MAGNET: A Multi-Agent Graph Neural Network for Efficient Bipartite Task Assignment

Donald Loveland
University of Michigan
Ann Arbor, United States
dlovelan@umich.edu

James Usevitch
Brigham Young University
Provo, United States
james_usevitch@byu.edu

Zachary Serlin
MIT Lincoln Laboratory
Lexington, United States
zachary.serlin@ll.mit.edu

Danai Koutra
University of Michigan
Ann Arbor, United States
dkoutra@umich.edu

Rajmonda Caceres
MIT Lincoln Laboratory
Lexington, United States
rajmonda.caceres@ll.mit.edu

ABSTRACT

Assignment problems are prevalent in many autonomous agent contexts, where the learning task involves mapping elements from a domain set to a range set. While current state-of-the-art machine learning solutions employ graph neural networks (GNNs) on bipartite agent-task graphs, these approaches frequently fall short when addressing more complex constraints and objectives. To broaden the utility of GNNs for a wider variety of assignment problems, we introduce **MAGNET**—a novel **M**ulti-**A**gent **G**raph **N**eural network designed for **E**fficient **T**ask assignment. MAGNET is composed of three integral components that together deliver enhanced performance: (1) a pre-processor that expands the bipartite graph such that it is amenable to multi-task assignment, (2) an edge-centric GNN, enabled through a line graph transformation, which generates candidate assignments, and (3) a post-processor that filters these candidate assignments to ensure they meet the feasibility criteria. Recognizing that the line graph transformation can affect execution time, we enhance MAGNET's efficiency by incorporating an inference-time pruning strategy. This strategy leverages both GNN scoring and sparsification techniques to streamline the assignment process. Experimental evaluations demonstrate that MAGNET delivers substantial performance improvements over previous GNN-based and heuristic methods, and notably reduces execution time by several orders of magnitude compared to state-of-the-art commercial solvers.

KEYWORDS

Assignment; Graph Neural Network; Bipartite

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

ACM Reference Format:
Donald Loveland, James Usevitch, Zachary Serlin, Danai Koutra, and Rajmonda Caceres. 2025. MAGNET: A Multi-Agent Graph Neural Network for Efficient Bipartite Task Assignment. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

Assignment is a fundamental optimization problem that involves mapping elements from a domain set to a range set, aiming to maximize an objective while adhering to certain constraints. Assignment problems appear in various applications, with the goal of allocating *agents* to *tasks*. Notable examples include vehicle routing [32], robot coordination [49], and exploration and mapping [50], energy conservation in wireless networks [22], and industrial plant inspection [24]. Despite their broad applicability, assignment problems are typically NP-hard due to their combinatorial nature. However, while high-stakes applications can require guaranteed optimality, such as search and rescue [57], many settings can relax optimality to improve efficiency. For example, a near-optimal solution for vehicle routing may enable faster service delivery to most users rather than causing delays for all due to computational constraints. Despite the effort to build efficient algorithms to solve assignment problems [20], execution times often scale poorly with input size, especially when dealing with complex objectives and constraints. Thus, scalable solutions require algorithms that quickly provide high-quality and feasible solutions.

In scenarios where guarantees can be relaxed, deep learning has shown promise in efficiently learning heuristics for assignment. Early approaches employed convolutional and recurrent neural networks on the cost matrix of agent-task pairs [35, 42, 53]. However, these architectures impose an artificial ordering on agents and tasks, ignoring inherent problem invariances. To address this, graph neural networks (GNNs) have been adapted for assignment problems by representing them as weighted bipartite graphs [39, 45]. Despite their promise, GNNs often lack mechanisms to ensure feasible solutions when objectives and constraints become complex. Furthermore, the node-centric focus of GNNs can hinder learning for assignment problems, which require learning over agent-task pairs, or edges. While edge-centric GNNs are still emerging [6], some node-centric GNNs have adopted edge convolutions that update edges with summary statistics or embeddings of node features [39, 52]. Recent advancements have also explored using line graph transformations to learn

edge representations directly [4, 23]; however, these techniques have primarily been applied to node-level tasks.

To address these challenges, we propose **MAGNET**, a **M**ulti-**A**gent **G**raph **N**eural network for **E**fficient **T**ask assignment. MAGNET predicts feasible solutions for the probabilistic multi-agent knapsack problem, encompassing diverse applications and managing both multi- and duplicate-task assignments. MAGNET uses an edge-centric learning strategy via a line graph transformation to determine candidate assignments with GNN-learned edge scores. To reduce the computational cost of this transformation on GNN aggregation, we incorporate an inference-time pruning technique that exploits the assignment solutions' sparse structure, accelerating inference. Feasibility is ensured through post-processing of the GNN outputs. We demonstrate MAGNET's *effectiveness* through comprehensive empirical analysis, comparing it with node-centric GNNs and hand-crafted heuristics, showing enhanced edge scoring. Additionally, we study MAGNET's *efficiency*, revealing significant acceleration compared to MOSEK, a leading commercial solver. Lastly, we show MAGNET *generalizes* to larger problem sizes impractical for traditional solvers, even when trained on smaller problems. Our contributions are:

- We **propose MAGNET**, a GNN framework that expands the class of assignment problems solvable by GNNs and improves on the quality of solutions generated by learning-based algorithms.
- We provide a **comprehensive study** of MAGNET, showing strong performance across problems sizes with superior execution time.
- We demonstrate that **inference-time pruning** can accelerate execution time while maintaining performance quality.

2 BACKGROUND AND NOTATION

In this section, we outline the studied assignment problem and provide notation for GNNs.

2.1 Assignment Problem Formulation

The assignment problem aims to find a mapping between a set of n agents A and a set of m tasks T. Each agent, $a_i \in A$, has an associated capacity, $\alpha_i \in \mathbb{R}_{\geq 0}$, and each task, $t_j \in T$, has an associated reward $\tau_j \in \mathbb{R}_{\geq 0}$, if completed. There is a cost matrix $\mathbf{C} \in \mathbb{R}_{\geq 0}^{n \times m}$ with entries $c_{i,j}$, denoting the cost of a successful interaction between an agent a_i and task t_j. The matrix $\mathbf{P} \in [0,1]^{n \times m}$, with entries $p_{i,j}$, denotes the probability of an agent i successfully accomplishing a task j. Agent-task interactions are assumed to be independent events. The decision variables determining the number of interactions between agents and tasks are represented by a matrix $\mathbf{Y} \in \mathbb{Z}_{\geq 0}^{n \times m}$, where $y_{i,j}$ represents the number of times agent i attempts task j. The objective is to maximize the expected value of completed tasks, while ensuring that each agent's total cost is less than or equal to its capacity. This can be formalized as a mixed-integer convex program:

$$\begin{aligned} \max_{\mathbf{Y}} \quad & \sum_{t_j \in T} \tau_j \left(1 - \prod_{a_i \in A} (1 - p_{i,j})^{y_{i,j}} \right) \\ \text{s.t.} \quad & \sum_{t_j \in T} c_{i,j} y_{i,j} \leq \alpha_i, i \in 1,2,\ldots,n, \\ & y_{i,j} \in \mathbb{Z}_{\geq 0} \forall i,j. \end{aligned} \tag{1}$$

