Combinatorial optimization with physics-inspired graph neural networks

Martin J. A. Schuetz, J. Kyle Brubaker and Helmut G. Katzgraber

Huang Nan

Peking University

June 16, 2022

Outline

Outline

Combinatorial optimization

Combinatorial optimization is to search for the **minimum of an objective function** within a **finite buy large set of cadidate solutions**.
Formally[1], a combinatorial optimization problem A is a quadruple (I, f, m, g), where

1. I is a set of instances
2. $\forall a \in I$, $f(a)$ is the finite set of feasible solutions
3. $\forall a \in I, x \in f(a)$, $m(a, x)$ is the measure of solution x
4. g is the goal function (either $\min$ or $\max$)

And the goal is to find the optimal solution y given x such that

$$m(a, x) = g\{m(a, x^*) \mid x^* \in f(a)\}$$

Binary optimization

Binary optimization (BO) is a subclass of combinatorial optimization problems where the variables are restricted to a finite set of two values. And this can be described as

$$\arg\min_{\vec{x}} \left\{ H_a\left(\vec{x}\right) \mid \vec{x} \in \{0,1\}^N \right\}$$

subject to equality constraints $f(\vec{x}) = 0$ and inequality constraints $g(\vec{x}) > 0$, where the $H_a(\cdot)$ is the objective function given certain instance a.

Polynomial unconstrained binary optimization

Polynomial unconstrained binary optimizations (PUBOs) are a subset of BO problems with no constraints, and its objective function is a polynomial of the variables as follow

$$
\begin{aligned}
H(\vec{x}) = & \sum_{i_1}^{N} P_{i_1} \cdot x_{i_1} \\
& + \frac{1}{2!} \sum_{i_1}^{N} \sum_{i_2}^{N} P_{i_1 i_2} \cdot x_{i_1} x_{i_2} \\
& \vdots \\
& + \frac{1}{k!} \sum_{i_1}^{N} \cdots \sum_{i_k}^{N} P_{i_1 \cdots i_k} \cdot x_{i_1} \cdots x_{i_k}
\end{aligned}
$$

where the coupling constant P denotes the n-point interactions between variables, which can be interpreted as **(hyper)edges** of underlying **(hyper)graph**.

Quadratic unconstrained binary optimization

Quadratic unconstrained binary optimizations (QUBOs) are special cases of PUBOs, where the order of polynomial is limited to 2. The objective function is given by

$$H(\vec{x}) = \sum_{i_1}^{N} Q_{i_1} \cdot x_{i_1} + \frac{1}{2} \sum_{i_1}^{N} \sum_{i_2}^{N} Q_{i_1 i_2} \cdot x_{i_1} x_{i_2}$$

When mapping x to $z = 2x - 1$, this is equivalent to the canonical form of *Ising model* where each spin is either $+1$ or -1.

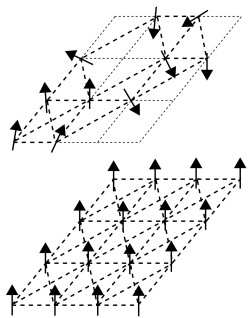


Figure: Schematic representation of the random spin structure of a spin glass (top) and the ordered one of a ferromagnet (bottom)

Source: `https://en.wikipedia.org/wiki/Spin_glass`

Examples of QUBO: Maximum cut

We want to find a subset of $\mathcal{V}$ and divide the whole graph into a bipartite graph maximizing edges in between.
In the sense of Hamiltonian, given that node assignment $x \in \{0, 1\}$, we can define the objective function as

$$H_{MaxCut} = \sum_{i<j} A_{ij} (2x_i x_j - x_i - x_j)$$
$$\equiv \sum_{i<j} J_{ij} z_i z_j$$

where $2x_i x_j - x_i - x_j$ is the math form of logic *XOR*, and $J = A/2 > 0$ favoring anti-ferromeganetic ordering.

