


# Distributed constrained combinatorial optimization leveraging hypergraph neural networks

Received: 15 November 2023

Accepted: 9 April 2024

Published online: 30 May 2024

 Check for updates

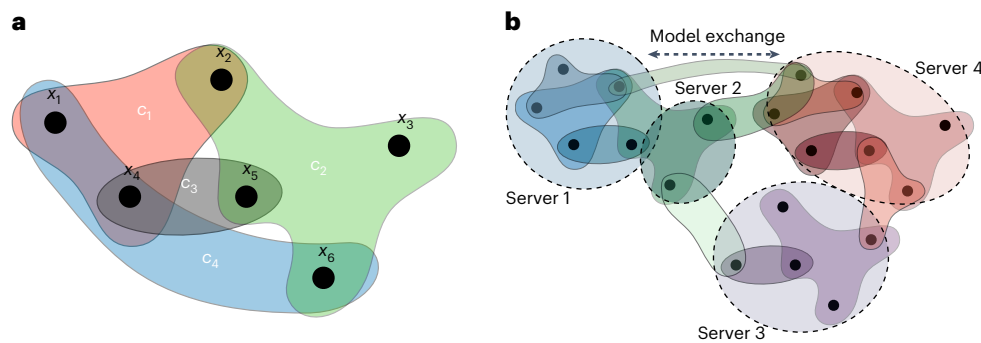
A list of authors and their affiliations appears at the end of the paper

Scalable addressing of high-dimensional constrained combinatorial optimization problems is a challenge that arises in several science and engineering disciplines. Recent work introduced novel applications of graph neural networks for solving quadratic-cost combinatorial optimization problems. However, effective utilization of models such as graph neural networks to address general problems with higher-order constraints is an unresolved challenge. This paper presents a framework, HypOp, that advances the state of the art for solving combinatorial optimization problems in several aspects: (1) it generalizes the prior results to higher-order constrained problems with arbitrary cost functions by leveraging hypergraph neural networks; (2) it enables scalability to larger problems by introducing a new distributed and parallel training architecture; (3) it demonstrates generalizability across different problem formulations by transferring knowledge within the same hypergraph; (4) it substantially boosts the solution accuracy compared with the prior art by suggesting a fine-tuning step using simulated annealing; and (5) it shows remarkable progress on numerous benchmark examples, including hypergraph MaxCut, satisfiability and resource allocation problems, with notable run-time improvements using a combination of fine-tuning and distributed training techniques. We showcase the application of HypOp in scientific discovery by solving a hypergraph MaxCut problem on a National Drug Code drug-substance hypergraph. Through extensive experimentation on various optimization problems, HypOp demonstrates superiority over existing unsupervised-learning-based solvers and generic optimization methods.

Combinatorial optimization is ubiquitous across science and industry. Scientists often need to make decisions about how to allocate resources, design experiments, schedule tasks or select the most efficient pathways among numerous choices. Combinatorial optimization techniques can help in these situations by finding the optimal or near-optimal solutions, thus aiding in the decision-making process. Furthermore, the integration of artificial intelligence (AI) into the field of scientific discovery is growing increasingly fluid, providing means

to enhance and accelerate research<sup>1</sup>. An approach to integrate AI into scientific discovery involves leveraging machine learning (ML) methods to expedite and improve the combinatorial optimization techniques to solve extremely challenging optimization tasks. Several combinatorial optimization problems are proved to be NP-hard, rendering most existing solvers non-scalable. Moreover, the continually expanding size of today's datasets makes existing optimization methods inadequate for addressing constrained optimization problems on such vast

✉ e-mail: [nheydaribeni@ucsd.edu](mailto:nheydaribeni@ucsd.edu)



**Fig. 1 | HyperOp methods. a, b,** Hypergraph modelling (a) and distributed training of HyperGNN (b) in HyperOp.

scales. To facilitate the development of scalable and rapid optimization algorithms, various learning-based approaches have been proposed in the literature<sup>2,3</sup>.

Learning-based optimization methods can be classified into three main categories: supervised learning, unsupervised learning and reinforcement learning. Supervised learning methods<sup>4,5,6,7,8</sup> train a model to address the given problems using a dataset of solved problem instances. However, these approaches exhibit limitations in terms of generalizability to problem types not present in the training dataset and tend to perform poorly on larger problem sizes. Unsupervised-learning approaches<sup>2,9,10</sup> do not rely on datasets of solved instances. Instead, they train ML models using optimization objectives as their loss functions. Unsupervised methods offer several advantages over supervised ones, including enhanced generalizability and eliminating the need for datasets containing solved problem instances, which can be challenging to acquire. Reinforcement-learning-based methods<sup>11,12,13,14</sup> hold promise for specific optimization problems, provided that lengthy training and fine-tuning times can be accommodated.

Unsupervised-learning-based optimization methods can be conceptualized as a fusion of gradient-descent-based optimization techniques with learnable transformation functions. In particular, in an unsupervised-learning-based optimization algorithm, the optimization variables are computed through an ML model, and the objective function is optimized by applying gradient descent over the parameters of this model. In other words, instead of conducting gradient descent directly on the optimization variables, it is applied to the parameters of the ML model responsible for generating them. The goal is to establish more efficient optimization paths compared to conventional direct gradient descent. This approach can potentially facilitate better escapes from subpar local optima. Various transformation functions are utilized in the optimization literature to render the optimization problems tractable. For example, there are numerous techniques to linearize or convexify an optimization problem by employing transformation functions that are often lossy<sup>15</sup>. We believe that adopting ML-based transformation functions for optimization offers a substantial capability to train customized transformation functions that best suit the specific optimization task.

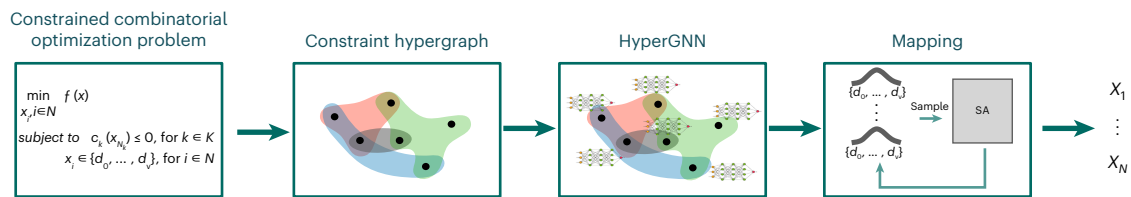
For combinatorial optimization problems over graphs, graph neural networks are frequently employed as the learnable transformation functions<sup>2,9,10</sup>. However, when dealing with complex systems with higher-order interactions and constraints, graphs fall short of modelling such relations. When intricate relationships among multiple entities extend beyond basic pairwise connections, hypergraphs emerge as invaluable tools for representing a diverse array of scientific phenomena. They have been utilized in various areas including biology and bioinformatics<sup>16,17</sup>, social network analysis<sup>18</sup>, chemical and material science<sup>19</sup> and image and data processing<sup>20</sup>. Hypergraph neural networks (HyperGNN)<sup>21</sup> have also been commonly used for certain learning tasks such as image and visual object classification<sup>21</sup> and material removal

rate prediction<sup>19</sup>. It is anticipated that HyperGNNs may serve as valuable tools for addressing combinatorial optimization problems with higher-order constraints.

Although unsupervised-learning approaches for solving combinatorial optimization problems offer numerous advantages, they may face challenges, including susceptibility to getting trapped in suboptimal local solutions and scalability issues. In a recent work<sup>22</sup>, the authors argue that for certain well-known combinatorial optimization problems, unsupervised-learning optimization methods may exhibit inferior performance compared to straightforward heuristics. However, it is crucial to recognize that these unsupervised methods possess a notable advantage in their generic applicability. Generic solvers like gradient-based techniques (such as stochastic gradient descent and Adam), as well as simulated annealing (SA), may not be able to compete with customized heuristics that are meticulously crafted for specific problems. Nonetheless, they serve as invaluable tools for addressing less-known problems lacking effective heuristics. Consequently, the strategic utilization of unsupervised-learning-based optimization methods can enhance and extend the capabilities of existing generic solvers, leading to the development of efficient tools for addressing a diverse range of optimization problems.

