Accelerating Multi-Agent Planning Using Graph Transformers with Bounded Suboptimality

Chenning Yu[*,1], Qingbiao Li[*,2], Sicun Gao[1], Amanda Prorok[2]

Abstract— **Conflict-Based Search is one of the most popular methods for multi-agent path finding. Though it is complete and optimal, it does not scale well. Recent works have been proposed to accelerate it by introducing various heuristics. However, whether these heuristics can apply to non-grid-based problem settings while maintaining their effectiveness remains an open question. In this work, we find that the answer is prone to be no. To this end, we propose a learning-based component, i.e., the Graph Transformer, as a heuristic function to accelerate the planning. The proposed method is provably complete and bounded-suboptimal with any desired factor. We conduct extensive experiments on two environments with dense graphs. Results show that the proposed Graph Transformer can be trained in problem instances with relatively few agents and generalizes well to a larger number of agents, while achieving better performance than state-of-the-art methods.**

I. INTRODUCTION

Multi-Agent Path Finding (MAPF) is central to many multi-agent problems. The solution to MAPF is to generate collision-free paths guiding agents from their start positions to designated goal positions. MAPF has practical applications in item retrieval in warehouses [1], mobility-on-demand services [2], surveillance [3] and search and rescue [4].

Conflict-Based Search (CBS) is one of the most popular planners for MAPF [5]. It is provably complete and optimal. However, solving MAPF optimally is NP-hard [6], [7]. Consequently, CBS suffers from scalability, as the search space grows exponentially with the number of agents. Bounded-suboptimal algorithms [8], [9] guarantee a solution that is no larger than a given constant factor over the optimal solution cost. Though these methods often run faster than CBS for grid-based MAPF instances, their effectiveness remains an open question for non-grid-based problem settings, wherein agents can move in an arbitrary continuous domain. In addition, since most of these heuristics rely heavily on collision checking for conflicts, their computational costs may become considerable when the graphs are dense.

Recently, learning-based methods have shown their potential in solving MAPF tasks efficiently [10]–[13], which offload the online computational burden into an offline learning procedure. Yet, we find nearly none of them address graph-based MAPF settings. Therefore, the main interest of this work is to explore whether and how a learning-based method could accelerate MAPF planners in non-grid-based settings, especially for dense graphs.

Contributions. We propose to use a Graph Transformer as a heuristic function to accelerate Conflict-Based Search (CBS) in a non-grid setting. Similar to previous works [12],

by introducing focal search to CBS, our framework guarantees both the completeness and bounded-suboptimality of the solution. Our contributions are as follows:

- We propose a novel architecture, i.e., the Graph Transformer, which leverages the underlying structure of the MAPF problem. The proposed architecture has several desired properties, e.g., dealing with an arbitrary number of agents, making it a natural fit for the MAPF problem. To our knowledge, our work is one of the first works to introduce a learning component to MAPF problems under *non-grid-based* problem settings.
- We design a novel training objective, i.e., Contrastive Loss, to learn a heuristic that ranks the search nodes. Unlike [12], our loss can be directly optimized without introducing an upper bound, which is suitable for deep learning.
- We demonstrate the generalizability of our model by training with relatively few agents and testing in unseen instances with larger agent numbers. Results show that our approach can accelerate CBS significantly while ECBS, using handcrafted heuristics [8], fails.

Related Work. Leading MAPF planners mainly include three types: optimal planners, bounded-suboptimal planners, and unbounded-suboptimal planners. Optimal planners include BCP solvers [14], [15], and Conflict-Based Search (CBS) [5], followed by its variants, e.g., CBSH [16] and CBSH2 [17]. Bounded-suboptimal planners are another line of work with better scalability while guaranteeing completeness and bounded-suboptimality. Representative works include EPEA* [18], A* with operator decomposition [19], M* [20], ECBS [8], and EECBS [9]. Last but not least, there are unbounded-suboptimal planners that aim to aggressively accelerate the planning. Examples include Prioritized Planning [21], ORCA [22], Push-and-Swap [23], and Parallel Push-and-Swap [24]. These works can find solutions fast, but do not guarantee the solution quality [25], [26].