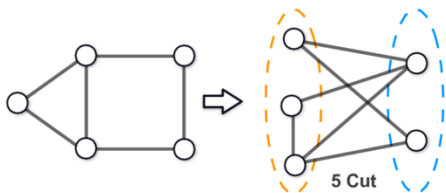


Figure: Illustration of the maximum cut problem

Source: https://www.researchgate.net/publication/324472130_Quantum_Algorithm_Implementations_for_Beginners

Examples of QUBO: Maximum independent set

We want to find a maximum subset of $\mathcal{V}$ that are not connected with each other.
In the sense of Hamiltonian, given that node assignment $x \in \{0, 1\}$, we introduce a penalty term P to the objective function as

$$H_{MIS} = -\sum_i x_i + P \sum_{i<j} A_{ij} x_i x_j$$

where $x_i x_j$ is the math form of logic *AND*, and empirically setting $P = 2$.

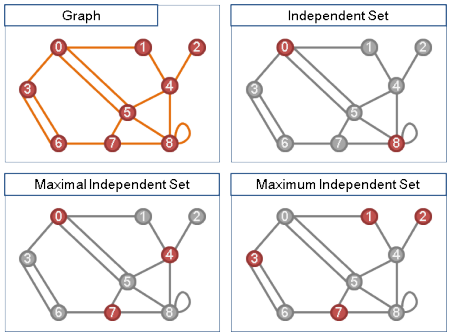


Figure: Illustration of the maximum independent set problem

Source: `https://web.ntnu.edu.tw/~algo/Domination.html`

Examples of QUBO: Minimum vertex cover

We want to find a minimum subset of $\mathcal{V}$ that connects to all the edges $\mathcal{E}$.
In the sense of Hamiltonian, given that node assignment $x \in \{0, 1\}$, we introduce a penalty term P to the objective function as

$$H_{MVC} = \sum_i x_i + P \sum_{i<j} A_{ij}(1 - x_i - x_j + x_i x_j)$$

where $1 - x_i - x_j + x_i x_j$ is the math form of logic *NOR*.

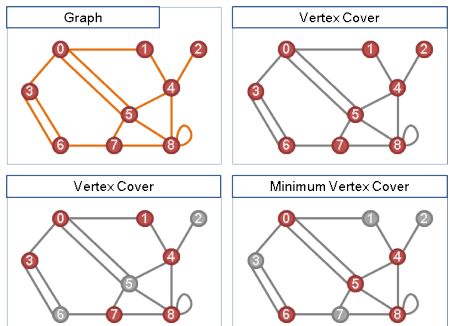


Figure: Illustration of the maximum vertex cover problem

Source: https://web.ntnu.edu.tw/~algo/Domination.html

Solutions to combinatorial optimization

Previous examples are non-deterministic polynomial-time hard problems (*NP*-hard for short), and current methods focus on the following aspects:

1. approximation algorithms for specific problem that run in P-time
2. **Quantum annealing algorithms** and **simulated quantum annealing algorithms** that attempt to find the global optimum in P-time or even in constant time

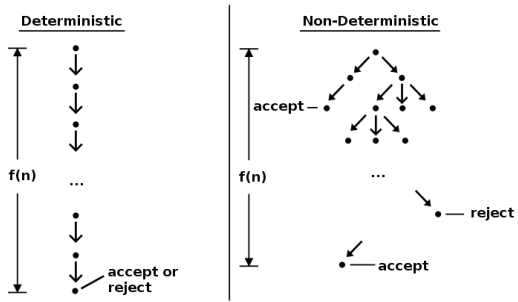


Figure: Comparison of deterministic and nondeterministic computation

Source: https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine

Machine learning approaches

Despite handcrafting algorithms that solve certain types of combinatorial optimization problems, we have seen that machine learning can be used for approximation as well, yet usually are not universel.

Supervised learning

1. empirically selected **handcrafted loss functions**
2. seq2seq models & graph neural networks
3. relies on **large, labelled training datasets (preferably unbiased and representative)**

Reinforcement learning

1. optimization problem described as native objective function
2. graph attention network
3. **currently capped at** $\sim 1,000 - 1,200$ **nodes**

Unsupervised learning

1. empirically selected **handcrafted loss functions**
2. maximum constraint satisfaction model
3. **currently capped at** $\sim 5,000$ **nodes**

Outline

Why working with graph

1. The graph nature of the combinatorial problems we are interested in
2. Graph are permutation-invariant (i.e. the permutation of the node orders will not affect the solution), which cannot be preserved by RNN (which assumes time-invariant) and CNN (which assumes space-invariant)
3. Scalability of such architectures (i.e. cost of computation on graph grows almost linearly with graph size, achieved by matrix multiplication)

