Embeddings for Optimization Modulo Theories Learned by Graph Neural Network Guided Solvers are Robust to Logical Space Perturbations

Chirag Sharma

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-143
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-143.html

May 12, 2023

Embeddings for Optimization Modulo Theories Learned by Graph Neural Network Guided Solvers are Robust to Logical Space Perturbations

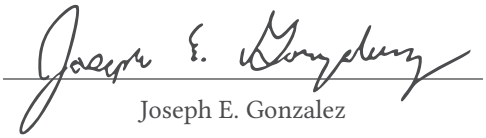
Chirag Sharma

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee



Joseph E. Gonzalez
Research Advisor

5/11/2023
(Date)

★ ★ ★ ★ ★ ★ ★

Sanjit A. Seshia
Second Reader

5/12/2023
(Date)

Embeddings for Optimization Modulo Theories Learned by Graph Neural Network Guided Solvers are Robust to Logical Space Perturbations

by

Chirag Sharma

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Joseph E. Gonzalez, Chair
Professor Sanjit A. Seshia

Spring 2023

Embeddings for Optimization Modulo Theories Learned by Graph Neural Network Guided Solvers are Robust to Logical Space Perturbations

Abstract

Embeddings for Optimization Modulo Theories Learned by Graph Neural Network Guided Solvers are Robust to Logical Space Perturbations

by

Chirag Sharma

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Joseph E. Gonzalez, Chair

Optimization Modulo Theories (OMT) is a highly expressive class of combinatorial optimization problems that can be used to formulate many computationally hard real-world problems in areas such as scheduling, transportation, resource allocation, and supply chain management. Solving OMT problems in practice requires careful manual design of general heuristics but for specialized applications, this can be replaced with an automated machine learning framework that learns to predict OMT problem solutions from task-specific datasets. We encode OMT problems as graphs and train Graph Convolution Networks on the task of directly predicting optimal assignments to integer variables via a classification objective. We introduce a logical space perturbation technique for sampling slightly modified versions of a given OMT problem and show that the embeddings of OMT problems that are learned via our training approach capture sufficient latent structural information about the problems to be robust to these perturbations. We demonstrate this claim by generating training data curriculums for the problem classes of DAG Multiresource Task Scheduling and the Multi-Agent Traveling Saleseman Problem and running experiments to compare the variable prediction accuracy of the direct training approach, training on an augmented dataset that includes a large number of perturbation examples, and finetuning a pretrained model that has learned to identify structural differences between a problem and perturbations of it. In all experiments, the direct training approach performs just as well as the data augmentation and pretraining approaches.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

I would like to thank some of the many people who have supported me and helped me grow over the past 5 years at Berkeley. Thank you to my advisor Professor Joey Gonzalez for giving me the opportunity to continue this research project that I picked up as an undergraduate through the past year as a masters student. Thank you to Justin Wong for being an amazing research mentor, friend, and collaborator over the past 2 years. Thank you to my fellow 5th year masters student and research collaborator Mohamed Elgharbawy. Thank you to Professor Sanjit Seshia for feedback and suggestions on this thesis. Finally, thank you to all of the phenomenal Berkeley professors who inspired me, my college friends who made the past few years the most fun and memorable experience of my life, my family for supporting me and getting me to where I am today, and the city of Berkeley for being a newfound home.

Chapter 1

Introduction

Combinatorial optimization (CO) problems are regularly formulated in industry to represent and solve problems in areas including but not limited to scheduling, transportation, resource allocation, and supply chain management. However, many CO problems are NP-hard and as a result, finding exact solutions to them is highly computationally inefficient in the worst case. Moreover, for many CO problems, such as the Traveling Salesman Problem (TSP), Graph Coloring (GC), and Set Covering (SC), it is computationally inefficient in the worst case even to find approximate solutions with a guaranteed approximation ratio [22] [24]. Nevertheless, these problems can often be solved exactly and efficiently in practice, using heuristics obtained by exploiting special structures in typical instances of the problem. For example, Li et al. [20] introduce a symmetry-breaking constraint for Multi-Agent Path Finding that can exponentially reduce the search space and thereby speed up solve times on certain instances of the problem. Thus, the task of designing CO solvers that identify and use good heuristics based on the distribution of input problems is highly relevant and necessary for efficiently computing exact solutions.

Mixed Integer Linear Programming (MILP) formulations of CO problems are widely used in practice and heuristic-based general-purpose solvers such as Gurobi are carefully designed to solve large problems at scale [14]. Integer programming (IP), which is a special case of MILP, is well-known to be NP-hard. Optimization Modulo Theories (OMT) further generalize the MILP formulation to allow for more complicated constraints involving first-order logic quantifiers, such as conjunctions and disjunctions. State-of-the-art OMT solvers such as νZ [3] and OptiMathSAT [30] have also been developed but in practice, OMT problems are approximated using MILP solvers. The decision version of OMT, Satisfiability Modulo Theories (SMT), is itself typically NP-hard, and in many cases even undecidable [2]. Thus, worst-case theoretical analysis of solving OMT does not provide helpful characterization of practical performance. Moreover, designing general purpose solvers such as the ones mentioned is a hard task because it requires using manually designed heuristics and expert knowledge to achieve fast solve times.

Practical solvers such as νZ and Gurobi are able to efficiently find solutions to many typical CO instances by using heuristics to decrease the size of the solution search space, such as by identifying symmetries and invariances in the constraints. However, since they are general-purpose solvers, these heuristics must be carefully and intentionally engineered to work across a wide range of problems without sacrificing performance on any one type of problem. While this approach has its benefits, in many situations it is preferable to have a specialized solver that achieves highly efficient performance in one class of problems while potentially sacrificing performance in all other classes of problems. For example, a delivery company that regularly needs to solve TSPs for optimal delivery routes in a certain city does not need a general-purpose OMT solver but rather a solver that is highly efficient on instances that they need to solve. In fact, even a general TSP solver is likely too broad since instances run by the company likely share a certain substructure (for example, the distances from the warehouse to different locations are always the same across instances).