In this study, we build upon the unsupervised-learning-based optimization method for quadratic-cost combinatorial optimization problems on graphs introduced in ref. 2 and present HyperOp, a new scalable solver for a wide range of constrained combinatorial optimization problems with arbitrary cost functions. Our approach is applicable to problems with higher-order constraints by adopting hypergraph modelling for such problems (Fig. 1(a)); HyperOp subsequently utilizes HyperGNNs in the training process, a generalization of the graph neural networks employed in ref. 2. HyperOp further combines unsupervised HyperGNNs with another generic optimization method, SA<sup>23</sup>, to boost its performance. Incorporating SA with HyperGNNs can help with mitigation of the potential subpar local optima that may arise from HyperGNN training.

To establish a scalable solver and expedite the optimization process, HyperOp proposes two algorithms for parallel and distributed training of HyperGNNs. First, it develops a distributed training algorithm in which the hypergraph is distributed across a number of servers and each server has only a local view of the hypergraph. We develop a collaborative distributed algorithm to train the HyperGNN and solve the optimization problem (Fig. 1(b)). Second, HyperOp proposes a parallel HyperGNN training approach where the costs associated with constraints are computed in a scalable manner. We further exhibit the transferability of our models, highlighting their efficacy in solving different optimization problems on the same hypergraph through transfer learning. This not only shows the generalizability of HyperOp but also considerably accelerates the optimization process. HyperOp is tested by comprehensive experiments, thoughtfully



**Fig. 2 | HypOp overview.** Overview of the steps performed in HypOp for solving problem (1).

designed to provide insights into unsupervised-learning-based optimization algorithms and their effectiveness across diverse problem types. We validate the scalability of HypOp by testing it on huge graphs, showcasing how parallel and distributed training can yield high-quality solutions even on such massive graphs. In summary, our contributions are as follows:

- (1) Presenting HypOp, a scalable unsupervised-learning-based optimization method for addressing a wide spectrum of constrained combinatorial optimization problems with arbitrary cost functions. Notably, we use HyperGNNs within the realm of learning-based optimization for general combinatorial optimization problems with higher-order constraints.
- (2) Enabling scalability to much larger problems by introducing a distributed and parallel architecture for HyperGNN training.
- (3) Demonstrating generalizability to other problem formulations by knowledge transfer from the learned experience of addressing one set of cost/constraints to another set for the same hypergraph.
- (4) Substantially boosting solution accuracy and improving run time using a combination of fine-tuning (SA) and distributed training techniques.
- (5) Demonstrating the superiority of HypOp over existing unsupervised-learning-based solvers and generic optimization methods through extensive experimentation on a variety of combinatorial optimization problems. We address a set of scientific problems including a hypergraph MaxCut problem, satisfiability problems (3SAT) and resource allocation.
- (6) Showcasing the application of HypOp in scientific discovery by solving a hypergraph MaxCut problem on the National Drug Code (NDC) drug-substance hypergraph.

## Article structure

The remainder of this Article is structured as follows. We provide the problem statement in the next section, followed by the HypOp method. We describe two algorithms for distributed and scalable training of HypOp in ‘Distributed and parallel HypOp training’. Our experimental results are provided in the ‘Experiments’ section. We showcase the possibility of transfer learning in HypOp in ‘Transfer learning with HypOp for different types of problems’. Then we examine the applications of HypOp in scientific discovery and conclude with a discussion. Related work, preliminaries, extra details on the experimental setup and results, and limitations are provided in Supplementary Information.

## Problem statement

We consider a constrained combinatorial optimization problem over  $x_i$ ,  $i \in \mathcal{N}$ , where  $\mathcal{N} = \{1, \dots, N\}$  is the set of optimization variable indices. The variables  $x_i$  are integers belonging to the set  $\{d_0, \dots, d_v\}$ , where  $d_0, \dots, d_v$  are some given integers and  $d_i < d_j$  for  $i < j$ . There are  $K$  constraint functions,  $c_k(x_{\mathcal{N}_k})$ ,  $k \in \mathcal{K}$ , where  $\mathcal{K} = \{1, \dots, K\}$  is the set of constraint indices,  $x_{\mathcal{N}_k} = \{x_i, i \in \mathcal{N}_k\}$ , and we have  $\mathcal{N}_k \subset \mathcal{N}$ . We consider the following optimization problem with an arbitrary cost function  $f(x)$ :

$$\min_{x_i, i \in \mathcal{N}} f(x) \quad (1a)$$

$$\text{subject to } c_k(x_{\mathcal{N}_k}) \leq 0, \text{ for } k \in \mathcal{K} \quad (1b)$$

$$x_i \in \{d_0, \dots, d_v\}, \text{ for } i \in \mathcal{N} \quad (1c)$$

Solving the above optimization problem can be challenging, potentially falling into the category of NP-hard problems. This complexity arises from the presence of discrete optimization variables, as well as the potentially non-convex and non-linear objective and constraint functions. Consequently, efficiently solving this problem is a complex task. Additionally, the discreteness of optimization variables implies that the objective function is not continuous, making gradient-descent-based approaches impractical. To address this, a common strategy involves relaxing the state space into continuous spaces, solving the continuous problem and then mapping the optimized continuous values back to discrete ones. Although this approach may not guarantee globally optimal solutions due to the relaxation and potential convergence to suboptimal local optima, it remains a prevalent practice due to the availability and efficiency of gradient-descent-based solvers. Such solvers, such as Adam<sup>24</sup>, are widely developed and used as the main optimization method for ML tasks. The relaxed version of optimization problem (1) is given below:

$$\min_{p_i, i \in \mathcal{N}} f(p) \quad (2a)$$

$$\text{subject to } c_k(p_{\mathcal{N}_k}) \leq 0, \text{ for } k \in \mathcal{K} \quad (2b)$$

$$p_i \in [d_0, d_v], \text{ for } i \in \mathcal{N}, \quad (2c)$$

where  $p_i$  represents the continuous (relaxed) form of  $x_i$  and is within the continuous interval  $[d_0, d_v]$ .

## HypOp method

In this section, we describe our method, called HypOp, for solving problem (1). An overview of our algorithm is provided in Fig. 2, with each component explained subsequently. The main steps of the algorithm are (1) hypergraph modelling of the problem, (2) solving the relaxed version of the problem (problem (2)) using HyperGNNs in an unsupervised-learning manner and (3) mapping the continuous outcomes to discrete values using a postprocessing mapping algorithm. In the following, we describe each step of HypOp in more detail.

### Hypergraph modelling

To leverage HyperGNNs to solve problem (1) (or (2)), we model the constraints of this problem through a hypergraph as follows. Consider a hypergraph with nodes  $x_i$ ,  $i \in \mathcal{N}$ . For each constraint  $c_k(x_{\mathcal{N}_k})$ , we form a hyperedge comprising nodes  $x_{\mathcal{N}_k}$ . This hypergraph is referred to as the constraint hypergraph of the problem. For instance, in a problem with six variables and four constraints, the constraint hypergraph is illustrated in Fig. 1a. It is worth noting that for problems with an inherent graph or hypergraph structure, such as graph MaxCut, the constraint hypergraph coincides with the underlying graph or hypergraph of the problem.

## HyperGNN as a learnable transformation function

One can solve optimization problem (2) directly using gradient-descent-based methods and then map the continuous outcomes to the discrete values. Penalty-based methods are a common practice in these approaches to ensure that the constraints are satisfied in the final outcome. However, as will be demonstrated in this Article, one can achieve better solutions by solving an alternative optimization problem. In this alternative problem, the optimization variables  $p_i$  are the outcomes of a transformation function based on HyperGNNs. Such transformation functions are utilized with the goal of capturing the intricate patterns and interdependence of the variables through the problem constraints and therefore obtaining better solutions.