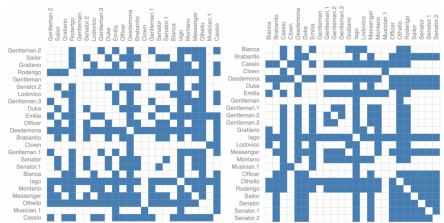


Figure: Two adjacency matrices representing the same graph

Source: https://distill.pub/2021/gnn-intro/

Brief introduction to graph neural networks

In essence, graph neural networks (GNNs) is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries.[2]

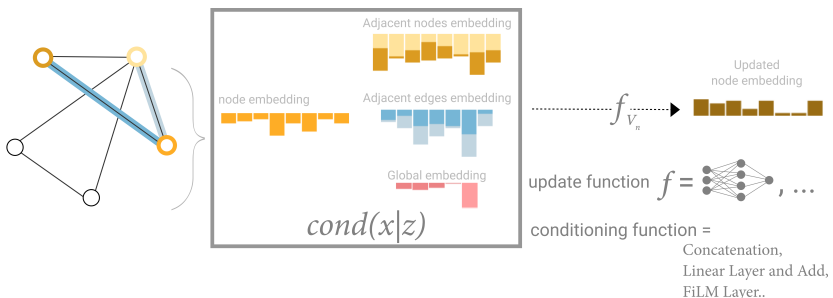


Figure: Schematic for conditioning the information of one node based on three other embeddings

Source: https://distill.pub/2021/gnn-intro/

Learning node representation on graph

Take graph convolutional networks (GCNs) for example, they typically are composed of three functions

1. Message passing: exchanges information with neighboring nodes through edges
2. Aggregation: combines the received messages into a single representation
3. Update activation: transforms the representation (usually non-linearly)

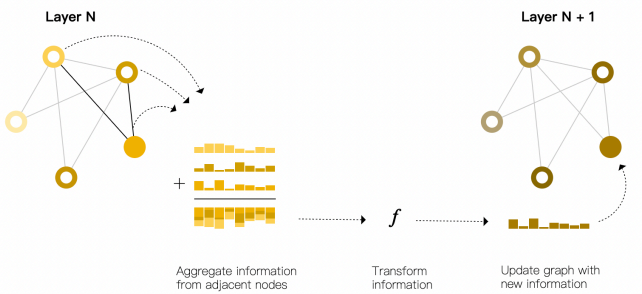


Figure: Schematic representation of a GCN layer

Source: https://distill.pub/2021/gnn-intro/

Common form

In general, the compution done by GNNs is usually in the following form

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

for the layers $k = 1, \cdots, K$ with
$\mathcal{N}_{\nu} = \{u \in \mathcal{V} \mid (u, \nu) \in \mathcal{E}\}$ denoting the neighbors, where each node representation $\mathbf{h}_{\nu}^{k} \in \mathbb{R}^{d_k}$ and the parametric function f_{θ}^{k} is learnable.
Note that single layer of such only encapsulates its immediate neighborhood, but we can broaden the receptive field by stacking multiple layers.

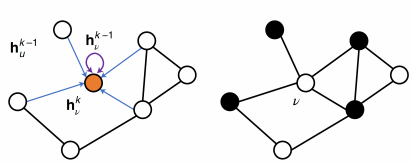


Figure: Schematic of a GNN operation

Graph convolutional networks

GCN is close to the convolutional layers as given by the following updating equation

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

where $\mathbf{W}$ and $\mathbf{B}$ are trainable parameters that are shared by all the nodes, $\frac{1}{\mathcal{N}_\nu}$ being a normalization factor (equivalent to taking the mean of neighboring node features) and $\sigma(\cdot)$ is a non-linear activation function such as *sigmoid* or *ReLu*.

Graph attention networks

Similar to attention mechanism in *Transformer*, we can also implement attention mechanism for nodes in graph.

$$\mathbf{h}_{\nu}^{k} = \alpha_{\nu\nu}\mathbf{W}_{k}\mathbf{h}_{\nu}^{k-1} + \sum_{u\in\mathcal{N}_{\nu}} \alpha_{\nu u}\mathbf{W}_{k}\mathbf{h}_{u}^{k-1}$$

$$\text{where } \alpha_{\nu u} = \frac{\exp\left(\sigma\left(\mathbf{a}^{T}\left[\mathbf{W}_{k}\mathbf{h}_{\nu}^{k-1} \| \mathbf{W}_{k}\mathbf{h}_{u}^{k-1}\right]\right)\right)}{\sum_{w\in\mathcal{N}_{\nu}\cup\{\nu\}} \exp\left(\sigma\left(\mathbf{a}^{T}\left[\mathbf{W}_{k}\mathbf{h}_{\nu}^{k-1} \| \mathbf{W}_{k}\mathbf{h}_{w}^{k-1}\right]\right)\right)}$$