Intuitively, each term represents the success probability for task t_j. These tasks are weighted by their reward τ_j and summed to obtain the total expected reward. The constraints ensure that the total cost of assignments for each agent is less than that agent's capacity α_i, and that the agent-task interactions are discrete. This problem is a mixed-integer convex program, since the objective is convex in each $y_{i,j}$ and the constraints are affine in each $y_{i,j}$. Additionally, this formulation includes several classical problems, including the unbounded knapsack problem when $p_{i,j} = 1$ for all i,j [14], and the weapon-to-target assignment problem when $c_{i,j} = 1$ and $\alpha_i \in \mathbb{Z}$ for all i,j [28].

Remark 2.1. The difficulty of assignment problems can be organized by classes [8]. Previous neural network solvers consider simple (integer) Linear Programs (LP) with equality constraints [2, 39]. LPs are a subset of quadratic programs (QPs), which are further subsets of quadratically constrained quadratic programs (QCQPs), and these, in turn, are a subset of conic programs (CPs). Typically, solving LPs is easier than QPs, QPs are easier than QCQPs, and QCQPs are easier than CPs. This hierarchy also applies to (mixed) integer LPs, QPs, etc. as solving the relaxed problem (where integer constraints are ignored) is a fundamental technique to obtain the mixed integer solution. According to this hierarchy, previously studied problems lie at the bottom of the difficulty spectrum, while the problem in equation (1) is classified as an integer conic program and thus lies at the top of the difficulty spectrum. Consequently, modern optimization software often scales poorly for larger instances of equation (1). To our knowledge, no neural network-based attempts have been made to solve this challenging formulation despite its broad applicability.

2.2 Representing the Optimization Problem

Previous work represents assignment problems as weighted bipartite graphs [16, 39]. Overloading the notation, the graph is denoted as $G = (A,T,E)$, where nodes A encode the agents, and nodes T encode the tasks. Each node $a_i \in A$ has a feature scalar/vector α_i that corresponds to a_i's capacity, and each node $t_j \in T$ has a feature scalar/vector τ_j that corresponds to t_j's reward. The cost matrix is encoded as edge weights on E, where an edge between a_i and t_j, $e_{i,j}$, will have weight $c_{i,j}$. The goal is to predict an assignment matrix $\mathbf{Y}'$ with $y'_{i,j}$ elements denoting the predicted assignments between a_i and t_j, while satisfying that a_i does not surpass capacity α_i. This structure is shown in Figure 1a, and is extended to more complex problems in Section 3.

2.3 Graph Neural Networks

GNNs are designed to operate directly on graph data. In a k-layer GNN, learning occurs through message passing over k-hop neighborhoods of a graph [18]. A node u's representation is updated by iteratively aggregating the features from its 1-hop neighborhood (AGGR) and embedding the aggregated features with u's features, usually through a non-linear transformation parameterized by $\mathbf{W}$ (ENC). The update is expressed for u as $r_u^l = \text{ENC}(r_u^{l-1}, \text{AGGR}(r_v^{l-1}, v \in N(u)))$ for $l \in \{1,2,\ldots,k\}$, where $r_u^0 = x_u$, u's original node features, and $N(u)$ is the set of one-hop neighbors around u. Within AGGR, it is common to modulate the incoming messages from neighboring nodes, either through degree, attention weights, or more complicated statistics. In weighted graphs, the modulation is performed through the edge weights. The update function is applied k times, resulting in updated representations for the node u.

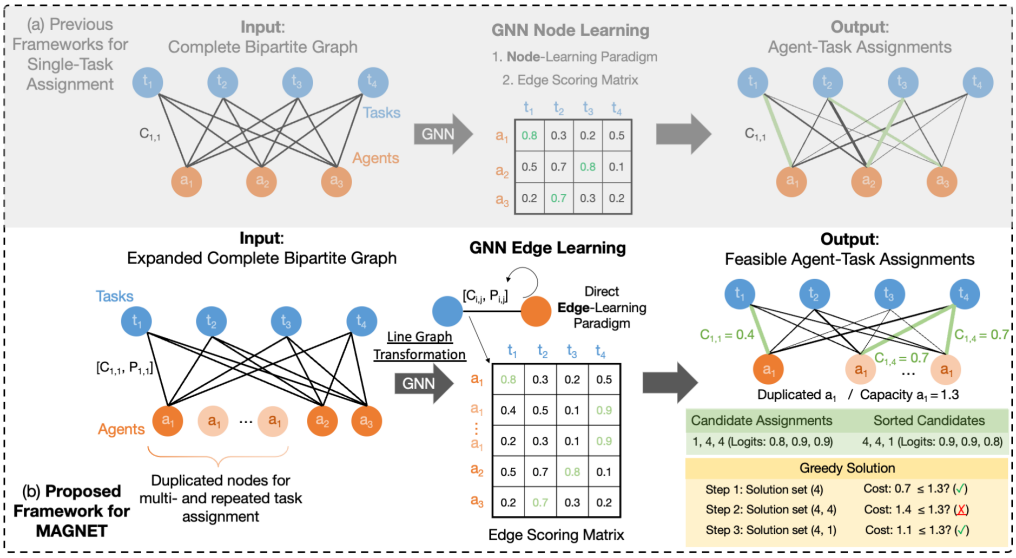

Figure 1: Blue and orange nodes are tasks and agents, respectively. The final assignments are green. *(a)* Single task: The GNN scores each agent-task pair. Then, the continuous scores are transformed into assignments via argmax. *(b)* MAGNET's multi-task: The graph is expanded to enable multiple and repeated task assignment. Then, a line graph transformation is performed to enable direct edge learning. Finally, a greedy assignment procedure, based on the logit scores, ensures feasibility.

3 MAGNET: A FRAMEWORK FOR COMPLEX ASSIGNMENT PROBLEMS

We focus on supervised learning over the assignment task, where a ground truth assignment matrix $\mathbf{Y} \in \mathbb{Z}_{\geq 0}^{n \times m}$ is known. Starting from the weighted bipartite graph discussed in Section 2.2, we extend the scalar edge weights to vectors, encoding both costs and probabilities. The goal is then to find a mapping $F : G \rightarrow \mathbf{Y}'$, parameterized by a GNN, that captures the objective in Equation (1). Here, $\mathbf{Y}'$ represents the predicted assignments with $y'_{i,j}$ indicating an assignment between agent a_i and task t_j. Beyond minimizing the error between $\mathbf{Y}$ and $\mathbf{Y}'$, we focus on ensuring feasibility of $\mathbf{Y}'$. Note that while the optimization objective can be used as a loss function, when differentiable, this might not be available in real-world settings. Instead, we rely on access to historic assignment instances, and use the optimization objective as a post-hoc evaluation of the GNN.