Recently, learning-based methods were introduced to solve multi-agent tasks efficiently, using imitation learning [10], [27] and reinforcement learning [13], [28], [29]. These end-to-end methods are often good at memorizing the patterns that are seen during training, which could save significant online computation when deployed to similar tasks. However, it is hard to ensure their completeness. To this end, several works have been proposed to train a learning-based heuristic and use it to guide the tree search [30]–[33]. To accelerate CBS and ECBS, Huang et al. [11], [12] use Supported Vector Machines (SVM) to bias the search, and train it via imitation learning. Our work is not only one of the first works that apply ML to *non-grid-based* problem settings, but also one of the first works that apply *deep learning* to MAPF with completeness and bounded-optimality guarantees.

*Equal contribution.
[1] Computer Science and Engineering Department, University of California San Diego. `<chy010, sicung>@ucsd.edu`
[2] Department of Computer Science and Technology, University of Cambridge. `<ql295, asp45>@cam.ac.uk`
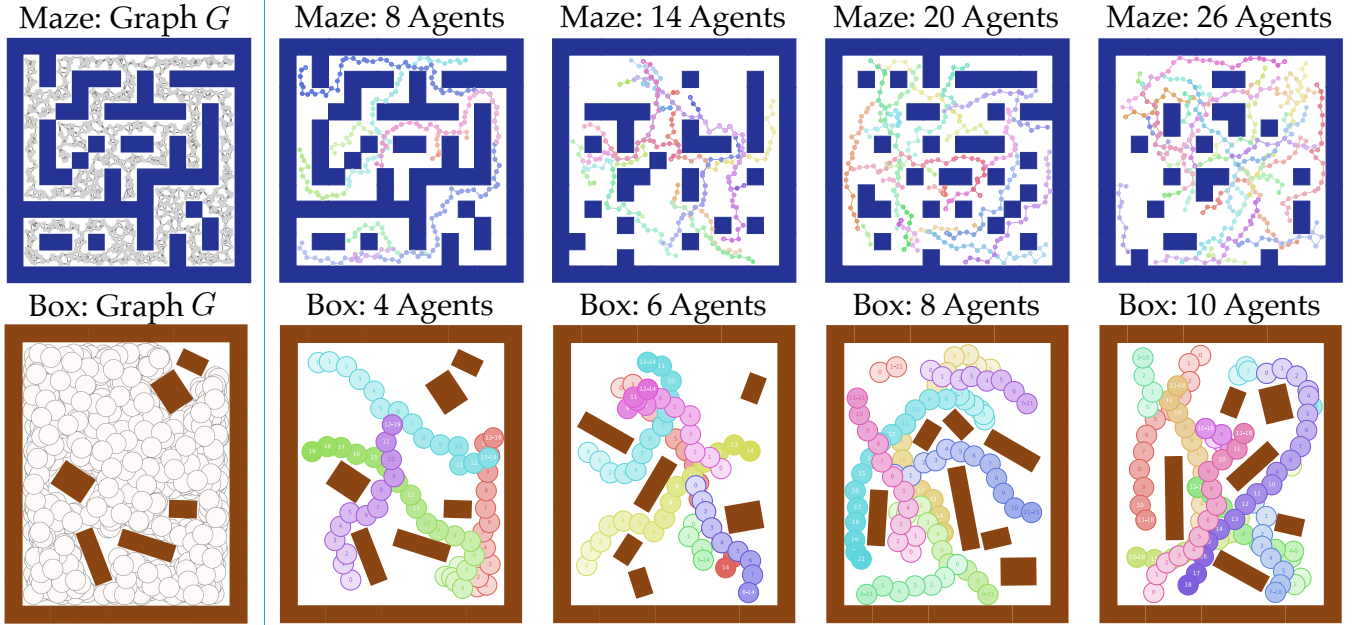
Fig. 1: **Left:** Examples of our graph-based MAPF instances. To construct the graph, we sample vertices randomly from the free space and connect them with collision-free edges. **Right:** Problem instances that our approach solves while other baselines fail. Different colors represent the trajectories of different agents. Vertices in the same trajectory have deeper colors if their respective time steps are later.