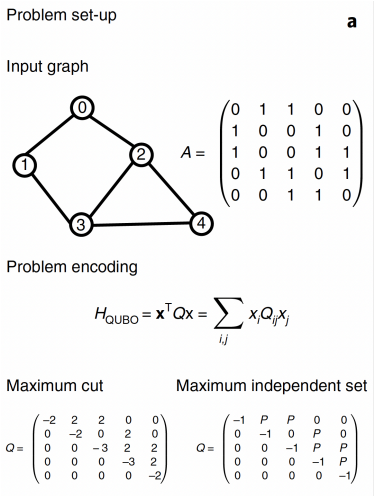
where $\mathbf{W}$ and $\mathbf{a}$ are trainable parameters that are shared by all the nodes, and $\alpha_{\nu u}$ is the attention coefficient calculated through *softmax*.

Outline

Problem setup

1. the graph input is given by adjacency matrix A
2. the optimization problem is encoded in Q

Note that the linear term in QUBO problem is always absorbed into Q as $x_i^2 = x_i$ holds for $x_i \in \{0, 1\}$, and problem constraints can be expressed in Q as penalty terms.

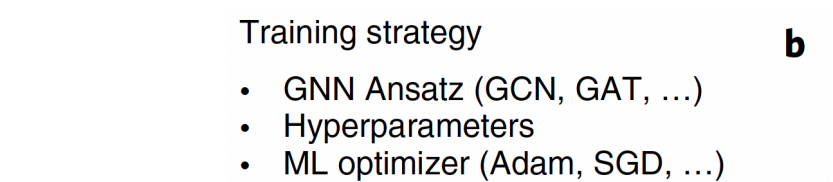


Training strategy

All of the following aspects of the training strategy can be optimized in an outer loop (e.g. grid search, Bayesian optimization methods etc.)

1. choices of GNN layers (e.g. architecture, number)
2. hyperparameters (e.g. embedding dimensions, penalty)
3. optimizers (e.g. learning rate, Adam, SGD)

Here the authors use a simple 2 layer GCN with $d_0 = \lfloor \sqrt{n} \rceil$ if $n \geq 10^5$ else $d_0 = \lfloor \sqrt[3]{n} \rceil$, $d_1 = \lfloor d_0/2 \rceil$ and $d_2 = 1$ (for QUBO).



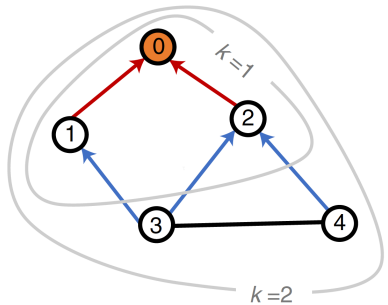
Training process

At training step, the initial embeddings are randomly initialized, and to make the model trainable, we convert the optimization problem Hamiltonian into a differentiable one by promoting the discrete variables to continuous probability with following relaxation

$$x_i \rightarrow p_i(\theta) \in [0, 1]$$
$$H_{QUBO} \rightarrow \mathcal{L}_{QUBO}(\theta) = \sum_{i,j} p_i(\theta)\, Q_{ij} p_j(\theta)$$

The model is trained for up to $\sim 10^5$ epochs with early stopping absolute tolerance of 10^{-4} and a patience of 10^3.