In optimization literature, various transformation functions are utilized to render optimization problems tractable. For instance, there are numerous techniques for linearizing optimization problems through the use of transformation functions<sup>15</sup>. However, in much of the existing literature, the transformation function is a fixed function. In HypOp, we introduce a parameterized transformation function based on HyperGNNs. In particular, we consider  $p = \mathcal{G}(\sigma; W)$  (or  $p_i = \mathcal{G}_i(\sigma; W)$  for  $i \in \mathcal{N}$ ), where  $p_i$  is the (continuous) assignment to node  $i$ ,  $W$  is the parameter vector of the HyperGNN-based transformation function,  $\mathcal{G}$ , and  $\sigma = [\sigma_1, \dots, \sigma_N]$  is the input embedding ( $\sigma_i$  is node  $i$ 's input embedding). By adopting a parameterized transformation function, we optimize our objective by simultaneously learning a good transformation function ( $W$ ) and optimizing the input embedding ( $\sigma$ ). This alternative optimization problem is described below:

$$\min_{\sigma, W} f(\mathcal{G}(\sigma; W)) \quad (3a)$$

$$\text{subject to } c_k(\mathcal{G}_{\mathcal{N}_k}(\sigma; W)) \leq 0, \text{ for } k \in \mathcal{K} \quad (3b)$$

$$\mathcal{G}_i(\sigma; W) \in [d_0, d_v], \text{ for } i \in \mathcal{N} \quad (3c)$$

We define the HyperGNN  $\mathcal{G}(\sigma; W)$  on the constraint hypergraph with the following augmented (penalized) loss function:

$$\hat{f}(p) = f(p) + \sum_{k \in \mathcal{K}} \lambda_k (c_k(p_{\mathcal{N}_k}))^+ \quad (4)$$

where  $(a)^+ = \max(0, a)$  and  $\lambda_k$  is the weight of constraint  $k$ . The HyperGNN model used in HypOp is based on hypergraph convolutional networks introduced in ref. 21. In particular, we have the following layer-wise operation in our HyperGNN model:

$$H^{(l+1)} = \sigma \left( D_v^{-\frac{1}{2}} \tilde{M} D_v^{-\frac{1}{2}} H^{(l)} W^{(l)} \right) \quad (5)$$

where we define

$$\tilde{M} = A \tilde{D}_e^{-1} A^T - \text{diag} \left( A \tilde{D}_e^{-1} A^T \right) \quad (6)$$

and  $H^{(l)} \in \mathbb{R}^{N \times f_l}$  is the matrix of node features at the  $l$ th layer,  $f_l$  is the size of the  $l$ th layer node features,  $A$  is the hypergraph incidence matrix defined as  $(A)_{ij} = 1$  if node  $i$  belongs to hyperedge  $j$ , and  $(A)_{ij} = 0$  otherwise.  $D_v$  and  $D_e$  are the diagonal degree matrices of nodes and hyperedges, respectively, and  $\tilde{D}_e = D_e - I$ .  $W^{(l)} \in \mathbb{R}^{f_l \times f_{l+1}}$  is the  $l$ th layer trainable weight matrix.

## Mapping with SA

To transform the continuous outputs of the HyperGNN to integer node assignments, we use the continuous outputs generated through the HyperGNN to derive a probability distribution over the discrete values

$\{d_0, \dots, d_v\}$ . This probability distribution is constructed such that the probability of a variable being equal to a given value is inversely proportional to the distance of the variable's continuous output with that value. For example, for the binary case of 0 and 1, the output will be the probability of 1. The mapping then proceeds as follows. We take a sample,  $x_1, \dots, x_N$ , from the output distribution corresponding to  $p_1, \dots, p_N$ . We then initialize a SA algorithm with that sample and minimize  $f(x)$ . This process is repeated a given number of times. The best outcome is chosen as the final solution.

## Distributed and parallel HypOp training

In this section, we introduce two scalable multi-GPU HyperGNN training algorithms for HypOp: parallel multi-GPU training that shuffles constraints among multiple servers (graphics processing units (GPUs)) and trains the HyperGNN in a parallel way; and distributed training, where each server (GPU) holds part of the hypergraph (local view of the hypergraph) and trains the HyperGNN in a distributed manner in collaboration with other servers. For a comparison between our approach and the existing state of the art, see Supplementary Information Section I.

### Distributed training

In distributed training, we assume that the hypergraph is either inherently distributed across a number of servers, and these servers collaborate to solve the optimization problem, or that the hypergraph is partitioned between the GPUs (servers) beforehand to expedite the optimization process and handle large-scale problems effectively. In this setup, each GPU maintains a local view of the hypergraph (has access to a subgraph of the original hypergraph). The training of HypOp takes place collaboratively among these GPUs. Figure 1b shows a schematic example of distributed HyperGNN training in HypOp.

The outline of the distributed multi-GPU training is summarized in Algorithm 1. Given a hypergraph  $G = (V, E)$  and  $S$  GPUs to train the HyperGNN model  $M$ , we first partition the hypergraph  $G$  into  $S$  subgraphs. The subgraph  $G^s = (V^s, E^{s, \text{inner}}, E^{s, \text{outer}})$  consists of three components:  $V^s$  represents the nodes in  $G^s$ ,  $E^{s, \text{inner}}$  includes inner hyperedges connecting nodes within  $G^s$ , and  $E^{s, \text{outer}}$  consists of outer hyperedges linking nodes in  $G^s$  to other subgraphs. During the forward pass, each GPU computes local node embeddings using  $E^{s, \text{inner}}$ , and during the backward pass, it calculates the loss with both local node embeddings and node embeddings connected by  $E^{s, \text{outer}}$  to update model weights  $M^s$ . Subsequently, the gradients of each  $M^s$  are aggregated and disseminated to all GPUs.

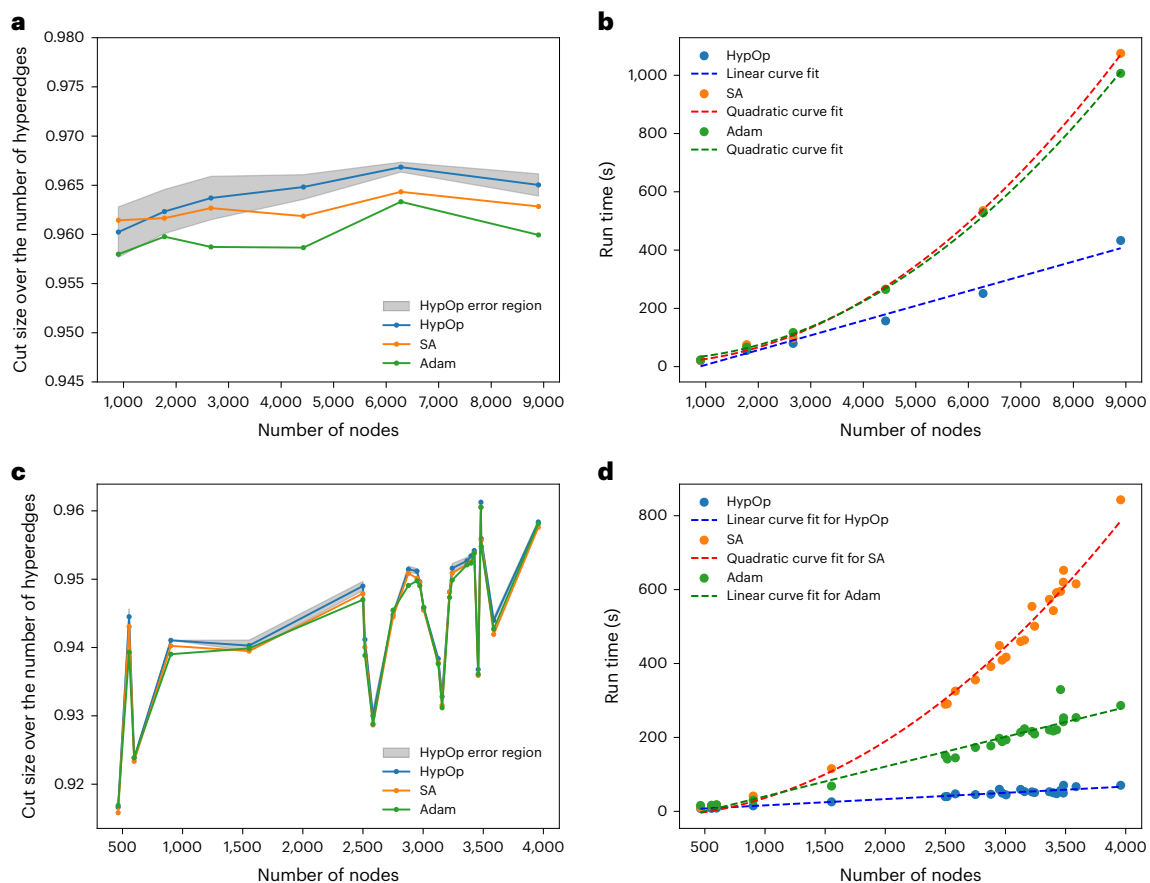
### Algorithm 1: Distributed multi-GPU HyperGNN training.