II. PROBLEM FORMULATION

We study Multi-Agent Path Finding (MAPF) in the 2D continuous space $\mathcal{C} \subseteq \mathbb{R}^2$. The configuration space $\mathcal{C}$ consists of a set of obstacles $\mathcal{C}_{obs} \subseteq \mathcal{C}$ and free space $\mathcal{C}_{free} : \mathcal{C} \setminus \mathcal{C}_{obs}$. Note that $\mathcal{C}_{obs}$ could be different from what appears in the workspace, since it also considers the geometric shape of the agent, which may not solely be a point mass.

A random geometric graph $G = \langle V, E \rangle$ is sampled from the space. Every sampled vertex $v \in V$ is collision-free, i.e., $v \in \mathcal{C}_{free}$. A directed edge $e \in E : (v_i \to v_j)$ connects v_i to v_j, if (i) v_j is one of the neighbors of v_i, and (ii) the edge is collision-free, i.e., $e \subseteq \mathcal{C}_{free}$. The neighbor set can be defined as the r-radius or k-nearest neighbors.

Suppose there are M agents on this graph G. Each agent i occupies a region $\mathcal{R}(q) \subseteq \mathcal{C}$, associated with a vertex $q \in V$. We assign a start vertex s_i and a goal vertex g_i to each agent i. We denote the path of agent i as $\sigma_i : \{v_i^t\}_{t \in [1 \cdots T_i]}$, where $T_i \in \mathbb{Z}_{>0}$, and agent i is on vertex v_i^t at time step t. We denote e_i^t as the edge $(v_i^t \to v_i^{t+1})$ that traverses from v_i^t to v_i^{t+1} in 1 timestep.

Problem Description. We consider a tuple $(G, \mathcal{S}, \mathcal{G}, \mathcal{C}, \mathcal{R})$ as a problem instance of MAPF, where $\mathcal{S} : \{s_i\}_{i \in [1 \cdots M]}$ and $\mathcal{G} : \{g_i\}_{i \in [1 \cdots M]}$ are the start and goal vertices. A conflict-free solution $\{\sigma_i\}_{i \in [1 \cdots M]}$, should satisfy the following objectives, given arbitrary time t and pair of agents i, j [34]:

(Endpoint) $v_i^0 = s_i \land v_i^{T_i} = g_i$

(Obstacle) $v_i^t \in \mathcal{C}_{free} \land e_i^t \subseteq \mathcal{C}_{free}$

(Inter-agent) $\mathcal{R}(q_i) \cap \mathcal{R}(q_j) = \emptyset, \forall q_i \in e_i^t, q_j \in e_j^t$

Note: If $t \geq T_i$, we assume the corresponding v_i^t is equal to $v_i^{T_i}$. This means that the agent will stay at the goal starting from time step $v_i^{T_i}$.

Solution Quality. We assume each edge requires 1 time step to traverse. The quality of the solution is measured by the sum of travel times (flowtime): $\sum_{i \in [1 \cdots M]} T_i$.

III. BACKGROUND: CONFLICT-BASED SEARCH WITH BIASED HEURISTICS

In this section, we first introduce Conflict-Based Search (CBS), an optimal multi-agent planner [5]. Then we introduce *focal search* [35], [36], which incorporates the biased heuristic into the CBS framework, while preserving the guarantees of bounded-suboptimality and completeness [8].

A. Conflict-Based Search

Conflict-Based Search (CBS) is an optimal bi-level tree search algorithm of MAPF. The high-level planner aims to solve inter-agent conflicts, while the low-level planner aims to generate optimal individual paths. Here we denote an inter-agent conflict as $(i, j, t, v_i^{t-1}, v_j^{t-1}, v_i^t, v_j^t)$, which implies two edges, $(v_i^{t-1} \to v_i^t)$ and $(v_j^{t-1} \to v_j^t)$, dissatisfy the inter-agent objective mentioned in Section II.

The high-level planner maintains a tree and decides which search node to expand in a best-first manner. To this end, each search node N stores the following information:

(1) A set of constraints $\mathcal{T}(N)$. A constraint (i, v, t) indicates that agent i should not traverse to graph vertex v at time t.

(2) A solution $\sigma(N): \{\sigma_i\}_{i \in [1 \cdots M]}$. The solution satisfies the endpoint and obstacle objective, but may or may not satisfy the inter-agent objective. In addition, the solution should obey the constraints $\mathcal{T}(N)$, i.e., $\forall i, \forall v_i^t \in \sigma_i, (i, v, t) \notin \mathcal{T}(N)$.