GNN training

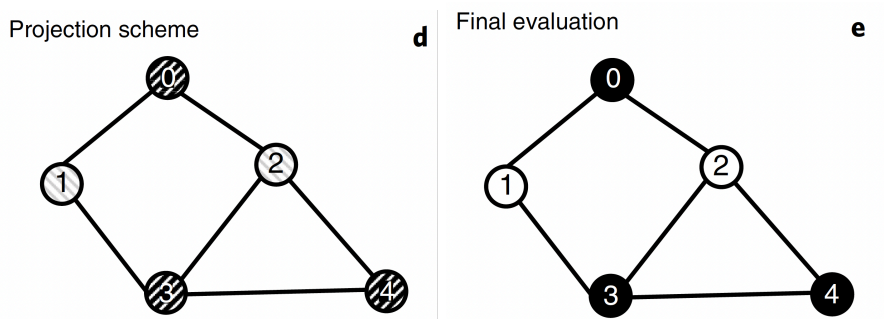


Loss function

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

Evaluation

As for evaluation, apply projection heuristics to map p_i back to x_i, e.g. simply $x_i = \lfloor p_i \rceil$. Note that the initial embeddings $\mathbf{h}_\nu^k$ are randomly initialized as they do not carry any inherent features. And this may result in random projections in the evaluation. The authors propose to take multiple shots of evaluation and look for the best result. This can also be taken from training steps with almost no computational cost.



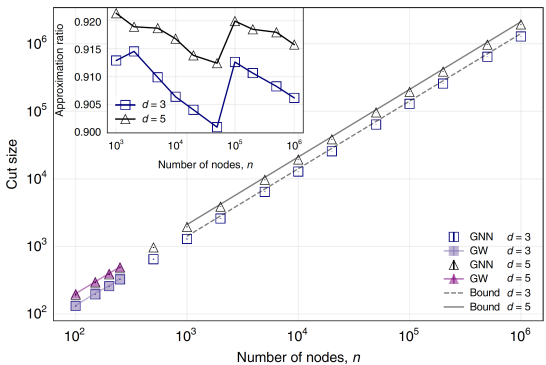
Result on MaxCut: performance

Numerical benchmarks on MaxCut problem are tested on 3-regular and 5-regular graphs with $n = 100$ nodes. The upper bound on d-regular graph is estimated from the theoretical optimum[0] as

$$cut^\star = \left(\frac{d}{4} + P_\star\sqrt{\frac{d}{4}} + \mathcal{O}\left(\sqrt{d}\right)\right) n + \mathcal{O}(n)$$

$$\Rightarrow cut_{ub} = \left(\frac{d}{4} + P_\star\sqrt{\frac{d}{4}}\right) n$$

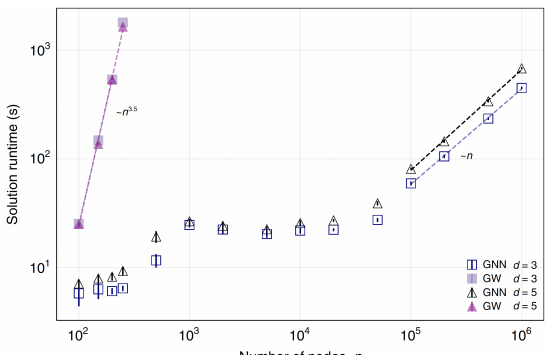
The performance is also evaluated through approximation ratio $\alpha = \frac{cut_{approx}}{cut_{ub}}$



[0] $P_\star \approx .7632$ is a constant related to the graound-state energy of Sherrington-Kirkpatrick model

Result on MaxCut: time cost

For comparison, Goemans-Williamson (GW) algorithm using semi-definite programming and randomized rounding is the best known P-time estimate. The time complexity of GW is $\tilde{O}\left(n^{3.5}\right)$ capped at ~ 250 nodes, while the proposed GNN-based approach scales linearly with $\tilde{O}(n)$ at the scale of $10^5 \sim 10^6$ nodes.



Result on MaxCut: complete benchmark

Further benchmark on MaxCut problem is conducted on Gset dataset. The competing algorithms are (1) Breakout local search (BLS, best known on Gset), (2) SDP solver using dual scaling (DSDP), (3) Tabu Search metaheuristic (KHLWG), (4) Recurrent GNN for maximum constraint satisfaction problems (RUN-CSP) and (5) Physics-inspired GNN (PI-GNN, this work). Relative error is given by $\epsilon = (cut_{best} - cut_{PI-GNN}) \,/\, |\mathcal{E}|$.