**Require:** Hypergraph  $G = (V, E)$ , HyperGNN model  $M$ , number of GPUs  $S$ , epoch number  $EP$

**Ensure:** Trained HyperGNN model  $M$

- 1: Partition  $G$  into  $S$  subgraphs:  $G^1, G^2, \dots, G^S$
- 2: Initialize  $M$  on each GPU
- 3: Distribute  $G^s$  to GPU<sup>s</sup>, for  $s = 1$  to  $S$
- 4: **for**  $ep$  in  $EP$  **do**
- 5:   **for**  $s \leftarrow 1$  to  $S$  **do**
- 6:     Extract inner hyperedges  $E^{s, \text{inner}}$  and outer hyperedges  $E^{s, \text{outer}}$  from  $G^s$
- 7:     **Forward pass:** Compute embeddings using  $E^{s, \text{inner}}$  on GPU<sup>s</sup>
- 8:     **Backward pass:** Calculate loss using embeddings,  $E^{s, \text{inner}}$  and  $E^{s, \text{outer}}$
- 9:     Compute gradients based on loss
- 10:   **end for**
- 11:   Aggregate gradients from all GPUs
- 12:   Update  $M$  based on aggregated gradients
- 13:   Broadcast updated  $M$  to all GPUs
- 14: **end for**
- 15: **return**  $M$





**Fig. 3 | HypOp versus SA and Adam.** **a, b**, Performance (**a**) and run time (**b**) of HypOp compared with SA and Adam for hypergraph MaxCut problem on synthetic random hypergraphs. **c, d**, Performance (**c**) and run time (**d**) of HypOp compared with SA and Adam for hypergraph MaxCut problem on real American Physical Society (APS) hypergraphs. For almost the same performance (**a** and **c**), the run time of HypOp is remarkably less compared to SA and Adam (**b** and **d**)

**d**), with SA's run time growing quadratically fast, as opposed to the run time of HypOp, which grows linearly. HypOp performance is presented as the average of results from ten sets of experiments, and the error region shows the standard deviation of the results. The run-time standard deviations were insignificant, with an average of 3.7 s.

### Parallel multi-GPU training

The HyperGNN-based optimization tool used in HypOp allows us to take advantage of GNN and HyperGNN parallel training techniques to accelerate the optimization and enable optimization of large-scale problems. The outline of the parallel multi-GPU training is summarized in Supplementary Information Algorithm 2. Given an input hypergraph  $G = (V, E)$  and  $S$  GPUs, the HyperGNN model  $M$  is initialized on each GPU. In each epoch, hypergraph  $G$ 's hyperedge information  $E$  are randomly partitioned into  $S$  parts. Then each GPU<sup>*s*</sup> receives its corresponding hyperedges  $E^s$  and trains  $M^s$  on them. The gradients are then aggregated and disseminated to all GPUs.

### Experiments

We have tested HypOp on multiple types of combinatorial optimization problems, including hypergraph and graph MaxCut, graph maximum independent set (MIS), satisfiability and resource allocation problems. We note that we have tested our method on the known and highly studied problems of satisfiability, graph MaxCut and MIS to show that even though our solver might not be able to compete with heuristics that are specifically designed for those problems, it can still be used as a descent solver for such well-known problems. Some of our results are provided in Supplementary Information Section VII.

### Baselines

Because HypOp is proposed as a generic solver for combinatorial optimization problems, we compare HypOp with other generic solvers such

as SA<sup>23</sup> and gradient-descent-based methods (for example, Adam)<sup>24</sup>. Specifically, we use the following optimization methods as our baselines:

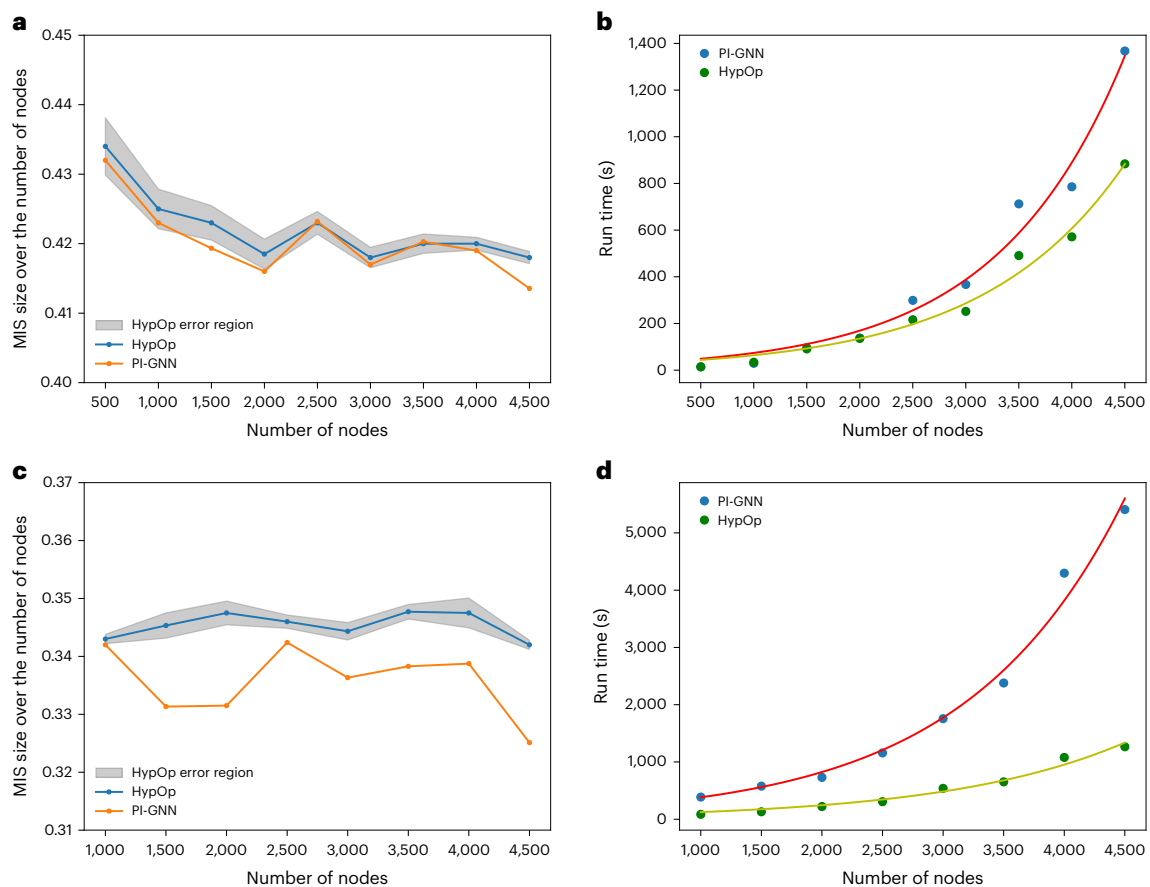
(1) SA: the experiments with the SA solver are conducted by setting the HyperGNN training epochs in HypOp to 0 and randomly initializing the SA step.