(3) The cost of the solution $c(N)$. The high-level planner prioritizes which search node to expand based on this metric.

On the high level, CBS first creates a root search node with no constraints, then keeps selecting a search node and expanding it. A search node N^* is selected if it is a leaf node with the lowest cost. CBS then checks whether the solution $\sigma(N^*)$ has an inter-agent conflict. If there is no conflict, $\sigma(N^*)$ will be returned as the final result. Otherwise, CBS

chooses the first conflict C: $(i, j, t, v_i^{t-1}, v_j^{t-1}, v_i^t, v_j^t)$, and splits it into two constraints C_1: (i, v_i^t, t) and C_2: (j, v_j^t, t). Two child search nodes N_1 and N_2 are then generated, with constraints $\forall i = 1, 2, \mathcal{T}(N_i) = \mathcal{T}(N^*) \cup \{C_i\}$ respectively. Then an optimal low-level planner, e.g., A* [37], is called by each child search node, which replans the path for each affected agent and records the respective solution and cost. CBS guarantees completeness and optimality, since both the high-level and low-level planners are performing best-first search [5].

B. Incorporating Biased Heuristics using Focal Search

CBS is an optimal planner, but it does not scale well even for grid-based problems settings. To improve the scalability, focal search [35], [36] was introduced by previous works, e.g., Bounded CBS (BCBS) and Enhanced CBS (ECBS) [8]. Here we describe a simplified version of BCBS.

We present the pseudocode of focal search in Algorithm 1. Focal search introduces the focal set to the CBS framework. A focal set ($Focal$) maintains a fraction of the leaf search nodes in the CBS tree (i.e., $Open$). We denote LB as the lowest solution cost in leaf search nodes. All the leaf search nodes that satisfy a near-optimal solution quality $c \le w \cdot LB$ will be added to $Focal$. This new CBS will select a search node from $Focal$ to expand, instead of that from $Open$. Compared to the original CBS, the new algorithm also performs the best-first search on $Focal$, but the search priority changes from the solution cost to a new heuristic function ψ. Typically, ψ is a handcrafted function that takes a solution as the input, and outputs a value that prefers solutions with fewer conflicts. We instead use a learned heuristic function based on the **Graph Transformer**.

Algorithm 1 CBS with Biased Heuristics [8], [12]

Input: A MAPF instance and suboptimality factor w
Input: Heuristic function ψ (e.g., Graph Transformer)
Generate the root search node R with an initial solution
Initialize open list $Open \leftarrow \{R\}$
$LB \leftarrow c(R)$, and initialize focal list $Focal \leftarrow \{R\}$
while $Open$ is not empty
   $N^* \leftarrow \arg\min_{N \in Focal} \psi(\sigma(N))$
   $C \leftarrow$ first conflict in $\sigma(N^*)$
   if C does not exist
      return solution $\sigma(N^*)$
   Remove N^* from $Open$ and $Focal$
   if $\min_{N \in Open} c(N) > LB$
      $LB = \min_{N \in Open} c(N)$
      $Focal = \{N \in Open : c(N) \le w \cdot LB\}$
   Generate two children nodes N_1 and N_2 from node N^*
   Add C_i to $\mathcal{T}(N_i)$, for $i = 1, 2$
   Call low-level planner to get $\sigma(N_i)$, for $i = 1, 2$
   Add N_i to $Open$, for $i = 1, 2$
   Add N_i to $Focal$ if $c(N_i) \le w \cdot LB$, for $i = 1, 2$
return No solution

Proposition. *Algorithm 1 is complete and bounded-suboptimal with a factor of $w \ge 1$, as mentioned in [8].*

Suppose the problem is feasible, but Algorithm 1 does not find a solution given a sufficient time budget. Then for an arbitrary search node N with a feasible solution, there exists an ancestor search node N^p added to $Open$ but not expanded. Suppose N^{p*} is the search node with the lowest cost among these unexpanded ancestors. N^{p*} is not selected by $Focal$, since it is not expanded. Thus, either (i) N^{p*} is not in $Focal$, or (ii) N^{p*} is in $Focal$ but not selected. (i) is impossible, because there do not exist infinitely many solutions that have costs lower than $\frac{1}{w} \cdot c(N^{p*})$. (ii) cannot happen, because there do not exist infinitely many solutions with costs lower than or equal to $w \cdot c(N^{p*})$. As a result, N^{p*} will be selected eventually and expanded. Therefore, we have proved the algorithm to be complete by contradiction. The focal search never expands search nodes with costs higher than w times the optimal solution; therefore, it is bounded-suboptimal with a factor of w.

