, A. Computer aided layout design. *Math. Program. Study* **9**, 75–94 (1978).
102. Gallo, G., Hammer, P. & Simeone, B. Quadratic knapsack problems. *Math. Program.* **12**, 132–149 (1980).
103. Witsgall, C. *Mathematical Methods of Site Selection for Electronic System (EMS)* NBS Internal Report (NBS, 1975).
104. Chardaire, P. & Sutter, A. A decomposition method for quadratic zero-one programming. *Manag. Sci.* **41**, 704–712 (1994).
105. Phillips, A. & Rosen, J. B. A quadratic assignment formulation of the molecular conformation problem. *J. Glob. Optim.* **4**, 229–241 (1994).
106. Iasemidis, L. D. et al. Prediction of human epileptic seizures based on optimization and phase changes of brain electrical activity. *Optim. Methods Software* **18**, 81–104 (2003).
107. Kalra, A., Qureshi, F. & Tisi, M. Portfolio asset identification using graph algorithms on a quantum annealer. *SSRN* https://ssrn.com/abstract=3333537 (2018).
108. Markowitz, H. Portfolio selection. *J. Finance* **7**, 77–91 (1952).
109. Kolen, A. Interval scheduling: a survey. *Naval Res. Logist.* **54**, 530–543 (2007).
110. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J. & Schieber, B. A unified approach to approximating resource allocation and scheduling. *J. ACM* **48**, 1069–1090 (2001).
111. Speziali, S. et al. Solving sensor placement problems in real water distribution networks using adiabatic quantum computation. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)* 463–464 (IEEE, 2021).
112. McMahon, P. L. et al. A fully programmable 100-spin coherent Ising machine with all-to-all connections. *Science* **354**, 614–617 (2016).
113. Bansal, N. & Khot, S. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In *Proc. Automata, Languages and Programming (ICALP 2010) Lecture Notes in Computer Science* Vol. 6198 (eds Abramsky, S. et al.) 250–261 (Springer, 2010).
114. Hernandez, M., Zaribafiyan, A., Aramon, M. & Naghibi, M. A novel graph-based approach for determining molecular similarity. Preprint at https://arxiv.org/abs/1601.06693 (2016).
115. Terry, J. P., Akrobotu, P. D., Negre, C. F. & Mniszewski, S. M. Quantum isomer search. *PLoS ONE* **15**, e0226787 (2020).
116. Combinatorial optimization with graph neural networks. *GitHub* https://github.com/amazon-research/co-with-gnns-example (2022).

Acknowledgements

We thank F. Brandao, G. Karypis, M. Kastoryano, E. Kessler, T. Mullenbach, N. Pancotti, M. Resende, S. Roy, G. Salton, S. Severini, A. Urweisse and J. Zhu for fruitful discussions.

Author contributions

All authors contributed to the ideation and design of the research. M.J.A.S. and J.K.B. developed and ran the computational experiments and also wrote the initial draft of the the manuscript. H.G.K. supervised this work and revised the manuscript.

Competing interests

M.J.A.S., J.K.B. and H.G.K. are listed as inventors on a US provisional patent application (no. 7924-38500) on combinatorial optimization with graph neural networks.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-022-00468-6.

Correspondence and requests for materials should be addressed to Martin J. A. Schuetz, J. Kyle Brubaker or Helmut G. Katzgraber.

Peer review information *Nature Machine Intelligence* thanks Thomas Vandal and Estelle Inack for their contribution to the peer review of this work.