The desire for specialized CO solvers that are catered to a specific problem distribution motivates the use of machine learning (ML) to automate the identification and application of the most useful heuristics for solving. A popular approach towards using ML in this setting is to encode CO problems as graphs and perform graph learning via deep neural networks (DNNs) to learn good graph embeddings and predict variable assignments [25], an approach that we also use in this work. A relatively straightforward approach is to learn graph embeddings end-to-end via the natural training objective of using them to predict the optimal variable assignment and then backpropagating losses through the learned embeddings. However, the complex nature of the task suggests that embeddings learned in this fashion may be brittle and may not generalize well to similar but unseen examples. That is, there are no guarantees that a model trained on a dataset of OMT problems will produce similar embeddings for a problem in the dataset and a minimally perturbed version of it. For example, in image data, which is often semantically complex, evaluating models on trained images which have been very slightly perturbed adversarially can cause highly inaccurate predictions [6]. One approach towards learning more robust embeddings is to train on datasets that include examples of augmentations and perturbations. This simple solution often works well in practice [27]. An alternative approach, in line with the recent popularity of pretraining methods, is to first learn embeddings through training on another pretraining task with high data availability [35], and then finetune them on the downstream task of variable assignment prediction. The pretraining approach has been shown to induce more robust embeddings in many situations [15].

In this work, we show that the simple end-to-end training approach for optimal variable assignment prediction in OMT problems is sufficient for learning good quality embeddings that are robust to small input perturbations. That is, using data augmentation or pretraining and finetuning approaches as described does not improve the downstream accuracy of the model compared to the naive end-to-end approach. To this end, we introduce a logical space perturbation technique, wherein the logical structure of a solved instance is extracted

as a 'Boolean backbone', and then perturbed in a logically consistent way, to produce a modified problem. We also introduce a pretraining task for learning graph embeddings for OMT problems via a denoising approach involving classifying a perturbed problem as optimal, suboptimal, or unsatisfiable, relative to the cost of the original problem. We show that the embeddings learned via data augmentation or pretraining with the introduced perturbation technique perform no better on downstream variable prediction than direct end-to-end training on 2 tasks: the Multi-Agent Traveling Salesman Problem (MATSP) and DAG Multiresource Task Scheduling (DMTS). Our work builds on top of the Ashera ML-guided OMT solver in Wong et al. [32].

Chapter 2

Background and Related Work

2.1 Optimization Modulo Theories (OMT)

Optimization Modulo Theories extend SMT by adding an optimization objective function over the feasible region of the configuration space. An SMT problem is a formulation of the problem of deciding the satisfiability of a first-order logic formula within a background theory or combination of theories [2]. For the purposes of this work, we focus on formulas within the theory of Linear Integer Arithmetic (LIA) [4], which guarantees decidability. Moreover, we only consider problems in which the variables are constrained to be integral (or Boolean). These can easily be extended to real-valued or floating point variables by specifying a tolerance δ.

We define *atoms* or *atomic formulas* in LIA to be linear inequalities of the form

$$atom(\vec{x}) := \vec{a} \cdot \vec{x} \bowtie b \tag{2.1}$$

where $\bowtie \in \{<, \leq, =, \geq, >\}, b \in \mathbb{R}, \vec{a} \in \mathbb{Z}^n$, and each $x_i \in \mathbb{Z}$ or $\{0, 1\}$. Here, n denotes the number of variables under consideration.

We subsequently build up constraints or *clauses* as formulas in disjunctive normal form (DNF), i.e., each constraint is a disjunction of conjunctions of atoms. So each clause takes the form

$$clause_i = \bigvee_j conjunction_j \tag{2.2}$$

where each $conjunction_j$ takes the form

$$conjunction_j = \bigwedge_k atom_k \tag{2.3}$$

Finally, the formula in LIA is a collection of clauses or constraints that must all be satisfied, i.e.,

$$formula(\vec{x}) := \bigwedge_i clause_i \tag{2.4}$$

Thus, an SMT problem is the problem of deciding whether the following statement is true:

$$\exists \vec{x} \in \mathbb{R}^n \text{ s.t. } formula(\vec{x}) = True \tag{2.5}$$

and finding a feasible $\vec{x}$ if it exists. An OMT problem is then of the form:

$$\begin{aligned} &\min_{\vec{x} \in \mathbb{R}^n} C(\vec{x}) \\ &\text{s.t. } formula(\vec{x}) = True \end{aligned} \tag{2.6}$$

where $C(\vec{x}) = \vec{c} \cdot \vec{x}$ is a linear objective function with integral coefficients $\vec{c} \in \mathbb{Z}^n$. Note that a solution to the OMT problem does not exist if the formula is not satisfiable. Furthermore, there are no convexity guarantees in OMT, so disjunctions can lead to disconnected local optima. Hence, an OMT solver may need to keep track of disconnected feasible regions at all times while searching for the globally optimum solution.

νZ OMT solver

The νZ solver for OMT iteratively uses the Z3 [7] SMT solver to find strictly better feasible solutions with respect to the objective function and performs a local coordinate search to improve the solution further. It uses carefully engineered MaxSAT and propositional SAT solvers to efficiently solve OMT problems that have subproblems that can be reduced to one of these 2 types of problems. For general OMT problems that do not fall into these domains, the performance is typically poor.

Z3 solves SMT problems using a two-level procedure that first chooses a candidate variable assignment, which induces a Boolean backbone for the problem, and then locally searches for a feasible solution amongst the assignments that are consistent with the Boolean backbone. A *Boolean backbone* $\mathcal{B}$ corresponding to an assignment $\mathcal{A}$ to $\vec{x}$ is simply a mapping from each atom, conjunction, and clause to its truth value under $\mathcal{A}$. If no solutions are found after picking a specific backbone, Z3 picks another candidate assignment (and induced backbone) and locally searches again while also factoring in any conflict information learned from previous searches.

This notion of a one-to-one relationship between a variable assignment and its induced backbone motivates our approach of backbone-level perturbation of problems in this work.

2.2 Mixed Integer Linear Programming (MILP)

Mixed integer linear programs are an NP-hard class of optimization problems that are a hybrid between linear programs and integer linear programs. They can be formulated as follows:

$$\begin{aligned} &\min_{\vec{x}} \vec{c} \cdot \vec{x} \\ &\text{s.t. } A\vec{x} \leq \vec{b} \\ &\vec{x} \geq 0 \end{aligned} \tag{2.7}$$

where $A \in \mathbb{R}^{m \times n}, \vec{b} \in \mathbb{R}^m, \vec{c} \in \mathbb{R}^n$, and each $x_i \in \mathbb{R}$ or $\mathbb{Z}$. If $\vec{x} \in \mathbb{R}^n$, this reduces to a linear program, and if $\vec{x} \in \mathbb{Z}^n$, this reduces to an integer linear program. If all numbers involved in the constraints are integers, then we can also include strict inequalities by converting $\vec{a_i} \cdot \vec{x} < b_i$ into the inequality $\vec{a_i} \cdot \vec{x} \leq b_i - 1$.