We note that the focal search described here is a special case of BCBS [8], i.e., BCBS $(w, 1)$, as the focal search is only applied to the high-level planner. In our graph-based problem settings, there is no significant improvement when applying the focal search to the low-level planner. Rather, if we introduce the focal search to the low-level planner, it would consume a notable portion of computation on the collision checking of edges, which has no improvement in the overall performance. We refer readers to Question 4 in Section V-B for further details.

IV. GRAPH TRANSFORMERS AS HEURISTIC FUNCTIONS

In this section, we describe the architecture of the Graph Transformer and how to train it represent a heuristic function that accelerates CBS.
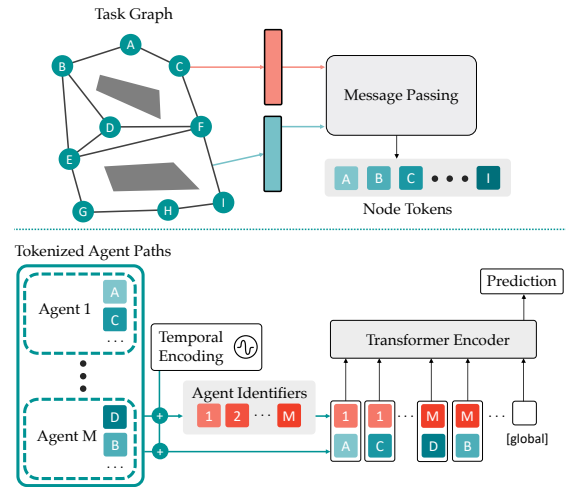

Fig. 2: The proposed Graph Transformer architecture. It has several desired properties that are specifically designed to deal with MAPF inputs. See Section IV-A for more details.

A. Network Architecture

The input to the graph transformer ϕ is a graph G, and a solution $\sigma = \{\sigma_i\}_{i \in [1 \cdots M]}$. The output $\phi(G, \sigma)$ predicts a scalar value as the heuristic. Such a predicted value correlates with the chance that the current search node will yield descendant search nodes with feasible solutions. For example, compared to those nodes that cannot eventually reduce the conflicts, the promising nodes leading to feasible solutions should have lower $\phi(G, \sigma)$ values.

The graph transformer has two stages: *graph tokenization* and *attentive aggregation*. We describe each stage as follows.

Stage 1: Graph Tokenization. The graph tokenization transforms each graph vertex into an embedding using a Graph Neural Network (GNN). Here we use Message-Passing Neural Networks [38] (MPNN) as the GNN architecture. The input to the MPNN is a graph $G = \langle V, E \rangle$, where the feature d_i^v for each graph vertex $v_i \in V$ is its respective 2D position, and the feature d_l^e for each edge $e_l = (v_i \to v_j)$ is the relative position of v_j to v_i. With two linear layers f_x and f_y, the vertices and edges are first encoded as x and y using $\forall v_i \in V, x_i = f_x(d_i^v); \forall e_l \in E, y_l = f_y(d_l^e)$. Then, using three MLPs $\{f_k\}_{k \in [1,2,3]}$, the MPNN updates the information for each graph vertex $v_i \in V$ as follows:

$$x_i \leftarrow x_i + \max\{f_k(x_i, x_j, y_l)\}, \forall e_l : (v_i \to v_j) \in E. \quad (1)$$

After all x_i are updated using MLP f_1, the MPNN continues to update x_i using MLP f_2 and so on. The max denotes the max-pooling over the feature dimension. We use max-pooling to take a set with an arbitrary number of elements while ensuring robustness [39]. Since it is invariant to the permutation of these elements, the MPNN here can take graphs with an arbitrary number of vertices and edges, but also is permutation invariant by construction.