To handle more complex problems, we propose MAGNET, improving upon current assignment GNNs in three ways: (1) MAGNET supports multi-task and probabilistic settings, (2) MAGNET leverages edge-centric learning to better align with the optimization task, (3) MAGNET ensures feasibility. These improvements are integrated as components within MAGNET. First, the bipartite graph is processed to expand nodes, converting the multi-task problem into a binary one. Then, MAGNET advances beyond node-centric approaches [2, 39] by applying a line graph transformation, enabling direct learning over edges. Finally, feasibility is assured through a greedy selection of candidate assignments provided by the edge-based learning component. Details of MAGNET's components are discussed below.

3.1 Probabilistic and Multi-Task Learning

Adapting multi-agent, multi-task assignment problems to GNNs requires outputting a variable number of unbounded positive integers. MAGNET addresses this by preprocessing the agent-task bipartite graph, transforming the problem into a series of binary classifications, as seen in the first step of Figure 1b. Specifically, for an agent a_i with capacity α_i, the maximum number of assignments that can be given to a_i is $\phi_i = \lfloor \alpha_i / \min(c_i) \rfloor$, where c_i is the cost vector across targets for a_i. MAGNET duplicates each agent node into ϕ_i instances, each retaining the same characteristics of their original agent, including the capacity, edges, and edge weights. $\mathbf{Y}$ is also expanded such that each agent's duplicates are labeled with possible assignments until all are allocated. Specifically, the k^{th} duplicate of agent i has label $y_{i,j,k} = 1$ if the agent is assigned to j. If the number of duplicates exceeds the total assignments, extra duplicates do not receive labels. As the local neighborhood of duplicated agents are similar, we use dropout during training and inference to encourage diverse assignments.

3.2 GNNs for Edge Scoring

High-quality solutions for complex assignment problems require a GNN that can effectively score edges. While previous GNNs for assignment rely on node-centric learning with an edge scoring operator, this paradigm fails to directly learn over the decision variables. Instead, edge-centric learning offers a more direct method to learn edge scores without introducing any artificial bottlenecks through fixed operators [19, 26] or additional convolution modules to design and learn over [39]. Thus, our second contribution is facilitating edge-centric learning through a line graph transformation, defined in Definition 3.1, of the weighted bipartite graph, allowing MAGNET to directly update edge representations. MAGNET is the first direct use case of edge-centric learning for assignment and is notably different from previous applications of edge-centric learning – many applications use edge-centric learning to augment node classification tasks or perform link prediction of missing edges [4, 6, 44].

Definition 3.1. A line graph transformation of a graph G results in a graph G', where edges in G are nodes in G', and for every two edges incident to the same node in G, there is an edge between their corresponding nodes in G'. The vertex set $V(G')$ and edge set $E(G')$ are:

$$\begin{aligned} V(G') &= E(G) = \{e_1, \ldots e_k\}, \\ E(G') &= \{(e_i, e_j) : e_i \cap e_j \neq \emptyset\}. \end{aligned} \tag{2}$$

where e_i and e_j are edges in G, and G has k edges.

To create the attributed line graph, we convert the bipartite graph G, as detailed in Definition 3.1, without attributes. Then, edge features in G become node features in G'. Node features in G become edge features in G' by extracting the shared node between the original edges in G and assigning the corresponding node feature. This transformation allows for the application of many GNNs as the edge weights in G' are now scalars when the node features in G are scalars. The ground truth assignments $\mathbf{Y}$ are not changed, other than being associated with the nodes in G', rather than the edges in G. Coupled with the expansion component, assignment scores for nodes in G' are trained directly through a balanced multi-task binary cross-entropy (BCE) loss function. The balancing parameter $\gamma = m-1$ serves as a positive class weight, due to there being only one *assignment* label per expanded agent and $m-1$ *no assignment* labels.

3.2.1 Speeding Up Inference via Pruning. Assignment problems, traditionally solved through supervised learning [2, 16, 39], can also be viewed as a graph pruning task, where the remaining edges represent assignments. While pruning is commonly applied for compression [46] and sparsification [37, 40], its potential in assignment remains under explored. Thus, we employ an inference-time pruning strategy that prunes the graph during MAGNET's forward pass to minimize the elements aggregated during message passing, unlike traditional pruning, which focuses on reducing model weights. We perform pruning by scoring edges according to their respective activation output after the encoding step (ENC), and remove low scoring edges. As node-centric learning paradigms do not have edge activation, i.e. no edge scores, we focus on pruning MAGNET's edge-centric GNN. The percent of the total edges per agent for a layer l after pruning is expressed as $\lceil \frac{m\rho}{l+1} \rceil$, where m is the number of tasks, and $\rho \in (0,1)$ is the pruning factor. When ρ is small, few edges are kept and the pruning is stronger, and when ρ is large, more edges are kept and the pruning is weaker. To maintain feasibility, edges are dropped on a per-agent basis to ensure agents are never disconnected. We also include an L1 regularization of the activation scores to encourage sparsity. While this is similar to channel-wise L1 regularization to remove weights [54], coupling this technique to a GNN architecture offers a unique capability to edit the underlying graph. The final loss function of MAGNET becomes the sum of the balanced BCE loss and L1 regularization, with the regularization weighted by λ.

3.2.2 Weighted Bipartite Edge Vectors. In the case of edges possess feature vectors, such as in the weighted bipartite representation, we employ two convolution modules per layer: one for probability values and one for cost values. After each layer, we concatenate the resulting aggregated representations, weighted by their respective property, creating new node representations. Thus, for a single node v, where P is the matrix of interaction probabilities and C is the matrix of interaction costs:

$$\begin{aligned} r_{v,p}^{l+1} &= ENC(r_v^l, AGGR(r_u^l, P_{v,u} : u \in N(v))), \\ r_{v,c}^{l+1} &= ENC(r_v^l, AGGR(r_u^l, C_{v,u} : u \in N(v))). \end{aligned} \tag{3}$$

The final representation for the node v is below, where $||$ denotes concatenation and f is an embedding function, represented as an MLP:

$$r_v^{l+1} = f(r_{v,p}^{l+1} || r_{v,c}^{l+1}). \tag{4}$$

3.3 Ensuring Feasibility

Previous works have treated feasibility as a soft constraint [39]. In contrast, MAGNET guarantees feasible solutions using a greedy algorithm based on assignment scores. To accomplish this, we utilize MAGNET's first component by predicting an edge scoring matrix for each duplicate of agent a_i with ϕ_i duplicates. An argmax operation identifies the highest-scoring tasks for each duplicated agent, forming the candidate assignment set. Motivated by the greedy assignment process for single-agent knapsack [14], we generate the final assignment set by sorting the candidate set based on their predicted scores and greedily adding assignments to the solution set from the sorted candidate set. If an assignment for a_i causes the total cost to exceed α_i, the assignment is discarded. This is done until all ϕ_i possible candidates are checked. An example of this process can be found in the third step of Figure 1b. For problems without costs and/or probabilities, attaining feasibility can be seen as special cases of this component. In the *no cost case*, all costs equal one, making assignment dependent solely on capacity, allowing direct use of the candidate assignment set. In the *no probabilities case*, we can assume that all connections succeed, and thus agents do not need to be assigned to the same task multiple times. In this scenario, the expansion component is turned off, and the same greedy algorithm can be applied over each agent's individual task scoring vector. Additionally, pruning does not impact feasibility given edges are dropped on a per-agent basis.