Big-M Relaxation of OMT to MILP

Observe that if we restrict ourselves to OMT problems in LIA, as defined above, in which all atoms involve integer coefficients, the MILP formulation is capable of representing everything except the disjunctions. However, we can use a Big-M method to relax disjunctions into a conjunction of inequalities that represent the same solution space in practice [13].

Suppose we have a disjunctive clause of the form

$$(\vec{a_1} \cdot \vec{x} \leq b_1) \vee (\vec{a_2} \cdot \vec{x} \leq b_2) \tag{2.8}$$

We can introduce a binary decision variable $\alpha \in \{0, 1\}$ to convert this into the (conjunction of) 2 constraints

$$\begin{aligned} \vec{a_1} \cdot \vec{x} &\leq b_1 + \alpha M \\ \vec{a_2} \cdot \vec{x} &\leq b_2 + (1 - \alpha) M \end{aligned} \tag{2.9}$$

for some very large constant M. Ideally, $M = \infty$, in which case $\alpha = 0$ selects the first constraint since the second constraint becomes $\vec{a_2} \cdot \vec{x} \leq \infty$, which is vacuously true. Similarly, $\alpha = 1$ selects the second constraint. In practice, since M must be finite, the Big-M reformulation is not exactly equivalent to the original disjunctive constraint since it introduces additional slack by expanding the feasible region, and hence is termed a relaxation.

It is straightforward to apply this Big-M relaxation to more complicated disjunctive clauses by using multiple decision variables and replicating them across atoms in a conjunction. Thus, the OMT problems that we are concerned with, which are in LIA and only involve integer coefficients, can be fairly tightly relaxed to MILP problems. Since MILP solvers such as Gurobi are significantly faster than OMT solvers like νZ in practice, good approximate solutions to OMT problems can be quickly obtained this way.

MILP solvers

Solvers for MILP, such as Gurobi, typically use the branch and bound technique, which makes multiple calls to an LP solver to solve a growing tree of LP relaxations of the original MILP problem. Various heuristics are employed at each step to generate new LP relaxations by selecting a variable to branch on and prune exsiting branches that are predetermined to not lead to a globally optimum solution [17]. Branch and bound methods are known to perform poorly on problems with multiple disjoint optima since symmetric branches cannot be pruned before being fully explored. In practice, expert knowledge is required in these situations to perform intelligent symmetry breaking. Ideally, a machine learner trained to solve specific classes of MILP problems should learn good strategies for symmetry breaking and therefore achieve faster solve times across the board.

2.3 Graph Learning for Combinatorial Optimization

As the use of deep neural networks for graph learning has picked up in recent years in fields such as social networks, bioinformatics, economics, and computer vision [34], it has also increasingly led to much progress in the field of machine learned CO solvers.

Graph Neural Networks (GNNs) are DNNs that process structured graph data as inputs and make predictions at the node level, edge level, or at the aggregate graph level [33]. The input graph may contain any number of features for each node and edge. GNNs learn node embeddings via multiple rounds of message passing in which a node's embedding is updated based on a learned function of neighboring nodes' embeddings and incident edge features [11].

Graph Convolution Networks (GCNs), inspired by Convolutional Neural Networks (CNNs), have had more widespread success in learning to solve CO problems such as TSP [18], network planning [38], and chip placement [23]. Jian-Ya Ding et al. [9] used GCNs to develop a machine-learning accelerated integer program solver that outperforms state of the art generic solvers. Similar to CNNs, GCNs apply a layer-wise learned function to propagate the input at each layer, which enables the identification of patterns in connected subgraphs [19].

2.4 Data Augmentation and Pretraining for Graph Learning

Data augmentation techniques for images are frequently used for training dataset preprocessing in computer vision because of their ease of implementation and observed benefits. Perturbations applied to images include simple transformations, such as rotation, flipping, and cropping, as well as more advanced augmentations, such as masking and noise addition [36]. However, applying perturbations to graph data is a much less trivial task because node

connectivity is highly irregular and non-Euclidean. Moreover, since graphical data is less structured than images and natural language – in particular, both images and text can be encoded as graphs – there is no universal method for augmenting or perturbing graphs for learning. Thus, graph data augmentation is typically highly application specific and non-standardized and is hence an important experimental design consideration [37].

In recent years, pretrained language models such as BERT [8] and GPT [5] have had a massive impact in the domain of natural language processing (NLP) [26]. These huge models are initialized and pre-trained in an unsupervised fashion on large corpuses of largely unstructured text data to learn good universal language representations. They are then fine-tuned, i.e., trained again but initialized with previously obtained weights, on a smaller task-specific annotated dataset. In the context of NLP, examples of downstream tasks of interest include translation, sentiment classification, and summarization. Large pretrained models have also become popular in the field of computer vision (CV).

Inspired by the success of pretraining in NLP and CV, there has been recent work on developing pretrained models for representation learning with graphical data. Hu et al. [16] pretrained a Graph Isomorphism Network (GIN) encoder on a variety of node-level and graph-level tasks and displayed high expressive power and improved performance on downstream tasks. Large pretrained GNN models like GROVER [29] and MPG [21] have also had success in learning complex universal molecular graph representations. The pretraining approach across fields has been shown to lead to better generalization performance, faster convergence on downstream tasks with less labeled data, and more robust embeddings. However, graph-based pretraining so far has not yet achieved the same widespread level of success that it has in other areas like NLP and CV. This is likely due to graphical data being less structured than images and natural language, as discussed above. In particular, both data augmentation and pretraining in graphs are markedly different from their counterparts in other fields and current methods are highly specialized to individual learning tasks.

In our work, we are particularly interested in whether data augmentation and pretraining can lead to better generalization and embeddings. Specifically, by carefully designing an input perturbation technique for augmentation and a pretraining objective for representation learning, we can obtain models that should in principle be better-suited to the final downstream task of predicting the solutions to CO problem instances. This then provides grounds for comparison with the naive end-to-end training approach. We discuss our approach towards augmentation and pretraining in Chapter 3.

Chapter 3

Approach

We seek to define a data augmentation process via small perturbations that largely preserve the structure of the original OMT problem while making some small change to its feasible region. Non-convex optimization problems are inherently complex because of the possibility of large numbers of disconnected feasible regions with multiple possible optima. The perturbation that we implement (seeSection 3.1) is performed at the logical level for some clause in the input OMT problem's constraints. Since clauses are disjunctions that involve multiple generally disconnected regions in variable space, a model that learns to generalize to these perturbations must learn some *useful* embeddings of OMT problems. The learning process in this case can be either training on augmentations directly or through a pretraining objective that involves distinguishing between perturbations (similar to contastive learning [28]). We explore both possibilities in our experiments. We define useful embeddings to be embeddings of graphs that capture sufficient structural information about the CO problems that the graphs encode so that there is some notion of similarity between embeddings for perturbations of the same problem.