Stage 2: Attentive Aggregation. After we compute the token x_i for each graph vertex v_i from Stage 1, we model the inter-agent interactions using the Transformer [40]. The path of each agent σ_i is first tokenized as $\rho_i = \{x_j, \forall v_j \in \sigma_i\}$. To inject the temporal information of these tokenized solutions, we introduce Temporal Encoding [31]. The approach is similar to [40], [41] (as positional encoding in their settings). We denote $\rho_i^t \in \mathbb{R}^D$ as the vertex token of agent i at time step t. For each token ρ_i^t, we add it element-wisely with a temporal encoding $\rho_i^t \leftarrow \rho_i^t + TE(t) \in \mathbb{R}^D$. The $2k$-th and $2k+1$-th dimensions of $TE(t)$ are as follows:

$$TE(t)_{2k} = \sin(t/10000^{2k/D}), \quad (2)$$
$$TE(t)_{2k+1} = \cos(t/10000^{2k/D}). \quad (3)$$

The hyperparameter 10000 is used following the common practice [40]. To encourage the model to be aware of which agent each token ρ_i^t belongs to, we concatenate the Agent Identifier τ_i to each token $\rho_i^t \leftarrow \rho_i^t || \tau_i$, similar to [42]. For each agent i, the agent identifier τ_i is calculated by taking the max-pooling over all its vertex tokens: $\tau_i = \max\{\rho_i^t\}, \forall t \in [1 \cdots T_i]$. Then, all tokens from the solution of all agents $\{\rho_i^t : \forall i \in [1 \cdots M], \forall t \in [1 \cdots T_i]\}$ will be fed as the input to the Transformer Encoder. For global prediction, we append an extra trainable token $global$ to the input, following the common practice [43], [44]. The Transformer Encoder predicts an output for each input token, and we use the output of the trainable $global$ token as ζ. With a linear layer $f_\phi : \mathbb{R}^D \to \mathbb{R}$, the final output is computed as $\phi(G, \sigma) = f_\phi(\zeta)$.

Here we use the Transformer Encoder, since it enables the Graph Transformer to take a variable number of tokens and model their dependencies, while preserving the invariance to the permutation of tokens. We refer readers to [40] for more details on the Transformer Encoder.

Properties of the Graph Transformer. By construction, the Graph Transformer is able to handle the input graph with a variable number of vertices and edges, agents with a variable total number, and the input solution with a variable

length. Additionally, it is aware of the temporal information, and inter-agent interactions. Its output is permutation invariant to both the orders of graph vertices and the agents.
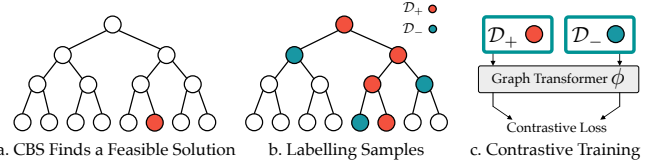
B. Training Graph Transformers

Fig. 3: The training framework. We use a supervised Contrastive Loss. The labels are generated from the CBS search tree.

Data Generation. Given a MAPF instance, we first use CBS to generate feasible solutions. No data will be collected if CBS fails to solve the instance. If it succeeds, we start to collect positive and negative samples from its search tree. A positive sample will be collected by dataset $\mathcal{D}_+$, if itself or one of its descendant search nodes contains the feasible solution. A negative sample will be collected by dataset $\mathcal{D}_-$, if it is a sibling search node of a positive sample. We record each sample's graph G and solution σ. In addition, we record the value d as its respective depth in the search tree.

Supervised Contrastive Learning. The objective of the model is to learn a ranking of the samples. Namely, given an arbitrary pair of positive sample (G, σ_+, d_+) and a negative sample (G, σ_-, d_-) from the same MAPF graph G, we learn the following ranking:

$$\phi(G, \sigma_+) < \phi(G, \sigma_-), \text{ if } d_+ \geq d_-.$$

We illustrate the intuition here. Imagine such ranking is learned perfectly and $w = \infty$, meaning all the leaf search nodes will be in $Focal$. Suppose at some time point, $Focal$ includes 1 positive sample p_+. $Focal$ may or may not include negative samples. If they exist, then their depths are no deeper than $d(p_+)$. Define this condition as a loop invariant. Then p_+ will be selected and expanded first according to the ranking. If p_+ is the feasible solution, then the algorithm terminates. Otherwise, $Focal$ will have one positive sample p'_+ and some negative samples, and all negative samples have depths no deeper than $d(p'_+)$. Thus, the loop invariant remains true. The loop invariant is also true for the base case, where $Focal$ only has the root search node. Therefore, by learning such ranking, we encourage Algorithm 1 to expand positive samples first and expand the negative samples as few as possible, which could save significant computation and greatly accelerate CBS.