4 EXPERIMENTS

This section presents experiments demonstrating the efficacy of MAGNET (code is available at github.com/dloveland/MAGNET). Through our empirical analysis, we aim to answer the following research questions (RQ):

- **RQ1**: How **effective** is MAGNET at solving the complex assignment problem formulated in Equation (1) compared to previous baselines?
- **RQ2**: How does MAGNET's performance **generalize** to problems of varying size?
- **RQ3**: How does MAGNET's execution time **scale** across varying problem sizes?

4.1 (RQ1) How effective is MAGNET?

This RQ aims to study how effective MAGNET is at solving the assignment problem specified in Equation (1), compared to previous GNN baselines and a state-of-the-art traditional solver. Specifically, we seek to understand how close the predicted assignments are to the optimal assignments, and how quickly MAGNET can predict such assignments.

Data. Benchmark datasets for assignment problems are limited, especially as complexity increases. Following past work [16, 25, 39, 48],

Table 1: Optimality gaps and execution times on the continuous and discrete cost probabilistic assignment tasks. Blue is used to indicate best metric. Across all models, MAGNET produces the highest quality assignments for both tasks, as seen by lower optimality gaps. MAGNET is also several orders of magnitude faster than MOSEK, while being competitive with other GNN approaches.

Model	Probabilistic and Continuous Cost Data		Probabilistic and Discrete Cost Data	
	Optimality Gap (↓)	Execution Time (s) (↓)	Optimality Gap (↓)	Execution Time (s)(↓)
MOSEK [3]	0.00±0.00	2.0710±0.2589	0.00±0.00	0.5642±0.0870
Greedy [28]	0.27±0.09	0.0003±0.00012	0.35±0.09	0.00035±0.00015
DOT-GCN	0.90±0.03	0.0119±0.0017	0.90±0.02	0.0070±0.0002
DOT-GCNII	0.89±0.05	0.0128±0.0011	0.86±0.01	0.0081±0.0002
DOT-FAGCN	0.84±0.06	0.0154±0.001	0.81±0.07	0.0070±0.0001
DOT-GPRGNN	0.89±0.06	0.0162±0.0096	0.90±0.02	0.0078±0.0001
CONCAT-GCN [2]	0.32±0.08	0.0400±0.0139	0.40±0.08	0.0085±0.0006
CONCAT-GCNII	0.31±0.09	0.0519±0.0371	0.40±0.06	0.0088±0.0006
CONCAT-FAGCN	0.32±0.09	0.0333±0.0163	0.38±0.09	0.0086±0.0009
CONCAT-GPRGNN	0.33±0.12	0.0348±0.0091	0.36±0.07	0.0090±0.0012
GLAN [39]	0.45±0.12	0.0398±0.0627	0.72±0.06	0.0351±0.0404
MAGNET-GCN	0.74±0.10	0.0662±0.0301	0.87±0.03	0.0161±0.0022
MAGNET-GCNII	0.29±0.08	0.0431±0.0155	0.31±0.06	0.0134±0.0006
MAGNET-FAGCN	0.20±0.07	0.0409±0.0105	0.27±0.06	0.0171±0.0027
MAGNET-GPRGNN	0.19±0.07	0.0445±0.0174	0.18±0.08	0.0137±0.0007

we generate synthetic data based on Equation (1). We begin by generating 100 *probabilistic assignment problems with continuous costs and capacities*. For each training instance includes 10 agents and 10 tasks, with capacities sampled from $U(1.0, 3.0)$, rewards from $U(1.0,10.0)$, probabilities from $U(0.1,0.9)$, and costs from $U(0.5,1.0)$, where $U(a,b)$ is the uniform distribution between a and b. Capacities and costs are set to allow no more than 6 assignments per agent. All instances of Equation (1) for our experiments were set up using the CVXPY modeling language [15], and solved using MOSEK. The true assignments are generated by MOSEK, a state-of-the-art commercial integer programming solver [3]. We use a 60\20\20 split for train\validation\test, respectively. To better understand the impact of continuous costs, we generate additional instances for *probabilistic assignment problems with discrete costs and capacities*, setting the costs to 1 and sampling the capacities from {1,2,3}, while maintaining the other parameters from the continuous case.

Models. We start by using MOSEK, one of the few solvers capable of exactly solving the exponential cone optimization defined by (1). We then design a set of baselines to understand how MAGNET's edge-centric learning paradigm improves scoring. We adapt MAGNET's first and third components for each of the baselines; thus, the core difference between MAGNET and the baselines is the scoring mechanism. We compare MAGNET with two node-centric methods: DOT, which uses a dot product of agent-task embeddings, and CONCAT, which employs an MLP transformation of concatenated embeddings. These methods, akin to recommendation/link prediction systems, combine node embeddings to evaluate potential links [52]. We also consider GLAN, a GNN specifically designed for assignment that utilizes an alternating learning style. First, GLAN applies an MLP to edge features and updated edge representation by an edge-based convolution operator, scaled by attention weights. Second, GLAN performs node updates by concatenating the edge representation to the node representation during aggregation. Since GLAN is trained using balanced BCE loss with soft constraints, it is unable to ensure feasibility. Since GLAN lacks an open-source implementation, we implemented it in PyTorch Geometric.

While GLAN specifies an architecture, MAGNET, and the node-embedding methods, are all model-agnostic. To understand their trade-offs, and to additionally study the impact of the GNN learning mechanism on edge scoring, we leverage four different GNN backbones: GCN [27], GCNII [12], FAGCN [7], and GPRGNN [13]. GCNII, FAGCN, and GPRGNN all build on the base GCN architecture, leveraging additional weighting strategies to improve performance and circumvent standard GCN issues, such as oversmoothing. Finally, we implement a greedy algorithm similar to that used for the Quiz Selection problem [28]. The original heuristic in [29] is based off of a strategy for maximum return on classical quiz problems, but focuses on problems with strictly integral costs and upper bounds in the constraints. Our heuristic uses the same value array described in [29]. To handle non-integer cost matrices $\mathbf{C}$ and capacities α_i we iterate over the rows of the value array, sort each row in descending order, and iteratively add the maximum number of assignments possible per task (in descending order) without violating the capacity constraints.