(2) Adam: the experiments with the Adam solver are conducted by removing the HyperGNN part from HypOp and directly optimizing with regard to the continuous variables  $p_i$  using Adam. The exact SA fine-tuning step in HypOp is then applied to the optimized results obtained using Adam to generate discrete variable assignments.

(3) Bipartite GNN: to justify the need for HyperGNNs to solve problems with higher-order constraints, we compare HypOp with a GNN-based solver similar to PI-GNN<sup>2</sup>. This GNN-based solver is developed by transforming the constraint hypergraph of the problem into a constraint bipartite graph, where each hyperedge is represented by a factor node connected to the nodes within that hyperedge. Then, HypOp solves the problem on the bipartite graph. Note that this is similar to PI-GNN being applied to a bipartite graph, with the addition of the fine-tuning step that is provided in HypOp.

(4) PI-GNN: for problems over graphs, such as graph MIS and MaxCut, we compare with PI-GNN<sup>2</sup>.

(5) Run-CSP: for the MaxCut problem over graphs, we also consider Run-CSP<sup>10</sup> as a baseline, which is another unsupervised-learning-based optimization method for problems over graphs. We also compare our graph MaxCut results with the best-known heuristic, breakout local search<sup>25</sup>.



**Fig. 4 | HypOp versus PI-GNN. a,b,** Performance (a) and run time (b) of HypOp and PI-GNN for the MIS problem on random regular graphs with  $d = 3$ . **c,d,** Performance (c) and run time (d) of HypOp and PI-GNN for the MIS problem on random regular graphs with  $d = 5$ . HypOp achieves better results than PI-GNN (a

and c) in less time (b and d). HypOp performance is presented as the average of results from ten sets of experiments, with the error region showing the standard deviation of the results. The run-time standard deviations were insignificant, with an average of 10 s.

## Hypergraph MaxCut

We used HypOp to solve the hypergraph MaxCut problem. In hypergraph MaxCut, the goal is to partition the nodes of a hypergraph into two groups such that the cut size is maximized, where the cut size is the number of hyperedges that have at least one pair of nodes belonging to different groups. We compare the performance of HypOp with SA and a direct-gradient-descent method (Adam) as the baselines. Also note that PI-GNN<sup>2</sup> cannot solve problems over hypergraphs, and thus we do not compare with it. We perform our experiments on both synthetic and real hypergraphs. The real hypergraphs are extracted from the American Physical Society<sup>26</sup>. See Supplementary Information Section V for more details on the real and synthetic hypergraphs used in our experiments. In Fig. 3, we show the performance of HypOp compared to SA and Adam. It can be seen that with almost the same performance, the run time of HypOp is remarkably less than the other two methods. Notably, the run time of HypOp grows linearly with the number of nodes, whereas the run time of SA grows quadratically.

In Extended Data Fig. 1, we compare HypOp with the bipartite GNN baseline. As depicted in the figure, for almost the same performance, HypOp has a considerably reduced run time compared to the bipartite GNN method (up to 8× improvement). The extended run time of the bipartite GNN can be attributed to the fact that the bipartite graph equivalent of a hypergraph with  $N$  nodes and  $K$  hyperedges would have  $N + K$  nodes. Given that the complexity of GNNs and HyperGNNs typically scale with the number of nodes, our experiments affirm that HyperGNNs are a more efficient alternative compared to bipartite graph neural networks.

## MIS problem

To compare HypOp with the baseline work PI-GNN<sup>2</sup> and highlight the benefit of the fine-tuning step using SA in HypOp, we solved MIS problems over graphs using HypOp. In Fig. 4, we show the performance and run time of PI-GNN and HypOp over regular random graphs with  $d = 3$  and  $d = 5$ . As can be seen in the figures, HypOp performs better than PI-GNN in terms of both optimality and run time. While generating better MIS sizes, the run time of HypOp is up to five times better than PI-GNN. HypOp also scales better than PI-GNN to larger graphs. Having different components in the optimization tool allows us to adjust their hyperparameters to better suit the specific problem at hand. Specially, we can have a more efficient algorithm by taking advantage of different types of solvers.

## Parallel and distributed training

In this section, we provide experimental results to verify the effectiveness of the scalable multi-GPU HyperGNN training described in Algorithm 1 and Supplementary Information Algorithm 2. Note that all of the prior results were generated by single-GPU training. For this section, we solved the MaxCut problem on one of the Stanford graphs, G22 (ref. 27), in addition to a larger graph, OGB-Arxiv (ref. 28) with 169,343 nodes and 1,166,243 edges. The results are summarized in Table 1. For the smaller Stanford graph (G22), the single-GPU training and Supplementary Information Algorithm 2 show similar performance, with improved run time using Supplementary Information Algorithm 2, and Algorithm 1 providing the outcome 3× faster than the single-GPU training. As the graph size becomes larger (OGB-Arxiv graph), single-GPU

**Table 1 | HypOp multi-GPU performance**

Graph	Nodes	Edges	Single GPU		Parallel multi-GPU		Distributed multi-GPU	
			Performance	Time	Performance	Time	Performance	Time
Stanford G22	2,000	19,990	13,185	15 s	13,160	12 s	13,128	5 s
OGB-Arxiv	169,343	1,166,243	OOM	–	855,505	750 s	854,746	411 s

Although parallel multi-GPU training can speed up optimization and resolve the OOM error, distributed training provides considerable speed-up with its result being within 0.09% of the parallel multi-GPU result.

**Table 2 | Hypergraph MaxCut with HypOp**

	HypOp	SA
Cut size	28,849 ± 91	28,472 ± 92
Run time (s)	1,679	3,616

Performance of HypOp for hypergraph MaxCut on the NDC dataset. Compared to SA, HypOp can achieve better results in less time.

training faces out of memory (OOM) errors, whereas multi-GPU training in HypOp can solve this problem. Algorithm 1 solves the problem almost 2× faster than Supplementary Information Algorithm 2, with slight performance degradation (~0.1% degradation).

## Transfer learning with HypOp for different types of problems

We explored the applicability of transfer learning within HypOp to address various problems on a single graph. Our approach involves pre-training the HyperGNN on a specific problem type and then utilizing the pretrained model for a different optimization problem on the same graph, focusing on optimizing the input embedding exclusively. In particular, we freeze the weight matrix  $W$  and only optimize the input embedding  $\sigma$ . The idea is that the HyperGNN learns an effective function to capture graph features (modelled by  $\mathcal{G}(\sigma; W)$ ) and therefore can potentially be used for other types of optimization problems on the same graph. The optimization on the embedding  $\sigma$  yields customized solutions tailored to the specific optimization problem. To validate this, we trained HypOp for the MaxCut problem and transferred it to address the MIS problem. In Extended Data Fig. 2, we show the performance of HypOp with and without transfer learning. The models are pretrained for the MaxCut problem and transferred to solve the MIS problem on the same graph. As we can see in the figure, the transferred model can achieve comparable results in almost no time compared to the vanilla training (training from scratch). Similar results can be seen in Extended Data Fig. 3, where we test the transferability of HypOp from the hypergraph MaxCut problem to the hypergraph partitioning (MinCut) problem. The hypergraph partitioning or MinCut problem aims at cutting the hypergraph into two groups of nodes with almost the same size and the smallest possible cut size. Notably, transfer learning has even provided better results in some instances. These results further underscore the promising potential of utilizing unsupervised-learning-based approaches to effectively and efficiently address complex optimization problems.

## Applications in scientific discovery

Combinatorial optimization is vital in diverse scientific and industrial applications, addressing challenges in resource allocation, experimental design and decision-making processes. In this section, we show an example of how HypOp can help with solving combinatorial optimization problems related to scientific discovery. We consider a hypothetical experimental design for national drug control, utilizing the NDC dataset<sup>29,30</sup> (see Supplementary Information Section VI for details). In this scenario, the goal is to select a set of substances for regulatory measures, dividing the dataset into two isolated groups of drugs. The objective is to determine the maximum number of substances requiring regulation to achieve this division. This problem can be formulated as

a hypergraph MaxCut problem, which we solved using HypOp and SA. As shown in Table 2, HypOp outperforms SA in terms of results and run time on the NDC hypergraph with 3,767 nodes and 29,810 hyperedges.