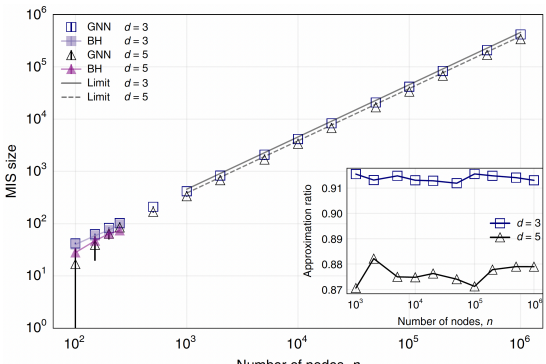
Table 1 | Numerical results for MaxCut on Gset instances

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

Result on MIS: performance

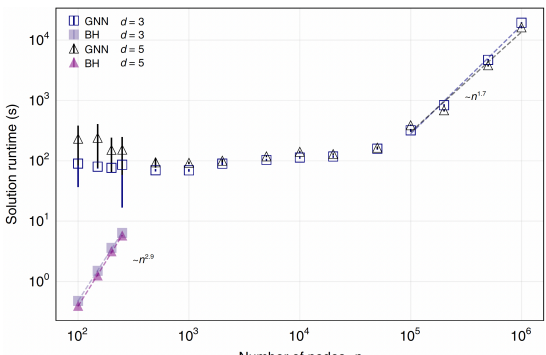
Recall that the Hamiltonian of MIS includes a penalty term as a soft constraint, so the decisions made by the model are not necessarily feasible decisions. While this only happens in very few cases when setting $P = 2$, the post-processing is still needed to greedily remove violations.

The best known bounds on the ratie $\frac{\alpha_d}{n}$ are derived using refined versions of Markov's inequality where $\frac{\alpha_3}{n} \approx 0.45537$, $\frac{\alpha_5}{n} \approx 0.38443$, and the approximation ratio is $\alpha = \frac{\alpha_{approx}}{\alpha_d}$



Result on MIS: time cost

For comparison, Boppana-Halldórsson (BH) algorithm has the best known performance guarantee. The time complexity of BH is $\tilde{O}\left(n^{2.9}\right)$ capped at ~ 500 nodes, while the proposed GNN-based approach scales better than linearly with $\tilde{O}\left(n^{0.8}\right)$ (training alone) and $\tilde{O}\left(n^{1.7}\right)$ (post-processing included) at the scale of $10^5 \sim 10^6$ nodes.



Outline

Real world applications

Other than numerical experiments, the proposed PI-GNN approach can also be applied to real world scenarios with ease.

Risk diversification

weighted version of MIS

$$H_{RD} = -\sum_{i\in\mathcal{V}} \mu_i x_i + P\sum_{(i,j)\in\mathcal{E}} \Sigma_{ij} x_i x_j$$

where μ is the expected return and Σ is the covariance matrix capturing volatility.

Interval scheduling

equivalent to MIS

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

where edges are constructed connecting nodes whose requested resource overlap.

Sensor placement in water distribution networks

weighted version of MVC

$$H_{SP} = \sum_{i\in\mathcal{V}} c_i x_i + P\sum_{(i,j)\in\mathcal{E}} (1-x_i)(1-x_j)$$

where c is cost of sensor placement at certain location.

Recap: Combinatorial optimization with physics-inspired graph neural networks

1. Combinatorial optimization: discrete NP-hard problems to solve
2. Physics-inspired: Ising model and quantum annealing
3. Graph neural networks: native graph-based neural network solver
4. Unsupervised learning: continuous relaxation to the discrete Hamiltonian
5. Highlights:
   a) general-purpose: shared form of solver for many problems
   b) efficient: fast approximation in nearly linear time
   c) scalable: can be used for large-scale problems
   d) generalizable: can be generalized to PUBOs and real world tasks
   e) robust: do not require state-of-art tweaks such as transfer learning, complex layer design (e.g. GraphSAGE, GIN or even GAT), hyper-parameter tuning, distributed training, randomized projection scheme, etc.

References


Wikipedia contributors. (2022) "Combinatorial optimization", *Wikipedia, The Free Encyclopedia.*


Sanchez-Lengeling, et al. (2021) "A Gentle Introduction to Graph Neural Networks", *Distill.*


Daigavane, et al. (2021) "Understanding Convolutions on Graphs", *Distill.*


Coles Patrick, et al. (2022) "Quantum Algorithm Implementations for Beginners", *ACM Transactions on Quantum Computing*


Boettcher Stefan. (2019) "Analysis of the Relation between Quadratic Unconstrained Binary Optimization (QUBO) and the Spin Glass Ground-State Problem", *Physical Review Research*