Formally, let Π be the set of all OMT problems of the form described in Section 2.1 and Γ be the set of all undirected graphs. Furthermore, let $G : \Pi \to \Gamma$ be an injective map encoding each OMT problem as an undirected graph. We describe how to construct such a map in Section 3.3. Then, a useful embedding map $e : \Gamma \to \mathbb{R}^m$ for some fixed high dimensionality m satisfies that for any *valid* similarity metric $d(\cdot, \cdot)$ defined on Π, there exists a similarity metric $s(\cdot, \cdot)$ on $\mathbb{R}^m$ so that $s(e(G(P)), e(G(P'))) \propto d(P, P')$.

Here, "valid" similarity metrics on Π are those that measure the logical similarity of problems in some way but are unknown to us. Thus, we need some proxy metric to experimentally evaluate the quality of learned embeddings. We decided to use downstream performance on optimal variable assignment prediction as the evaluation metric. Specifically, we show that training on an augmented dataset and pretraining as described in Section 3.2 both do not significantly outperform training a model directly on the non-augmented dataset in terms of accuracy. The evaluation pipeline is detailed in Section 3.7.

3.1 Data Augmentation Method

We consider the setting where the input problems to our neural-guided OMT solver can be modeled as coming from some data distribution $\mathcal{D}$. We assume that we can sample from this distribution, either via sampling uniformly from historical queries, or if the distribution is known, via simulated problem generation. We can thus construct a training dataset $\mathcal{D}_{train}$ of n OMT problems from a certain class, e.g. MATSP or DMTS, where we can run either νZ or Gurobi (via the Big-M relaxation) on each problem $P_i \in \mathcal{D}_{train}$ to get the optimal assignment $\mathcal{A}_i$ and cost C_i.

Our data augmentation method involves a random perturbation function $\rho : \Pi \to \Pi$, which given an OMT problem $P_i \in \mathcal{D}_{train}$ outputs a perturbed version of it, P_i', where the perturbation is a backbone-level logical perturbation done at a single randomly chosen disjunction of P_i. Algorithm 1 details how ρ is implemented. We use $\mathcal{B}[k]$ to denote indexing into the entry corresponding to key k in the backbone map $\mathcal{B}$. For each training curriculum $\mathcal{D}_{train}$, we construct an augmented pretraining dataset $\mathcal{D}_{aug}$ using ρ to obtain a random backbone-level perturbation $P_i^{(j)} \sim \rho(P_i)$ from problems $P_i \in \mathcal{D}_{train}$. $\mathcal{D}_{aug}$ expands D_{train} by a factor of k by including k sampled perturbations $P_i^{(1)}, \ldots, P_i^{(k)} \sim \rho(P_i)$ for each P_i.

Observe that in Algorithm 1, the problems sampled from ρ always contain no disjunctions since we simplify them by converting them into the set of constituent atoms that are marked true in the backbone. Thus, these problems are smaller and simpler MILP problems that can be solved directly by Gurobi; we can hence efficiently obtain assignment labels for perturbed problems.

3.2 Formulation of Pretraining Objective

For the pretraining approach, we train on the same augmented data as in $\mathcal{D}_{aug}$. However, each entry in our pretraining dataset $\mathcal{D}_{pretrain}$, is now a *perturbation pair* $(P_i, P_i^{(j)})$. A perturbation pair is a problem and one of its sampled perturbations. Furthermore, for each entry, we associate the classification label $y_i^j = \begin{cases} OPT & C(P_i^{(j)}) = C_i \\ SUBOPT & C(P_i^{(j)}) > C_i \\ UNSAT & P_i^{(j)} \text{ has no feasible solution} \end{cases}$.

By construction again, the perturbed problem cost cannot be better than the original problem's cost. Note that we assumed here that the problem has a cost minimization objective. For maximization, the inequality for $SUBOPT$ is switched. Given the graph encoding for each problem, $G(P_i, \mathcal{A}_i)$, as described in Section 3.3, our goal is to learn a function p_θ that estimates a probability distribution over the 3 classes above conditioned on a given perturbation pair. We parametrize p_θ via a neural network consisting of a standard GCN encoder applied to each problem in the pair, followed by a linear layer and softmax activation applied

Algorithm 1 Random backbone-level problem perturbation ρ

Input: OMT problem $P \in \mathcal{D}_{train}$ with optimal assignment $\mathcal{A}$
Output: Randomly perturbed OMT problem P'

```
P' ← P
B ← backbone(P', A)
D ← uniformly random clause in P'
Remove entry corresponding to D from backbone map B
C ← uniformly random conjunction in D = C ∨ C' ∨ ···
D ← C                                              ▷ Drop other conjunctions
for atom A in C = A ∧ A' ∧ ··· do
    B[A] ← True
end for
for clause E in P' do                              ▷ Recompute backbone
    for conjunction F in E do
        B[F] ← ⋀_{A∈F} B[A]
    end for
    B[E] ← ⋁_{F∈E} B[F]
end for
for clause E = F_1 ∨ F_2 ∨ ··· in P' do            ▷ Simplify disjunctions
    S ← {F_i : B[F_i] = True}
    Drop E from P' constraints
    for F_i in S do
        Add each atom A_j ∈ F_i as a constraint in P'
    end for
end for
return P'
```

to the concatenation of the GCN embeddings to perform the classification. Since we are performing graph-level classification, we use a global mean pooling layer before concatenation to get graph-level embeddings instead of node-level embeddings for performing classification. We learn the function by minimizing the standard cross-entropy loss via gradient descent:

$$\mathcal{L} = -\frac{1}{nk} \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{y=1}^{3} \mathbb{I}_y[y_i^j] \cdot \log p_\theta(y|(P_i, P_i^{(j)})) \tag{3.1}$$

where $\mathbb{I}_y[y_i^j] = 1$ if $y_i^j = y$ and 0 otherwise and $y = 1, 2, 3$ represents the 3 classes above.