We use supervised contrastive learning to train a Graph Transformer ϕ that ranks the positive samples above the negative samples. Given an arbitrary pair of positive and negative samples, $p_+ : (G_+, \sigma_+, d_+) \in \mathcal{D}_+, p_- : (G_-, \sigma_-, d_-) \in \mathcal{D}_-$, this pair is defined to be valid as: $\mathbb{I}(p_+, p_-) : (G_+ = G_-) \wedge (d_+ \geq d_-)$. With a hyperparameter $\gamma = 0.1$, we define $\delta(x) : \max(0, \gamma + x)$. We aim to minimize the Contrastive Loss as follows:

$$\frac{1}{L} \sum_{\substack{p_+ \in \mathcal{D}_+ \\ p_- \in \mathcal{D}_-}} \delta(\phi(G_+, \sigma_+) - \phi(G_-, \sigma_-)) \cdot \mathbb{I}(p_+, p_-), \quad (4)$$
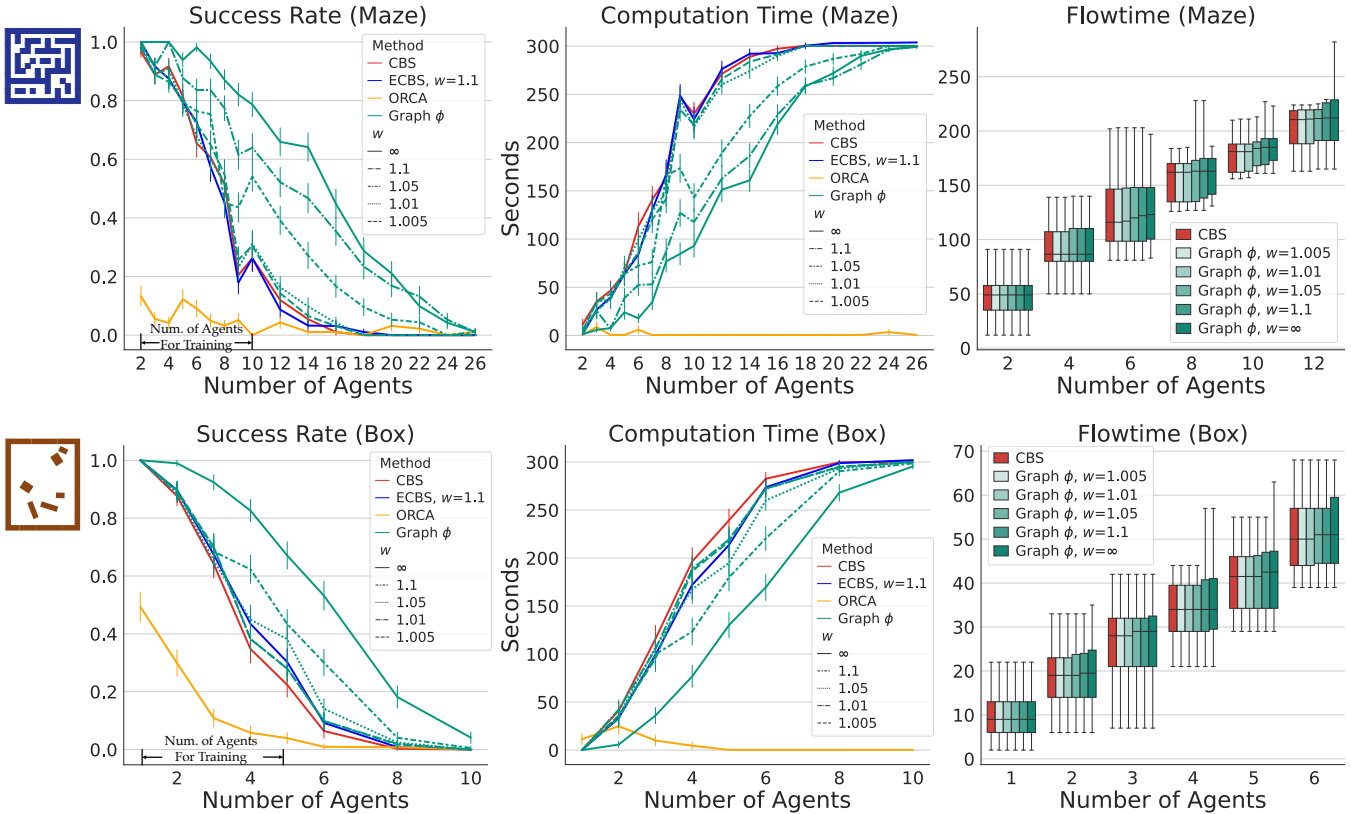
Fig. 4: Success rates, computation time, and flowtime within the runtime limit of 5 minutes, as functions of the number of agents. The results are averaged over 100 test instances for each setting of the agent number. We evaluate our approach with $w \in [1.005, 1.01, 1.05, 1.1, \infty]$, and compare its performance with CBS, ECBS ($w = 1.1$) and ORCA. Though trained with relatively few agents, results have shown that our approach generalizes well and significantly outperforms the baselines.