All models are implemented in PyTorch Geometric and tuned using the validation set to minimize the optimality gap. Hyperparameter tuning for all models included searching over the hidden dimensions ({16,32}), depth ({2,3}) and dropout ({0.3,0.5}). Additionally, we tune α ({0.1,0.5,0.9}) for GPRGNN, and set $K = 10$. For FAGNN, we tune ϵ ({0.1,0.5,0.9}). Models are trained with the Adam optimizer, tuning both the learning rates (0.001,0.0001,0.00001) and L1 regularization strength λ ({0.0,0.5}). For the experiments in this section, we do not use pruning, but employ it later to assess speedup capabilities. Training is conducted on an NVIDIA Volta V100 GPU.

Metrics. We assess all models based on *performance* and *execution time*. For performance, we calculate the optimality gap, which

measures the difference between the objective value of the optimal solution from MOSEK, O_{mosek}, and the objective value associated with the solution from the GNN architecture, O_{gnn}, defined as $|O_{mosek} - O_{gnn}|/O_{mosek}$. A smaller gap indicates that the GNN's solution is closer to optimal. The optimality gap for MOSEK is zero, serving as the benchmark. This metric accommodates non-unique solutions by focusing on objective values rather than specific edge choices. For execution time, we measure the inference duration for various GNN designs and compare it to MOSEK, including the time for line graph transformation and feasibility checks. We report both the average and standard deviation across test sets for each metric.

Results. In Table 1, we provide the optimality gaps and execution times. MAGNET outperforms the other GNN baselines, attaining a 12% optimality increase on the continuous cost dataset, and 18% on the discrete cost dataset with the GPRGNN backbone, relative to the next best GNN baseline (CONCAT-GCNII and CONCAT-GPRGNN, respectively). The greedy heuristic is competitive for the probabilistic and continuous cost datasets, however the handcrafted decision rules fail to generalize to both settings, as seen in the 8% and 17% drop in optimality gap relative to MAGNET over the continuous and discrete cases, respectively. Focusing on learning methods, these results indicate that the edge-centric paradigm of MAGNET provides a significant benefit in scoring edges, particularly in the continuous setting. MAGNET-GCN is a notable outliers compared to the rest of the MAGNET results, which we attribute to GCN's susceptibility to oversmoothing as the line graph transformation creates higher degree nodes. While the best performing variant of MAGNET attains a 19% optimality gap relative to MOSEK, we highlight that MAGNET is two orders of magnitude faster, offering a strong trade-off between execution time and performance. Compared to the node-centric baselines, MAGNET retains comparable execution times despite the added computation. This efficiency is attributed to node-centric baselines needing extra edge scoring operators and modules, increasing execution time.

Table 2: Ablation study for agent pre-processing step of MAGNET. The decrease in optimality gaps highlights the benefits of MAGNET's pre-processing component.

MAGNET Model w/o Comp. 1	Probabilistic and Continuous Cost Data	Probabilistic and Discrete Cost Data
	Opt. Gap Change (↓)	Opt. Gap Change (↓)
MAGNET-GCN	0.03±0.02	−0.09±0.03
MAGNET-GCNII	0.02±0.01	−0.03±0.01
MAGNET-FAGCN	−0.06±0.01	−0.05±0.02
MAGNET-GPRGNN	−0.08±0.01	−0.10±0.01

While we indirectly evaluate MAGNET's edge scoring component across different GNN baselines, we also conduct an ablation study on MAGNET's preprocessing component. Specifically, we bypass the expansion transformation and instead perform greedy assignment using logit scores from a single agent. We allow for the possibility of multiple assignments to a specific task by allowing high scoring candidates to be assigned multiple times, assuming that they do not exceed an agent's capacity. In Table 2, we report the change in the optimality gap metric from Table 1, computed as the original optimality gap - the optimality gap without preprocessing, for all datasets and MAGNET backbones. The standard deviation is over the differences. Negative values indicate a decrease in relative performance, while positive values indicate an increase in relative performance. The predominantly negative values indicate that preprocessing significantly improves MAGNET's performance, with the relative percent change approaching 50% for GPRGNN.

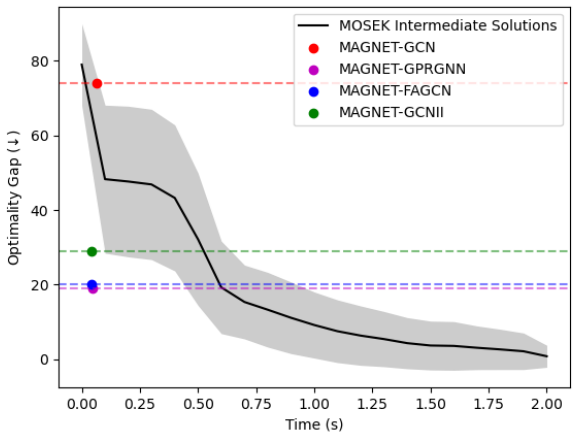

Figure 2: Optimality gap for MAGNET and intermediate solutions from MOSEK, where lower optimality gap is better. The gray region denotes the standard deviation in optimality gap across the test set. Dashed lines are used as a visual aid to demonstrate where MOSEK finds a comparable solution to each MAGNET variant. While MOSEK is capable of finding similar solutions to MAGNET faster than the allotted 2 seconds, MOSEK is still roughly an order of magnitude slower.

As a final performance study, we explore whether MAGNET can surpass MOSEK's intermediate solutions. MOSEK uses a branch-and-bound (B&B) algorithm for integer programming, iteratively refining lower and upper bounds on the optimal objective value. The lower bounds arise from relaxing integer constraints. During optimization, the best current feasible solution is called the "incumbent." If the gap between bounds becomes zero, this incumbent is confirmed as optimal. While MOSEK can ensure optimality, it may return suboptimal intermediate solutions before the B&B process concludes. Therefore, we compare MAGNET's performance against these intermediate solutions from MOSEK.

Using the same data as in RQ1, Figure 2 shows the average relative gap between MOSEK's objective value and its lower bound over 100 runs, measured across a 2-second time frame. The gray shaded region indicates the standard deviation of the relative gap. We compare these intermediate solutions with those from various MAGNET variants. Notably, while MOSEK achieves low relative gaps in about one-third of the time needed for the guaranteed optimal solution, it is still significantly slower than MAGNET in achieving similar solution quality. This result demonstrates that MOSEK's carefully designed

algorithm often still requires a significant search process to acquire a reasonable solution, rather than quickly identifying a strong solution and refining it. Consequently, even if one were to use MOSEK's intermediate solutions without optimality guarantees, they would take considerably longer to compute compared to MAGNET's solutions.