3.3 Graph Representations for OMT

Gasse et al. [10] encodes MILP problems as bipartite graphs with edges connecting nodes representing constraints to nodes representing the variables that participate in them. We ex-

tend upon their method to encode OMT problems P_i with optimal assignment $\mathcal{A}_i$ as graphs $G(P_i, \mathcal{A}_i)$. We encode each variable, atomic formula, conjunction, and clause as a node in the graph, as well as the final formula. Then, for each atom $\vec{a} \cdot \vec{x} \bowtie b$, we add an edge between the node for each x_i and the node for the atom, with weight a_i. We also add b as an attribute for the atom node. Each conjunction node is connected to its constitutent atom nodes with unit weight edges as a tree. Similarly, each clause is represented as a tree of conjunction nodes rooted at the clause node and the final formula is represented as a tree of clause nodes rooted at the formula node. Finally, we also have a special cost node, which is connected to each x_i node with an edge with weight c_i. Additionally, for each variable node, we add a variable ID as an attribute to ensure that the learned embeddings are invariant to relabeling and to preserve all information for reconstructing the problem.

Note that the encoding above doesn't depend on the assignment $\mathcal{A}_i$. However, for our pretraining datasets, we also add a *backbone_val* attribute to each atom, conjunction, and clause node in the graph. Recall that the Boolean backbone corresponding to an assignment $\mathcal{A}$ for a problem P, $backbone(P, \mathcal{A})$ is a mapping from each atom, conjunction, and clause in P to its truth value under the assignment $\mathcal{A}$. Since each backbone value is either 0 or 1 given an assignment, we set the attribute to 0.5 for all nodes during downstream evaluation, when we don't have access to the backbone for a given test problem.

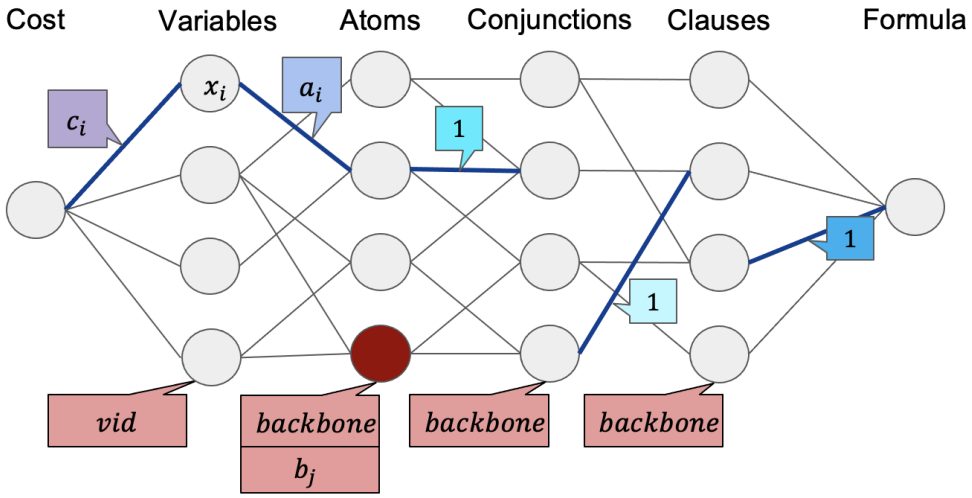


Figure 3.1: Graph encoding used to represent OMT problems as defined in Section 2.1. Variables and constraints are represented as nodes in a weighted undirected graph, with final constraints being hierarchically represented as trees of inner level constraints (atoms, conjunctions, and clauses).

3.4 DMTS Formulation

The DAG multiresource task scheduling problem belongs to a larger family of scheduling problems. It involves finding the optimal assignment of tasks to resources as well as assigning start times to tasks. Assignments must satisfy possible constraints on both resources and dependencies between tasks. The optimization objective is to maximize slack, i.e., the difference between a task's deadline and its expected completion time. It turns out to often be non-trivial to even find feasible schedules for many instances of DMTS. This family of problems appears in the workflow management platform Apache Airflow [1] and in DAG schedulers such as those used for the dynamic deadline-driven execution model for self driving [12].

We consider a set of N tasks, $T = \{t_i : 1 \leq i \leq n\}$, and a dependency matrix M where $M_{ij} = 1$ if t_i must complete before t_j otherwise 0. We also denote task deadlines and expected runtimes as d_i and e_i, respectively. Furthermore, for each task t_i, let $r_i = 1$ if it requires a GPU and 0 otherwise.

We seek to optimize with respect to two sets of variables, s_i and p_i, which denote the start times and placements of tasks. Specifically, let N_G and N_C be the number of GPUs and CPUs, respectively. Then, $p_i = k$ denotes task i being placed on the kth GPUs if $1 \leq k \leq N_G$ and on the $(k - N_G)$th CPU if $N_G < k \leq N_G + N_C$. Our maximization objective is the total slack

$$\sum_{i=1}^{N} d_i - (s_i + e_i) \tag{3.2}$$

with the following constraints:

- **Non-negativity constraints.** For all i, $s_i \geq 0$.
- **Finish before deadline.** For all i, $s_i + e_i \leq d_i$.
- **Placement constraints.** For all i, if $r_i = 1$, $1 \leq p_i \leq N_G$, else $1 \leq p_i \leq N_G + N_C$.
- **Dependency constraints.** For all i, j, if $M_{ij} = 1$, $s_i + e_i \leq s_j$.
- **Exclusion.** For all i, j, $p_i = p_j \implies (s_i + e_i \leq s_j \vee s_j + e_j \leq s_i)$.

Note that the last 3 categories of constraints require disjunctive expressions to be written in first-order logic. Thus, DMTS is a good representative problem for OMT.

3.5 MATSP Formulation

The multi-agent traveling salesman problem is a generalization of the well-known NP-complete traveling salesman problem. Variants of it are often used in practice in route

planning applications, such as for package deliveries in warehouse operations. An MATSP instance is specified by a set of W waypoints and V vehicles and the optimization task is to find an ordering of waypoints to be visited by each vehicle such that all waypoints are visited and the total time taken is minimized. Additional weight constraints are also imposed, as described below.