## Discussion

The comparison among HypOp, SA, Adam and PI-GNN in the ‘Experiments’ section imparts several crucial insights. First, relying solely on unsupervised-learning-based methods for end-to-end optimization has demonstrated limitations, as observed in this work and ref. 22. Combining such methods with generic optimization algorithms like SA substantially enhances performance, yielding acceptable solutions even when HyperGNNs or GNNs struggle to provide satisfactory results. Second, HypOp and other unsupervised-learning-based optimization methods can effectively serve as initializers for optimization variables in other solvers, such as SA. Our experiments demonstrate that if SA is not initialized with HyperGNN outputs, its run-time performance notably suffers. Third, our experiments confirm that HyperGNNs can function as effective learnable transformation functions, converting input embeddings into original optimization variables. This transformation acts as an acceleration for the solver, offering more efficient optimization paths. Given the widespread use of gradient descent across various domains, such as in ML model training, utilizing HyperGNNs as transformation functions has the potential to enhance gradient-descent-based optimization in numerous applications.

## Methods

In this study, we employed various methodologies including hypergraph modelling and HyperGNN training, distributed and parallel computation, SA and transfer learning to tackle general constrained combinatorial optimization problems. Initially, we utilized hypergraph modelling to represent the problem and applied a relaxation procedure to establish a smooth loss function. Subsequently, we defined a HyperGNN on the constraint hypergraph and trained it using an unsupervised-learning approach with the smooth loss function. Lastly, we employed a postprocessing mapping algorithm based on SA to convert the continuous outcomes of the HyperGNN into discrete values. Moreover, we improved scalability by integrating distributed and parallel HyperGNN training techniques and demonstrated the effectiveness of transfer learning methods.

## Reporting summary

Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

## Data availability

In this paper, we used publicly available datasets of the American Physical Society<sup>26</sup>, NDC<sup>29</sup>, Gset<sup>27</sup> and SATLIB<sup>31</sup>, together with synthetic hypergraphs and graphs. The procedure under which the synthetic hypergraphs and graphs are generated is explained throughout the paper. Some examples of the synthetic hypergraphs are provided with the code at ref. 32.

## Code availability

The code has been made publicly available at ref. 32. We used Python v.3.8 together with the following packages: torch v.2.1.1, tqdm v.4.66.1,

h5py v.3.10.0, matplotlib v.3.8.2, networkx v.3.2.1, numpy v.1.21.6, pandas v.2.0.3, scipy v.1.11.4 and sklearn v.0.0. We used PyCharm v.2023.1.2 and Visual Studio Code v.1.83.1 software.

## References

- Wang, H. et al. Scientific discovery in the age of artificial intelligence. *Nature* **620**, 47–60 (2023).
- Schuetz, M. J. A., Brubaker, J. K. & Katzgraber, H. G. Combinatorial optimization with physics-inspired graph neural networks. *Nat. Mach. Intell.* **4**, 367–377 (2022).
- Cappart, Q. et al. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.* **24**, 1–61 (2023).
- Khalil, E., Le Bodic, P., Song, L., Nemhauser, G. & Dilkina, B. Learning to branch in mixed integer programming. In *Proc. 30th AAAI Conference on Artificial Intelligence* 724–731 (AAAI, 2016).
- Bai, Y. et al. Simgnn: a neural network approach to fast graph similarity computation. In *Proc. 12th ACM International Conference on Web Search and Data Mining* 384–392 (ACM, 2019).
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L. & Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems 32* (eds Wallach, H. et al.) 15580–15592 (NeurIPS, 2019).
- Nair, V. et al. Solving mixed integer programs using neural networks. Preprint at <https://arXiv.org/2012.13349> (2020).
- Li, Z., Chen, Q. & Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *Proc. Advances in Neural Information Processing Systems 31* (eds Bengio, S. et al.) 537–546 (NeurIPS, 2018).
- Karalias, N. & Loukas, A. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Proc. Advances in Neural Information Processing Systems 33* (eds Larochelle, H. et al.) 6659–6672 (NeurIPS, 2020).
- Toenshoff, J., Ritzert, M., Wolf, H. & Grohe, M. Graph neural networks for maximum constraint satisfaction. *Front. Artif. Intell.* **3**, 580607 (2021).
- Mirhoseini, A. et al. A graph placement methodology for fast chip design. *Nature* **594**, 207–212 (2021).
- Yolcu, E. & Pócsos, B. Learning local search heuristics for boolean satisfiability. In *Proc. Advances in Neural Information Processing Systems 32* (eds Wallach, H. et al.) 7992–8003 (NeurIPS, 2019).
- 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/1911.04936> (2019).
- Kool, W., Van Hoof, H. & Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2018.
- Asghari, M., Fathollahi-Fard, A. M., Mirzapour Al-E-Hashem, S. M. J. & Dulebenets, M. A. Transformation and linearization techniques in optimization: a state-of-the-art survey. *Mathematics* **10**, 283 (2022).
- Feng, S. et al. Hypergraph models of biological networks to identify genes critical to pathogenic viral response. *BMC Bioinformatics* **22**, 1–21 (2021).
- Murgas, K. A., Saucan, E. & Sandhu, R. Hypergraph geometry reflects higher-order dynamics in protein interaction networks. *Sci. Rep.* **12**, 20879 (2022).
- Zhu, J., Zhu, J., Ghosh, S., Wu, W. & Yuan, J. Social influence maximization in hypergraph in social networks. *IEEE Trans. Netw. Sci. Eng.* **6**, 801–811 (2018).
- Xia, L., Zheng, P., Huang, X. & Liu, C. A novel hypergraph convolution network-based approach for predicting the material removal rate in chemical mechanical planarization. *J. Intell. Manuf.* **33**, 2295–2306 (2022).
- Wen, Y., Gao, Y., Liu, S., Cheng, Q. & Ji, R. Hyperspectral image classification with hypergraph modelling. In *Proc. 4th International Conference on Internet Multimedia Computing and Service* 34–37 (ACM, 2012).
- Feng, Y., You, H., Zhang, Z., Ji, R. & Gao, Y. Hypergraph neural networks. In *Proc. 33rd AAAI Conference on Artificial Intelligence* 3558–3565 (AAAI, 2019).
- Angelini, M. C. & Ricci-Tersenghi, F. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Mach. Intell.* **5**, 29–31 (2023).
- Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. Optimization by simulated annealing. *Science* **220**, 671–680 (1983).
- Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. Preprint at <https://arXiv.org/1412.6980> (2014).
- Benlic, U. & Hao, J.-K. Breakout local search for the max-cut problem. *Eng. Appl. Artif. Intell.* **26**, 1162–1173 (2013).
- APS dataset on Physical Review Journals, published by the American Physical Society, <https://journals.aps.org/datasets> (n.d.).
- Ye, Y. The gset dataset, <https://web.stanford.edu/~yye/yye/Gset> (Stanford, 2003).
- Hu, W. et al. Open graph benchmark: datasets for machine learning on graphs. In *Proc. Advances in Neural Information Processing Systems 33* (eds Larochelle, H. et al.) 22118–22133 (2020).
- Ndc-substances dataset. Cornell <https://www.cs.cornell.edu/~arb/data/NDc-substances/> (2018).
- Benson, A. R., Abebe, R., Schaub, M. T., Jadbabaie, A. & Kleinberg, J. Simplicial closure and higher-order link prediction. *Proc. Natl Acad. Sci. USA* **115**, E11221–E11230 (2018).
- Hoos, H. H., & Stützle, T. SATLIB: An online resource for research on SAT. Sat, 2000, 283–292 (2000).
- Heydaribeni, N., Zhan, X., Zhang, R., Eliassi-Rad, T. & Koushanfar, F. Source code for ‘Distributed constrained combinatorial optimization leveraging hypergraph neural networks’. *Code Ocean* <https://doi.org/10.24433/CO.4804643.v1> (2024).