4.2 (RQ2) How does MAGNET's performance generalize to problems of larger size?

RQ2 aims to understand how MAGNET (without pruning) can generalize to larger problem sizes. As the computational overhead of larger graphs can make training infeasible, we seek to demonstrate that MAGNET can be trained on smaller training instances and generalize to larger assignment problems with minimal degradation in performance. The generalization results can be seen in Figure 3.

Data. We follow a similar data generation strategy as in Section 4.1; however, we now generate optimization problems of different sizes. We generate test problems with agent and task set sizes between 10 to 15, 20 to 25, and 50 to 55. Notably, these sizes are comparable to recent real world applications, such as the 6 agents and 12 tasks in [57], and 3 agents and 25 tasks in [24]. As solving assignment problems through MOSEK with 15 agents and tasks can already require significant time to achieve the optimal solution (30+ minutes), we set a time limit of 300 seconds and use the *incumbent* solution, i.e. the best solution found within the 300 seconds, to compare against. All the models are trained using the same data generated in Section 4.1 (10 agents, 10 tasks). Then, for testing, we generate 40 problem instances for each of the larger sizes. In these sets, 50% of the instances are used for validation, and 50% are used for testing. The capacities, rewards, probabilities, and costs are sampled from the same distributions as RQ1.

Models. We focus on MAGNET's generalization using the same four GNN backbones as in Section 4.1. While MAGNET is not directly compared to MOSEK, the metrics are scaled by the incumbent solution after 300 seconds of optimization, thus providing comparison.

Metrics. We demonstrate generalization by measuring how the optimality gap changes with respect to different problem sizes. For performance, we compute the same optimality gap metric as in Section 4.1, however we now use the the incumbent solution found by MOSEK within 300 seconds.

Results. In Figure 3, we compare the optimality gaps between MOSEK's incumbent solutions and MAGNET's predictions using different GNN backbones. Consistent with Table 1, the GCNII, FAGCN, and GPRGNN backbones exhibit lower optimality gaps compared to GCN. Notably, as problem size increases, these MAGNET variants maintain similar performance, with GPRGNN and GCNII showing flat performance curves. Despite the vast size difference between training and test problems, MAGNET's consistent performance indicates robust, size-agnostic learning capabilities. This suggests that MAGNET can effectively reduce both inference and training times by requiring only small training examples.

4.3 (RQ3) How does the execution time of MAGNET scale across larger problem sizes?

The execution time results are shown in Figure 4, and contextualized with performance in Figure 5, elucidating the trade-off between speed and solution quality.

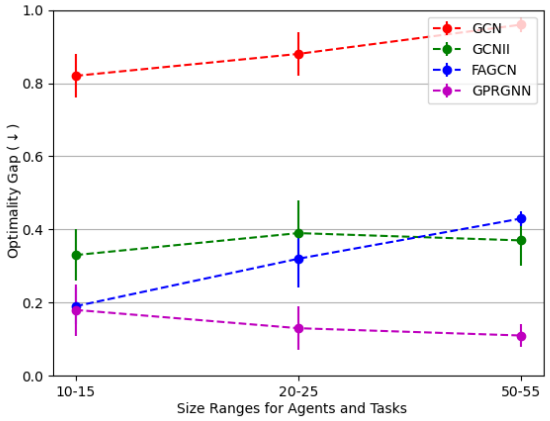

Figure 3: Optimality gap for MAGNET across different problem sizes, lower is better. Each model is trained on problems with 10 agents and tasks, then applied to the larger problems. Results demonstrate MAGNET generalizes well across problem sizes.

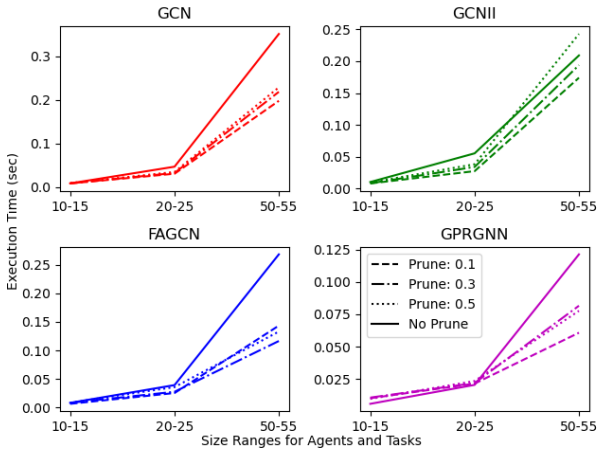

Figure 4: Execution time for MAGNET across problem sizes; lower is better. Each model is trained on problems with 10 agents and tasks, then applied to larger problems. All MAGNET variants achieve accelerated execution time, up to 53%.

Data. We use the same data as in Section 4.2, to study how execution time varies across larger problems. Again, the models are trained on the smaller optimization problems from 4.1 to demonstrate how MAGNET's performance scales, when trained on smaller datasets.

Models. We focus MAGNET's execution time using the same four GNN backbones as in Section 4.2. For each GNN backbone, we create four different variants based on different pruning strengths. According to the pruning equation specified in Section 3.2.1, we set $\rho \in \{0.1, 0.3, 0.5\}$, where 0.1 denotes strong pruning, and 0.5 denotes weaker pruning. We additionally include a variant without pruning.

Metrics. To demonstrate the execution time scaling of MAGNET, we measure the time it takes to perform inference with each GNN

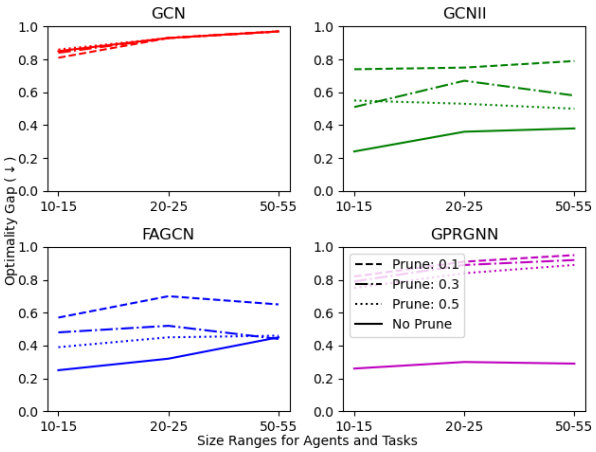

Figure 5: Optimality gap for MAGNET across problem sizes with different backbones and pruning rates, lower is better. Each model is trained on problems with 10 agents and tasks, then applied to larger problems. FAGCN and GCNII backbones retain low optimality gaps, especially as problem size grows.

backbone variant. Unlike before where the line graph transformation and feasibility step were included in the computation, we now only compute the time for edge scoring to isolate execution time changes due to pruning and GNN design. For performance, we compute the optimality gap between the incumbent solution computed by MOSEK within 300 seconds of optimization, similar to Section 4.2.