For each vehicle $v \in V$, let u_v be a Boolean variable indicating whether v is used and γ_v be the vehicle weight. Each waypoint $w \in W$ is visited exactly once, at time t_w by a vehicle x_w, which arrives there from a previous waypoint p_w. The time taken for v to go from w to w' is given by an array $\tau_{v,w,w'}$. For each vehicle, we also define an ordering vector $o_{w,v} = k$ if w is the kth waypoint visited by v. For convenience, we additionally define a Boolean array $\mathbf{M}_{v,w,w'}$, where each entry is 1 if vehicle v visits w' after w. Finally, we have a starting waypoint h (called the "harbor"), and a maximum total weight m_{max}. Note that some of the variables are redundant but this choice allows for an easier encoding specification. Our minimization objective is the total time

$$t = \sum_{v \in V, w \in W, w' \in W \setminus \{h\}} \mathbf{M}_{v,w,w'} \cdot \tau_{v,w,w'} \tag{3.3}$$

and the constraints are as follows:

- **All used vehicles start and end at harbor.** For all v, $\sum_{w \in W} \mathbf{M}_{v,w,h} = u_v$ and $\sum_{w' \in W} \mathbf{M}_{v,h,w'} = u_v$.
- **All waypoints visited.** For all $w' \neq h$, $\sum_{v \in V, w \in W} \mathbf{M}_{v,w,w'} = 1$.
- **Deterministic visit order.** For all w, $\sum_{v \in V, w' \in W \setminus \{h\}} \mathbf{M}_{v,w,w'} = 1$.
- **No self loops.** For all v, w, $\mathbf{M}_{v,w,w} = 0$.
- **Waypoints are transitionary.** For all v, w, $(\sum_{w'' \in W} \mathbf{M}_{v,w'',w} = 1) \implies (\sum_{w' \in W} \mathbf{M}_{v,w,w'} = 1)$.
- p_w **consistency.** For all w, $p_w = \sum_{v \in V, w' \in W} w' \cdot \mathbf{M}_{v,w',w}$.
- x_w **consistency.** For all w, $x_w = \sum_{v \in V, w' \in W} v \cdot \mathbf{M}_{v,w',w}$.
- u_v **and M consistency.** For all v, $u_v = \bigvee_{w,w' \in W} \mathbf{M}_{v,w,w'}$.
- **Weight constraint.** $\sum_v u_v \cdot \gamma_v < m_{max}$.
- **Ordering constraints**. For all $w \neq h$, $\bigvee_{w' \in W, v \in V} (\mathbf{M}_{v,w',w} \wedge o_{w',v} = o_{w,v} - 1)$ and for all v, $w \neq h$, $(o_{w,v} = 0) \vee (\bigvee_{w' \in W} \mathbf{M}_{v,w,w'})$.

Note that many of the constraints involve disjunctions (implications can also be reformulated using disjunctions). Thus, MATSP is also a good representative problem for OMT.

3.6 Dataset Generation

We generate multiple training curriculums for DMTS and MATSP. For each curriculum, we generate an augmented dataset $\mathcal{D}_{aug}$ by sampling $k = 20$ perturbations for each $P_i \in \mathcal{D}_{train}$ i.e., $|\mathcal{D}_{aug}| = 20|\mathcal{D}_{train}|$. We also generate the associated pretraining dataset $\mathcal{D}_{pretrain}$. Note that we also have $|\mathcal{D}_{pretrain}| = |\mathcal{D}_{aug}|$.

We in total generate 23,136 DMTS instances across multiple curriculums. For each curriculum, we fix the number of tasks, N, across all instances. We generate curriculums for $N = 6$ to 11. In each curriculum, we vary the number of CPUs available and introduce a single randomly placed dependency between two tasks in each instance. For consistency, we set all instances to have 2 GPUs and all tasks to have the same expected runtime of 15 seconds and release times of 2. Moreover, the symmetry introduced by this decision is known to be hard for traditional MILP solvers to break. To ensure that all instances are feasible, we scale up the deadline as the number of tasks increase.

For MATSP, we generate instances with 2 clusters of waypoints arranged in a polygon. We generate curriculums with 2500 instances each, where the number of waypoints per cluster, M, is fixed in each. Within each curriculum, we vary the distance between the cluster center and the vehicle origin, as well as the radius of the polygon. For consistency, we always set the number of vehicles to be 2 and the first waypoint to be the starting point for both vehicles. We generate curriculums for $M = 3$ to 5.

3.7 Downstream Performance Evaluation

Our downstream task for evaluation is the main solver goal of finding an optimal solution to an input OMT problem by predicting assignments to the variables being optimized over.

As described in Section 3.1, we construct training data curriculums $\{\mathcal{D}_{train}\}$, consisting of graph encodings $G(P_i)$ for n problems in a certain class, and label each with a cost optimal variable assignment $\mathcal{A}_i$, which is not necessarily unique. As mentioned in Section 3.3, unlike in $\mathcal{D}_{pretrain}$, we do not annotate graph nodes in each $\mathcal{D}_{train}$ with optimal Boolean backbone values; instead, we simply set them all to 0.5. In the case of pretrained models, this means that the learner essentially treats them as uniformly random noise variables when finetuning on $\mathcal{D}_{train}$.

We seek to learn a vector-valued function $\mathbf{f}_\theta$ that estimates a probability distribution over potential values for each variable. Since we are dealing only with bounded integer variables for our tasks, we treat potential values as independent classes and train a model to classify each variable. We use the same GCN architecture as in Section 3.2 but now with a multi-class classification decoder head. Moreover, unlike the pretraining task, we are performing

node-level classification on all of the variable nodes in the graph, so we do not use a pooling layer in our architecture. We learn $\mathbf{f}_\theta$ by minimizing the following modified cross-entropy loss via gradient descent:

$$\mathcal{L} = -\frac{1}{nm|V|} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{x \in V} \mathbb{I}_x[(\mathcal{A}_i)_j] \cdot \log f_\theta(x_j = x | P_i) \tag{3.4}$$

where $(\mathcal{A}_i)_j$ is the assignment to the jth variable out of m total variables in the optimal assignment $\mathcal{A}_i$ and $\mathbb{I}_x[(\mathcal{A}_i)_j] = 1$ if $x = (\mathcal{A}_i)_j$ and 0 otherwise. Note that V is the set of potential values for x_j and $f_\theta(x_j = x|P_i)$ is the probability assigned to x in the marginal distribution of $\mathbf{f}_\theta(\mathbf{x}|P_i)$ on the jth coordinate.

For finetuning pretrained models, we slightly modify the architecture. We use a dual encoder architecture consisting of 2 identical GCN encoders followed by a multi-class classification decoder head that takes as input the concatenated outputs of the encoders. We initialize one of the encoders randomly and the other with the weights from the pretrained model. Furthermore, we freeze the weights of the pretrained encoder during finetuning. We find that this approach leads to better stability than directly finetuning a single pretrained encoder.

Chapter 4

Experiments and Results

4.1 Impact of Training Data Augmentation on Evaluation Accuracy

We train models for each generated curriculum on both $\mathcal{D}_{train}$ and $\mathcal{D}_{aug}$ for optimal variable prediction. We use a hidden dimension of 64 across all layers of our GCN models. In principle, training on $\mathcal{D}_{aug}$ should enable the model to learn from a wider data distribution that contains all of the information in $\mathcal{D}_{train}$ as well as the information about the correlation between perturbations as described in Algorithm 1 and the resulting change in the problem solution. We only train on p_i and s_i variables for DMTS and x_w variables for MATSP since the other variables are either dummy variables for constraint simplification or too noisy for prediction.