## Acknowledgements

We acknowledge the support of the MURI programme of the Army Research Office under award no. W911NF-21-1-0322 and the National Science Foundation AI Institute for Learning-Enabled Optimization at Scale under award no. 2112665.

## Author contributions

All authors participated in developing the ideas implemented in the article, with N.H. taking the lead. The code was developed by X.Z., N.H. and R.Z. Experiment design and execution were carried out by N.H. and R.Z. The paper was initially drafted by N.H. and was later revised by F.K. F.K. and T.E.-R. supervised the work and reviewed the paper.

## Competing interests

N.H., R.Z., T.E.-R. and F.K. are listed as inventors on a patent application (serial number 63/641,601) on distributed constrained combinatorial optimization leveraging HyperGNNs. X.Z. declares no competing interests.

## Additional information

**Extended data** is available for this paper at <https://doi.org/10.1038/s42256-024-00833-7>.

**Supplementary information** The online version contains supplementary material available at <https://doi.org/10.1038/s42256-024-00833-7>.

**Correspondence and requests for materials** should be addressed to Nasimeh Heydaribeni.



**Peer review information** *Nature Machine Intelligence* thanks Petar Veličković and Haoyu Wang for their contribution to the peer review of this work.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

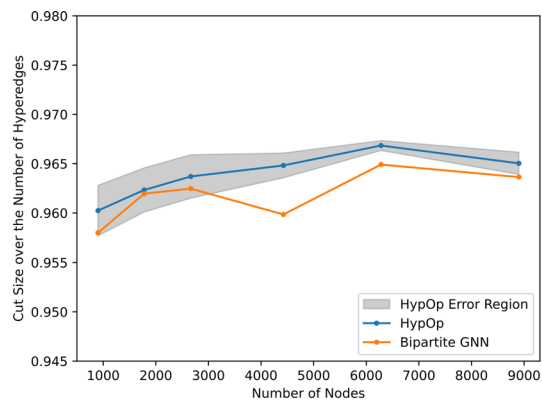
© The Author(s), under exclusive licence to Springer Nature Limited 2024

---

Nasimeh Heydaribeni <sup>1</sup>✉, Xinrui Zhan<sup>1</sup>, Ruisi Zhang <sup>1</sup>, Tina Eliassi-Rad<sup>2</sup> & Farinaz Koushanfar <sup>1</sup>

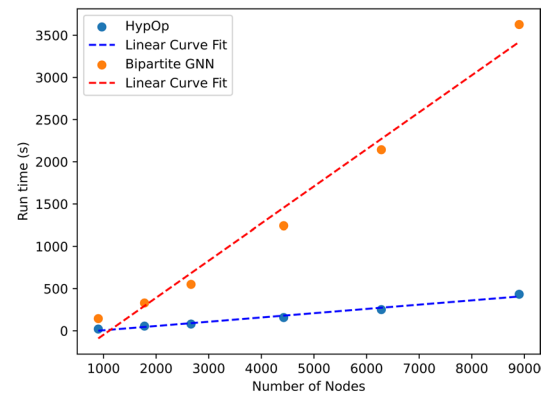
---

<sup>1</sup>Department of Electrical and Computer Engineering, University of California, San Diego, CA, USA. <sup>2</sup>Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA. ✉e-mail: [nheydaribeni@ucsd.edu](mailto:nheydaribeni@ucsd.edu)



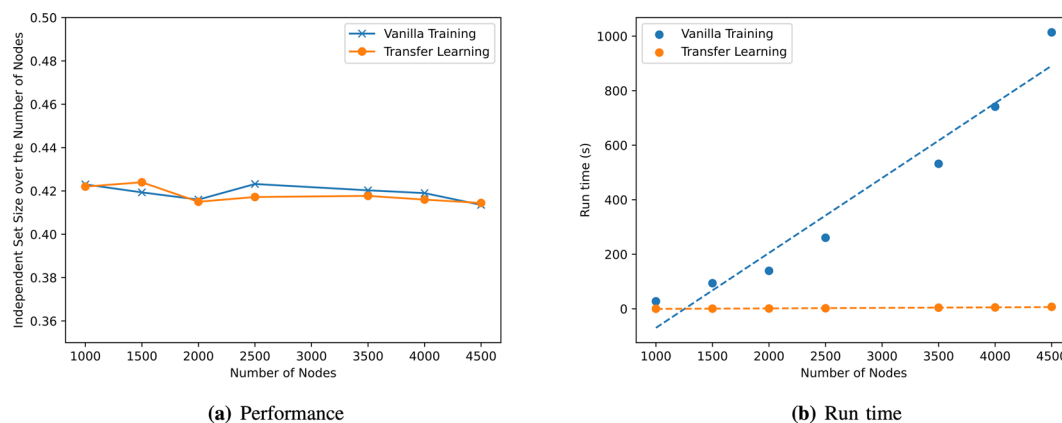
(a) Performance

**Extended Data Fig. 1 | HypOp vs. Bipartite GNN.** Comparison of HypOp with the bipartite GNN baseline for hypergraph MaxCut problem on synthetic random hypergraphs. For almost the same performance (a), HypOp has a remarkably

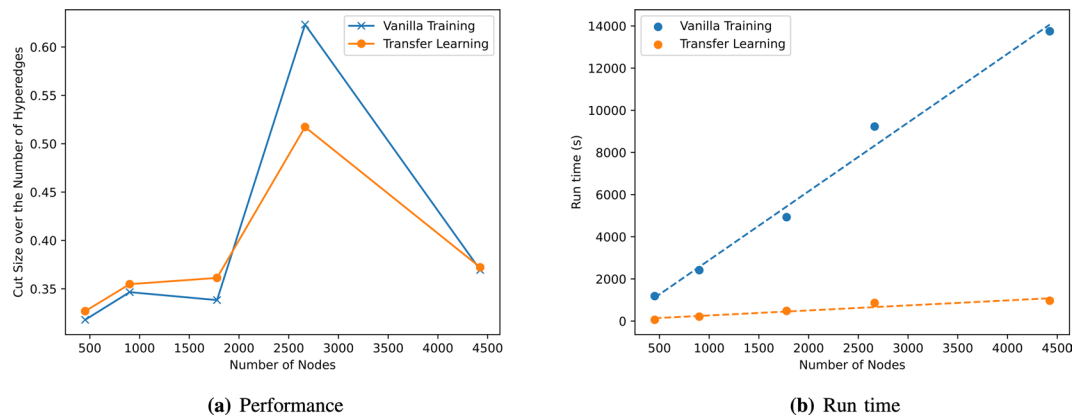


(b) Run time

less run time compared to the bipartite GNN baseline (b). HypOp performance is presented as the average of the results from 10 sets of experiments, with the error region showing the standard deviation of the results.



**Extended Data Fig. 2 | Transfer Learning.** Transfer Learning using HypOp from MaxCut to MIS problem on random regular graphs with  $d = 3$ . For almost the same performance (a), transfer learning provides the results in almost no amount of time compared to vanilla training (b).



**Extended Data Fig. 3 | Transfer Learning.** Transfer Learning using HypOp from Hypergraph MaxCut to Hypergraph MinCut on synthetic random hypergraphs. Compared to vanilla training, similar or better results are obtained using transfer learning (a) in a considerable less amount of time (b). Note that in the context of the Hypergraph MinCut problem, smaller cut sizes are favored.



Reporting Summary

Nature Portfolio wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Portfolio policies, see our [Editorial Policies](#) and the [Editorial Policy Checklist](#).