Results. In Figure 4, we provide execution time analysis for MAGNET with the different GNN backbones. For all models, pruning is able to reduce execution time, with FAGCN achieving a 53% improvement. Generally, the strength of the pruning strategy plays a less significant role due to earlier GNN layers retaining similar edge set sizes. Comparing execution time to performance, we see that the change in optimality gap does not significantly change as problem sizes increase. Thus, we can expect our pruning strategy to allow MAGNET to maintain efficiency as the assignment problems become larger. Notably, GNN backbones exhibit different behaviors post-pruning. For example, GCNII experiences higher execution times than FAGCN and GPRGNN, while GPRGNN suffers more performance degradation than GCNII and FAGCN. We hypothesize that the designs of the GNN backbones can impact these metrics – GCNII's identity mapping operation may dominate the execution time metric and GPRGNN's predict-then-propagate paradigm may cause an inability to recover when high-quality assignments are pruned. Designing GNNs that are amenable to this form of pruning may further increase acceleration.

5 RELATED WORK

Traditional Assignment Problems. Assignment problems have been studied for decades [47], with the Hungarian method being one of the earliest attempts to solve such problems [31]. The original assignment problem studied the single assignment case, including variants such as the assignment problem with agent qualification [10], the balanced assignment problem [41], and the quadratic assignment problem [33]. The generalized assignment problem (GAP) [11] extends the assignment problem to allow agents to be assigned to multiple tasks. This formulation was also expanded into several variations, including the multiple resource GAP [34], and the quadratic generalized assignment problem [21]. Assignment problems in general are NP-complete or NP-hard combinatorial problems, motivating a number of sampling-based heuristic methods to solving this problem inexactly [30]. Auction algorithms have also been proposed to solve assignment problems by allowing agents to bid on completing certain tasks; however, this greedy process generally does not result in optimal assignments [5, 55]. Integer programming solvers have also been traditionally used to solve various types of assignment problems [1, 9, 56]. Unlike heuristic methods, integer programming solvers have guarantees on finding optimal solutions. However, runtimes scale exponentially with the number of integer variables due to the NP-hard nature of these problems.

Deep Learning for Assignment Problems. Given the NP-complete or NP-hard nature of assignment problems, machine learning has been leveraged to predict assignment solutions. Neural network solvers have proven successful [38, 43, 51] across many scenarios [17, 36, 45]. Recently, GLAN explored GNN-based solutions, specifically for classic non-probabilistic single-task assignments with a cost matrix [39]. GLAN models the problem using a weighted bipartite graph, where edge weights represent agent-to-target assignment costs. Its learning approach alternates between node-based and edge-based convolution operations, combining edge embeddings with those of connected nodes, scaled by attention weights. GLAN offers faster performance than previous combinatorial methods while maintaining accuracy by iteratively updating edge weights during inference. However, GLAN is limited in handling more complex assignments. Additionally, [2] examines GNNs for bipartite assignment using a node-centric GCN with a CONCAT function, while [45] applies GNNs to the quadratic assignment problem.

6 CONCLUSION

This work introduces MAGNET, an efficient framework for solving multi-agent, multi-task probabilistic assignment problems. MAGNET consists of three key components: (1) a pre-processor that expands the bipartite graph for multi-task assignments, (2) an edge-centric GNN that produces high-quality solutions, and (3) a post-processor that ensures feasibility. To mitigate the computational cost of the line graph transformation, we enhance the second component with an inference-time pruning strategy to improve execution speed. Our comprehensive experiments show that MAGNET (a) generates high-quality assignments, (b) generalizes to larger problems, and (c) reduces inference time by two orders of magnitude compared to MOSEK. Overall, MAGNET enhances AI capabilities by efficiently solving a new class of assignment-based optimization problems.

ACKNOWLEDGMENTS

We thank the reviewers for their constructive feedback. This material in this work is partially supported by the National Science Foundation under a Graduate fellowship, IIS 2212143 and CAREER Grant No. IIS 1845491. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.

REFERENCES

[1] Jeph Abara. 1989. Applying integer linear programming to the fleet assignment problem. *Interfaces* 19, 4 (1989), 20–28.

[2] Carlo Aironi, Samuele Cornell, and Stefano Squartini. 2022. Tackling the Linear Sum Assignment Problem with Graph Neural Networks. In *International Conference on Applied Intelligence and Informatics*. Springer, 90–101.

[3] MOSEK ApS. 2023. *The MOSEK optimization toolbox for Python. V 10.1.*

[4] Sambaran Bandyopadhyay, Anirban Biswas, M Narasimha Murty, and Ramasuri Narayanam. 2019. Beyond node embedding: A direct unsupervised edge representation framework for homogeneous networks. *arXiv preprint arXiv:1912.05140* (2019).

[5] Dimitri P. Bertsekas. 1979. The auction algorithm: A distributed relaxation method for the assignment problem. In *Annuls of Operations Research*.

[6] Piotr Bielak and Tomasz Jan Kajdanowicz. 2022. PyTorch-Geometric Edge - a Library for Learning Representations of Graph Edges. In *Learning on Graphs Conference*.

[7] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond Low-frequency Information in Graph Convolutional Networks. In *AAAI*.

[8] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[9] Jon A Breslaw. 1976. A linear programming solution to the faculty assignment problem. *Socio-Economic Planning Sciences* 10, 6 (1976).

[10] Gaetan Caron, Pierri Hansen, and Brigitte Jaumard. 1999. The assignment problem with seniority and job priority constraints. *Operations Research* 47, 3 (1999), 449–453.

[11] Dirk G Cattrysse and Luk N Van Wassenhove. 1992. A survey of algorithms for the generalized assignment problem. *European journal of operational research* 60, 3 (1992), 260–272.

[12] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*.

[13] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations*.

[14] George B. Dantzig. 1957. Discrete-Variable Extremum Problems. *Operations Research* 5, 2 (1957), 266–277.

[15] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research* (2016).

[16] Danièle Gibbons, Cheng-Chew Lim, and Peng Shi. 2019. Deep Learning for Bipartite Assignment Problems*. *IEEE International Conference on Systems, Man and Cybernetics* (2019), 2318–2325.

[17] Daniel Gibbons, Cheng-Chew Lim, and Peng Shi. 2019. Deep learning for bipartite assignment problems. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE.

[18] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1263–1272.

[19] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA.

[20] Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual.

[21] Peter M Hahn, Bum-Jin Kim, Monique Guignard, J MacGregor Smith, and Yi-Rong Zhu. 2008. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications* 40 (2008), 351–372.