We report the evaluation set accuracy results for DMTS in Table 4.1 and for MATSP in Table 4.2. Here, accuracy is computed as the the mean across the evaluation set of the mean accuracy per instance, i.e., the average number of variables in the instance for which the model correctly assigns the maximum likelihood value.

We see that in both cases, i.e., DMTS and MATSP, the accuracy with augmentation is not significantly different from the accuracy without augmentation. This indicates that $\mathcal{D}_{aug}$ does not contain any information beyond what is contained in $\mathcal{D}_{train}$ that is useful for the task of variable assignment prediction. Beyond the evaluation task, this further suggests that the OMT problem embeddings learned directly without any augmentation are already maximally useful in that they have learned enough structural information to correctly handle perturbations.

Curriculum	Accuracy without Augmentation	Accuracy with Augmentation
$N = 6$	62.02%	62.19%
$N = 7$	56.87%	57.07%
$N = 8$	60.25%	60.24%
$N = 9$	56.54%	56.66%
$N = 10$	59.31%	57.94%
$N = 11$	55.50%	55.62%

Table 4.1: Comparison of downstream optimal variable prediction accuracy on DMTS curriculums between models trained on original datasets and models trained on perturbation augmented datasets. N is the number of tasks in the scheduling instance.

Curriculum	Accuracy without Augmentation	Accuracy with Augmentation
$M = 3$	82.76%	82.68%
$M = 4$	90.11%	90.95%
$M = 5$	87.48%	86.56%

Table 4.2: Comparison of downstream optimal variable prediction accuracy on MATSP curriculums between models trained on original datasets and models trained on perturbation augmented datasets. N is the number of waypoints in each cluster.

4.2 Finetuning Pretrained Models on Variable Assignment Prediction

To evaluate the quality of pretrained embeddings, we pretrain a GCN model on each $\mathcal{D}_{pretrain}$ as per the learning objective in Section 3.2, and further finetune it on the corresponding $\mathcal{D}_{train}$ as per the finetuning objective in Section 3.7. We again use a hidden dimension of 64 across all layers of our GCN.

Since our pretraining and finetuning are performed on the same data curriculum, i.e., all of the graphs in both the training and pretraining datasets belong to the same structural class, the finetuned model's downstream performance is indicative of the representation of the class that is captured by the pretrained model. Specifically, any observed difference between the final accuracy obtained by a finetuned model and a model trained from scratch on $\mathcal{D}_{train}$ can be wholly attributed to the quality of the pretrained model's embeddings. Due to the complex nature of the optimization problem and landscape, we can expect the initial GCN weights to have an observable impact on the final convergence point.

We report the downstream test accuracy of the base models trained from scratch and the

finetuned models after pretraining in Table 4.3 and Table 4.4 below. We see that for both DMTS and MATSP, the finetuned model has the exact same accuracy as the base model on all curriculums ($M = 3$ for MATSP has slightly improved performance on finetuning but not by much and is likely due to stochasticity). This suggests that the pretrained encoder provides no useful information to the model while finetuning and its embeddings are treated as noise while learning. All of the learning must then happen on the randomly initialized encoder, which reduces it to the exact same setup as the base model.

Curriculum	**Base Model Accuracy**	**Finetuned Model Accuracy**
$N = 6$	62.02%	62.02%
$N = 7$	56.87%	56.87%
$N = 8$	60.25%	60.25%
$N = 9$	56.54%	56.54%
$N = 10$	59.31%	59.31%
$N = 11$	55.50%	55.50%

Table 4.3: Comparison of downstream optimal variable prediction accuracy on DMTS curriculums between pretrained + finetuned models and models trained from scratch. The pretraining and finetuning is done on the same curriculum (pretraining on the augmented curriculum). N is the number of tasks in the scheduling instance.

Curriculum	**Base Model Accuracy**	**Finetuned Model Accuracy**
$M = 3$	82.76%	82.99%
$M = 4$	90.11%	90.11%
$M = 5$	87.48%	87.48%

Table 4.4: Comparison of downstream optimal variable prediction accuracy on MATSP curriculums between pretrained + finetuned models and models trained from scratch. The pretraining and finetuning is done on the same curriculum (pretraining on the augmented curriculum). M is the number of waypoints in each of the 2 clusters, i.e., $|W| = 2M$.

Chapter 5

Conclusion and Future Work

The experiments and results described in Chapter 4 provide evidence for the claim that the OMT problem embeddings learned by the GCN models while naively optimizing for the task of predicting solutions to DMTS and MATSP problems are robust and expressive enough to capture notions of logical space perturbations as per Algorithm 1. We see that both logical space perturbation based augmentation and related pretraining are unsuccessful at beating the performance of direct end-to-end training on problem to optimal variable assignment prediction. Our work thus suggests that improving the performance of neural guided OMT solvers requires a different approach than data augmentation, such as architectural modifications, alternative graph encodings, or modified optimization objectives.

Beyond the mentioned results, this work also makes novel contributions in the area of data augmentation for encodings of CO problems. While our perturbation algorithm makes use of the specific disjunctive structure of OMT problem constraints, the notion of extracting a logical backbone for a problem, applying a simple edit-distance based perturbation to it, and then propagating the resulting modified backbone back through the problem constraints, can be extended to any type of CO problem. Note that this augmentation method is also independent of the use of graph encodings. Furthermore, our experiments contribute benchmarks for evaluating learning-based OMT solvers on directed multiresource task scheduling and multi agent traveling salesman problems.

While our experiments do indeed provide evidence for our chosen data augmentation and pretraining approaches not being capable of boosting the performance of a learned OMT solver, they are fairly limited in scope and can be extended much more to see if the claim holds true more generally. Various schemes can be employed to augment problem instances with a wide variety of structurally similar problems. For example, instead of performing edit distance 1 perturbations on the logical backbone of problems, random perturbations or noisy constraint addition methods can be explored. We can also consider data augmentation in the form of permuting the variable name ordering and relaxing certain variables to be real-valued instead of integral. Moreover, our work used a specific pretraining approach, which may not

necessarily be the best choice. Other ongoing work suggests that contrastive learning as an auxiliary objective during training may improve downstream performance. Identifying the best pretraining task for representation learning is a hard task in general and even more so for our complicated setting. Similarly, our choice of downstream evaluation task may not be the best proxy for evaluating representation learning. Although augmentation and pretraining do not lead to better downstream accuracy, it is possible that the learned embeddings are more useful than in direct training for some other task, such as for example generalizing to unseen curriculums. Finally, newer GNN architectures such as Graph Attention Networks [31] may provide better performance and respond better pretraining compared to GCNs, in the same way that attention-based networks have become synonymous with transfer learning in NLP and CV.