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed
<input type="checkbox"/>	<input checked="" type="checkbox"/> The exact sample size ( <i>n</i> ) for each experimental group/condition, given as a discrete number and unit of measurement
<input type="checkbox"/>	<input checked="" type="checkbox"/> A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
<input checked="" type="checkbox"/>	<input type="checkbox"/> The statistical test(s) used AND whether they are one- or two-sided <i>Only common tests should be described solely by name; describe more complex techniques in the Methods section.</i>
<input type="checkbox"/>	<input checked="" type="checkbox"/> A description of all covariates tested
<input checked="" type="checkbox"/>	<input type="checkbox"/> A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
<input type="checkbox"/>	<input checked="" type="checkbox"/> A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
<input checked="" type="checkbox"/>	<input type="checkbox"/> For null hypothesis testing, the test statistic (e.g. <i>F</i> , <i>t</i> , <i>r</i> ) with confidence intervals, effect sizes, degrees of freedom and <i>P</i> value noted <i>Give P values as exact values whenever suitable.</i>
<input checked="" type="checkbox"/>	<input type="checkbox"/> For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
<input checked="" type="checkbox"/>	<input type="checkbox"/> For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
<input checked="" type="checkbox"/>	<input type="checkbox"/> Estimates of effect sizes (e.g. Cohen's <i>d</i> , Pearson's <i>r</i> ), indicating how they were calculated

Our web collection on [statistics for biologists](#) contains articles on many of the points above.

Software and code

Policy information about [availability of computer code](#)

Data collection	We used Python 3.8 together with networkx 3.2.1 and numpy 1.21.6 packages for data collection. We used PyCharm 2023.1.2 and Visual Studio Code 1.83.1 softwares.
Data analysis	We used Python 3.8 together with torch 2.1.1, tqdm 4.66.1, h5py 3.10.0, matplotlib 3.8.2, networkx 3.2.1, numpy 1.21.6, pandas 2.0.3, scipy 1.11.4, and sklearn 0.0 packages for data analysis. We used PyCharm 2023.1.2 and Visual Studio Code 1.83.1 softwares. We also used PI-GNN algorithm as a baseline. The code has been made publicly available at <a href="https://codeocean.com/capsule/6230809/tree/v1">https://codeocean.com/capsule/6230809/tree/v1</a>

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors and reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Portfolio [guidelines for submitting code & software](#) for further information.

Data

Policy information about [availability of data](#)

- All manuscripts must include a [data availability statement](#). This statement should provide the following information, where applicable:
- Accession codes, unique identifiers, or web links for publicly available datasets
  - A description of any restrictions on data availability
  - For clinical datasets or third party data, please ensure that the statement adheres to our [policy](#)

In this paper, we used publicly available datasets of APS [1], NDC [2], Gset [46], and SATLIB [20], together with

synthetic hypergraphs and graphs. The procedure under which the synthetic hypergraphs and graphs are generated is explained throughout the paper. Some examples of the synthetic hypergraphs are provided with the code at <https://codeocean.com/capsule/6230809/tree/v1>.

## Human research participants

Policy information about [studies involving human research participants and Sex and Gender in Research](#).

Reporting on sex and gender

Population characteristics

Recruitment

Ethics oversight

Note that full information on the approval of the study protocol must also be provided in the manuscript.

## Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☐ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see [nature.com/documents/nr-reporting-summary-flat.pdf](https://nature.com/documents/nr-reporting-summary-flat.pdf)

## Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	<i>Describe how sample size was determined, detailing any statistical methods used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient.</i>
Data exclusions	<i>Describe any data exclusions. If no data were excluded from the analyses, state so OR if data were excluded, describe the exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.</i>
Replication	<i>Describe the measures taken to verify the reproducibility of the experimental findings. If all attempts at replication were successful, confirm this OR if there are any findings that were not replicated or cannot be reproduced, note this and describe why.</i>
Randomization	<i>Describe how samples/organisms/participants were allocated into experimental groups. If allocation was not random, describe how covariates were controlled OR if this is not relevant to your study, explain why.</i>
Blinding	<i>Describe whether the investigators were blinded to group allocation during data collection and/or analysis. If blinding was not possible, describe why OR explain why blinding was not relevant to your study.</i>

## Behavioural & social sciences study design

All studies must disclose on these points even when the disclosure is negative.

Study description	<i>Briefly describe the study type including whether data are quantitative, qualitative, or mixed-methods (e.g. qualitative cross-sectional, quantitative experimental, mixed-methods case study).</i>
Research sample	<i>State the research sample (e.g. Harvard university undergraduates, villagers in rural India) and provide relevant demographic information (e.g. age, sex) and indicate whether the sample is representative. Provide a rationale for the study sample chosen. For studies involving existing datasets, please describe the dataset and source.</i>
Sampling strategy	<i>Describe the sampling procedure (e.g. random, snowball, stratified, convenience). Describe the statistical methods that were used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient. For qualitative data, please indicate whether data saturation was considered, and what criteria were used to decide that no further sampling was needed.</i>
Data collection	<i>Provide details about the data collection procedure, including the instruments or devices used to record the data (e.g. pen and paper, computer, eye tracker, video or audio equipment) whether anyone was present besides the participant(s) and the researcher, and whether the researcher was blind to experimental condition and/or the study hypothesis during data collection.</i>

Timing	Indicate the start and stop dates of data collection. If there is a gap between collection periods, state the dates for each sample cohort.
Data exclusions	If no data were excluded from the analyses, state so OR if data were excluded, provide the exact number of exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.
Non-participation	State how many participants dropped out/declined participation and the reason(s) given OR provide response rate OR state that no participants dropped out/declined participation.
Randomization	If participants were not allocated into experimental groups, state so OR describe how participants were allocated to groups, and if allocation was not random, describe how covariates were controlled.

## Ecological, evolutionary & environmental sciences study design

All studies must disclose on these points even when the disclosure is negative.

Study description	Briefly describe the study. For quantitative data include treatment factors and interactions, design structure (e.g. factorial, nested, hierarchical), nature and number of experimental units and replicates.
Research sample	Describe the research sample (e.g. a group of tagged <i>Passer domesticus</i> , all <i>Stenocereus thurberi</i> within Organ Pipe Cactus National Monument), and provide a rationale for the sample choice. When relevant, describe the organism taxa, source, sex, age range and any manipulations. State what population the sample is meant to represent when applicable. For studies involving existing datasets, describe the data and its source.
Sampling strategy	Note the sampling procedure. Describe the statistical methods that were used to predetermine sample size OR if no sample-size calculation was performed, describe how sample sizes were chosen and provide a rationale for why these sample sizes are sufficient.
Data collection	Describe the data collection procedure, including who recorded the data and how.
Timing and spatial scale	Indicate the start and stop dates of data collection, noting the frequency and periodicity of sampling and providing a rationale for these choices. If there is a gap between collection periods, state the dates for each sample cohort. Specify the spatial scale from which the data are taken
Data exclusions	If no data were excluded from the analyses, state so OR if data were excluded, describe the exclusions and the rationale behind them, indicating whether exclusion criteria were pre-established.
Reproducibility	Describe the measures taken to verify the reproducibility of experimental findings. For each experiment, note whether any attempts to repeat the experiment failed OR state that all attempts to repeat the experiment were successful.
Randomization	Describe how samples/organisms/participants were allocated into groups. If allocation was not random, describe how covariates were controlled. If this is not relevant to your study, explain why.
Blinding	Describe the extent of blinding used during data acquisition and analysis. If blinding was not possible, describe why OR explain why blinding was not relevant to your study.

Did the study involve field work? ☐ Yes ☒ No

## Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

### Materials & experimental systems

n/a	Involved in the study
<input checked="" type="checkbox"/>	<input type="checkbox"/> Antibodies
<input checked="" type="checkbox"/>	<input type="checkbox"/> Eukaryotic cell lines
<input checked="" type="checkbox"/>	<input type="checkbox"/> Palaeontology and archaeology
<input checked="" type="checkbox"/>	<input type="checkbox"/> Animals and other organisms
<input checked="" type="checkbox"/>	<input type="checkbox"/> Clinical data
<input checked="" type="checkbox"/>	<input type="checkbox"/> Dual use research of concern

### Methods

n/a	Involved in the study
<input checked="" type="checkbox"/>	<input type="checkbox"/> ChIP-seq
<input checked="" type="checkbox"/>	<input type="checkbox"/> Flow cytometry
<input checked="" type="checkbox"/>	<input type="checkbox"/> MRI-based neuroimaging