[22] Yichao Jin, Jiong Jin, Alexander Gluhak, Klaus Moessner, and Marimuthu Palaniswami. 2011. An intelligent task allocation scheme for multihop wireless networks. *IEEE Transactions on Parallel and Distributed Systems* 23, 3 (2011), 444–451.

[23] Jaehyeong Jo, Jinheon Baek, Seul Lee, Dongki Kim, Minki Kang, and Sung Ju Hwang. 2021. Edge representation learning with hypergraphs. *Advances in Neural Information Processing Systems* 34 (2021), 7534–7546.

[24] Kelin Jose and Dilip Kumar Pratihar. 2016. Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods. *Robotics and Autonomous Systems* 80 (2016), 34–42.

[25] Alla Kammerdiner, Alexander Semenov, and Eduardo L Pasiliao. 2022. Multi-dimensional Assignment Problem for multipartite entity resolution. *Journal of Global Optimization* 84, 2 (2022), 491–523.

[26] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural Relational Inference for Interacting Systems. In *International Conference on Machine Learning*. PMLR, 2688–2697.

[27] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* (2017).

[28] Alexander Kline, Darryl Ahner, and Raymond Hill. 2019. The Weapon-Target Assignment Problem. *Computers & Operations Research* (2019).

[29] Alexander G Kline, Darryl K Ahner, and Brian J Lunday. 2019. Real-time heuristic algorithms for the static weapon target assignment problem. *Journal of Heuristics* 25 (2019), 377–397.

[30] Alexander G Kline, Darryl K Ahner, and Brian J Lunday. 2020. A heuristic and metaheuristic approach to the static weapon target assignment problem. *Journal of Global Optimization* 78 (2020), 791–812.

[31] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[32] Michail G Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J Kleywegt, Sven Koenig, Craig A Tovey, Adam Meyerson, and Sonal Jain. 2005. Auction-Based Multi-Robot Routing.. In *Robotics: Science and Systems*, Vol. 5. Rome, Italy, 343–350.

[33] Eugene L Lawler. 1963. The quadratic assignment problem. *Management science* 9, 4 (1963), 586–599.

[34] Larry J LeBlanc, Avraham Shtub, and G Anandalingam. 1999. Formulating and solving production planning problems. *European Journal of Operational Research* 112, 1 (1999), 54–80.

[35] Mengyuan Lee, Yuanhao Xiong, Guanding Yu, and Geoffrey Li. 2018. Deep Neural Networks for Linear Sum Assignment Problems. *IEEE Wireless Communications Letters* PP (06 2018), 1–1. https://doi.org/10.1109/LWC.2018.2843359

[36] Mengyuan Lee, Yuanhao Xiong, Guanding Yu, and Geoffrey Ye Li. 2018. Deep neural networks for linear sum assignment problems. *IEEE Wireless Communications Letters* 7, 6 (2018).

[37] Gaotang Li, Marlena Duda, Xiang Zhang, Danai Koutra, and Yujun Yan. 2023. Interpretable Sparsification of Brain Graphs: Better Practices and Effective Designs for Graph Neural Networks. *arXiv preprint arXiv:2306.14375* (2023).

[38] Tao Li and Luyuan Fang. 1991. Competition based neural networks for assignment problems. *Journal of Computer Science and Technology* 6, 4 (1991), 305–315.

[39] He Liu, Tao Wang, Congyan Lang, Songhe Feng, Yi Jin, and Yidong Li. 2022. Glan: A graph-based linear assignment network. *arXiv preprint arXiv:2201.02057* (2022).

[40] Donald Loveland and Rajmonda Caceres. 2023. Network Design through Graph Neural Networks: Identifying Challenges and Improving Performance. In *International Conference on Complex Networks and their Applications*.

[41] Silvano Martello, William R Pulleyblank, Paolo Toth, and Dominique De Werra. 1984. Balanced optimization problems. *Operations Research Letters* 3, 5 (1984).

[42] Nguyen Minh-Tuan and Yong-Hwa Kim. 2019. Bidirectional Long Short-Term Memory Neural Networks for Linear Sum Assignment Problems. *Applied Sciences* 9, 17 (2019).

[43] MAS Monfared and M Etemadi. 2006. The impact of energy function structure on solving generalized assignment problem using Hopfield neural network. *European journal of operational research* 168, 2 (2006), 645–654.

[44] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. 2018. Dual-primal graph convolutional networks. *arXiv preprint arXiv:1806.00770* (2018).

[45] Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. 2018. Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop*. IEEE, 1–5.

[46] Hongwu Peng, Deniz Gurevin, Shaoyi Huang, Tong Geng, Weiwen Jiang, Orner Khan, and Caiwen Ding. 2022. Towards sparsification of graph neural networks. In *2022 IEEE 40th International Conference on Computer Design*. IEEE, 272–279.

[47] David W Pentico. 2007. Assignment problems: A golden anniversary survey. *European Journal of Operational Research* 176, 2 (2007).

[48] Feng Qian, Kai Su, Xin Liang, and Kan Zhang. 2023. Task Assignment for UAV Swarm Saturation Attack: A Deep Reinforcement Learning Approach. *Electronics* 12, 6 (2023).

[49] Chayan Sarkar, Himadri Sekhar Paul, and Arindam Pal. 2018. A scalable multi-robot task allocation algorithm. In *ICRA*. IEEE.

[50] Regis Vincent, Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Benoit Morisset, Charles Ortiz, Dirk Schulz, and Benjamin Stewart. 2008. Distributed multirobot exploration, mapping, and task allocation. *Annals of Mathematics and Artificial Intelligence* 52 (2008), 229–255.

[51] Eitan Wacholder. 1989. A neural network-based optimization algorithm for the static weapon-target assignment problem. *ORSA Journal on computing* (1989).

[52] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[53] Yihong Xu, Aljosa Osep, Yutong Ban, Radu Horaud, Laura Leal-Taixé, and Xavier Alameda-Pineda. 2020. How to train your deep multi-object tracker. In *CVPR*.

[54] Chih-Kuan Yeh, Ian E.H. Yen, Hong-You Chen, Chun-Pei Yang, Shou-De Lin, and Pradeep Ravikumar. 2019. DEEP-TRIM: REVISITING L1 REGULARIZATION FOR CONNECTION PRUNING OF DEEP NETWORK.

[55] Michael M. Zavlanos, Leonid Spesivtsev, and George J. Pappas. 2008. A distributed auction algorithm for the assignment problem. In *2008 47th IEEE Conference on Decision and Control*.

[56] Huizhen Zhang, Cesar Beltran-Royo, and Liang Ma. 2013. Solving the quadratic assignment problem by means of general purpose mixed integer linear programming solvers. *Annals of Operations Research* 207 (2013).

[57] Wanqing Zhao, Qinggang Meng, and Paul WH Chung. 2015. A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. *IEEE transactions on cybernetics* 46, 4 (2015), 902–915.