Bibliography

[1] URL: http://airflow.apache.org/.

[2] Clark Barrett et al. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability*. Ed. by Armin Biere et al. Second. IOS Press, 2021. Chap. 33, pp. 1267–1329.

[3] Nikolaj BjØrner, Anh-Dung Phan, and Lars Fleckenstein. "VZ - An Optimizing SMT Solver". In: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 194–199. ISBN: 9783662466803. DOI: 10.1007/978-3-662-46681-0_14. URL: https://doi.org/10.1007/978-3-662-46681-0_14.

[4] Martin Bromberger, Thomas Sturm, and Christoph Weidenbach. *Linear Integer Arithmetic Revisited*. 2020. arXiv: 1503.02948 [cs.LO].

[5] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[6] Ashutosh Chaubey et al. *Universal Adversarial Perturbations: A Survey*. 2020. arXiv: 2005.08087 [cs.CV].

[7] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08. Budapest, Hungary: Springer-Verlag, 2008, pp. 337–340. ISBN: 3540787992.

[8] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[9] Jian-Ya Ding et al. *Accelerating Primal Solution Findings for Mixed Integer Programs Based on Solution Prediction*. 2019. arXiv: 1906.09575 [cs.AI].

[10] Maxime Gasse et al. *Exact Combinatorial Optimization with Graph Convolutional Neural Networks*. 2019. arXiv: 1906.01629 [cs.LG].

[11] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG].

[12] Ionel Gog et al. "D3: A Dynamic Deadline-Driven Approach for Building Autonomous Vehicles". In: *Proceedings of the Seventeenth European Conference on Computer Systems*. Rennes, France: Association for Computing Machinery, 2022, pp. 453–471.

[13] Ignacio E. Grossmann and Francisco Trespalacios. "Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming". In: *Aiche Journal* 59 (2013), pp. 3276–3295.

[14] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[15] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. *Using Pre-Training Can Improve Model Robustness and Uncertainty*. 2019. arXiv: 1901.09960 [cs.LG].

[16] Weihua Hu et al. *Strategies for Pre-training Graph Neural Networks*. 2020. arXiv: 1905.12265 [cs.LG].

[17] Lingying Huang et al. *Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends*. 2021. arXiv: 2111.06257 [cs.LG].

[18] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. *An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem*. 2019. arXiv: 1906.01227 [cs.LG].

[19] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[20] Jiaoyang Li et al. "Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.33016087. URL: https://doi.org/10.1609/aaai.v33i01.33016087.

[21] Pengyong Li et al. "An effective self-supervised framework for learning expressive molecular global representations to drug discovery". In: *Briefings in Bioinformatics* 22 (May 2021). DOI: 10.1093/bib/bbab109.

[22] Carsten Lund and Mihalis Yannakakis. "On the Hardness of Approximating Minimization Problems". In: *J. ACM* 41.5 (Sept. 1994), pp. 960–981. ISSN: 0004-5411. DOI: 10.1145/185675.306789. URL: https://doi.org/10.1145/185675.306789.

[23] Azalia Mirhoseini et al. *Chip Placement with Deep Reinforcement Learning*. 2020. arXiv: 2004.10746 [cs.LG].

[24] Christos H. Papadimitriou and Santosh Vempala. "On the Approximability of the Traveling Salesman Problem (Extended Abstract)". In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*. STOC '00. Portland, Oregon, USA: Association for Computing Machinery, 2000, pp. 126–133. ISBN: 1581131844. DOI: 10.1145/335305.335320. URL: https://doi.org/10.1145/335305.335320.

[25] Yun Peng, Byron Choi, and Jianliang Xu. *Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art*. 2021. arXiv: 2008.12646 [cs.LG].

[26] XiPeng Qiu et al. "Pre-trained models for natural language processing: A survey". In: *Science China Technological Sciences* 63.10 (Sept. 2020), pp. 1872–1897. DOI: 10.1007/s11431-020-1647-3. URL: https://doi.org/10.1007%2Fs11431-020-1647-3.

[27] Sylvestre-Alvise Rebuffi et al. *Data Augmentation Can Improve Robustness*. 2021. arXiv: 2111.05328 [cs.CV].

[28] Nils Rethmeier and Isabelle Augenstein. *A Primer on Contrastive Pretraining in Language Processing: Methods, Lessons Learned and Perspectives*. 2021. arXiv: 2102.12982 [cs.CL].

[29] Yu Rong et al. *Self-Supervised Graph Transformer on Large-Scale Molecular Data*. 2020. arXiv: 2007.02835 [q-bio.BM].

[30] Roberto Sebastiani and Patrick Trentin. "OptiMathSAT: A Tool for Optimization Modulo Theories". In: July 2015, pp. 447–454. ISBN: 978-3-319-21689-8. DOI: 10.1007/978-3-319-21690-4_27.

[31] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[32] Justin Wong et al. *Ashera; Neural Guided Optimization Modulo Theory*. Tech. rep. UCB/EECS-2023-103. EECS Department, University of California, Berkeley, May 2023. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-103.html.

[33] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. DOI: 10.1109/tnnls.2020.2978386. URL: https://doi.org/10.1109%2Ftnnls.2020.2978386.

[34] Feng Xia et al. "Graph Learning: A Survey". In: *IEEE Transactions on Artificial Intelligence* 2.2 (Apr. 2021), pp. 109–127. DOI: 10.1109/tai.2021.3076021. URL: https://doi.org/10.1109%2Ftai.2021.3076021.

[35] Jun Xia et al. *A Survey of Pretraining on Graphs: Taxonomy, Methods, and Applications*. 2022. arXiv: 2202.07893 [cs.LG].

[36] Suorong Yang et al. *Image Data Augmentation for Deep Learning: A Survey*. 2022. arXiv: 2204.08610 [cs.CV].

[37] Tong Zhao et al. *Graph Data Augmentation for Graph Machine Learning: A Survey*. 2023. arXiv: 2202.08871 [cs.LG].

[38] Hang Zhu et al. "Network Planning with Deep Reinforcement Learning". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 258–271. ISBN: 9781450383837. DOI: 10.1145/3452296.3472902. URL: https://doi.org/10.1145/3452296.3472902.