where $L = \sum_{p_- \in \mathcal{D}_-}^{p_+ \in \mathcal{D}_+} \mathbb{I}(p_+, p_-)$ is the total number of valid sample pairs, and $\mathcal{D}^+, \mathcal{D}^-$ are the datasets of positive and negatives samples respectively.

Once the training of Graph Transformer ϕ reaches convergence, we could deploy it as the heuristic function ψ in Algorithm 1. However, in practice, we found that the model would predict relatively low values for multiple search nodes in the focal set, indicating that all of them may lead to feasible solutions. To break the tie, we instead represent ψ as the depth d combined with ϕ, i.e., $\psi = \langle -d, \phi \rangle$. It means that the algorithm would first prefer the search nodes with deeper depths; if there exist multiple search nodes with the deepest depths, then it would prefer the search nodes with lower ϕ. Such a design enables the algorithm to make decisions consistently if multiple promising nodes exist. Without further specification, we denote our approach with such a heuristic as **Graph ϕ**.

V. EXPERIMENTS

A. Main Experiments

Experimental Setup. We design two types of environments for evaluation: Maze and Box (see Fig. 1). Each environment includes 2700 MAPF instances with samples generated by CBS for training. For testing, there are 100 MAPF test instances w.r.t. each agent number setting. We generate a random map for each Maze instance and a set of random obstacles for each Box instance. The graph and start and goal vertices are also generated randomly for each instance.

The training instances vary between 2 and 10 agents for Maze, and vary between 1 and 5 agents for Box. The test instances vary between 2 and 26 agents (2-10, 12, 14, 16, 18, 20, 22, 24, 26) for Maze, and vary between 1 and 10 agents (1-5, 6, 8, 10) for Box. We ensure that the test instances are unseen in the training set. All experiments were conducted using a 12-core, 3.2Ghz i7-8700 CPU and 4 Nvidia GTX 1080Ti GPUs. We test $w \in [1.005, 1.01, 1.05, 1.1, \infty]$ for our method. For all methods, we set the runtime limit as 5 minutes, following the common practice [12].

Baselines. We compare our method with 3 baselines: (i) Conflict-Based Search. (ii) Enhanced Conflict-Based Search (ECBS): a bounded-suboptimal version of CBS [8]. It applies focal search to both high-level and low-level planners, using hand-crafted heuristic functions. We choose its $w = 1.1$. (iii) ORCA [22]: a reactive collision avoidance algorithm, which works effectively in low density environments.

We find that there are very few open-source implementations that could directly apply CBS and ECBS to non-grid-based problems. As a result, we implement all tree-search methods (CBS, ECBS, and our method) from scratch using Python. We use basically the same framework when implementing CBS, ECBS, and our approach. To make the comparison fair, we ensure that all these 3 methods are aggressively optimized by strictly following the C++ implementations [1] and original paper [5], [8], and using Bayesian hyperparameter search for the w of ECBS.
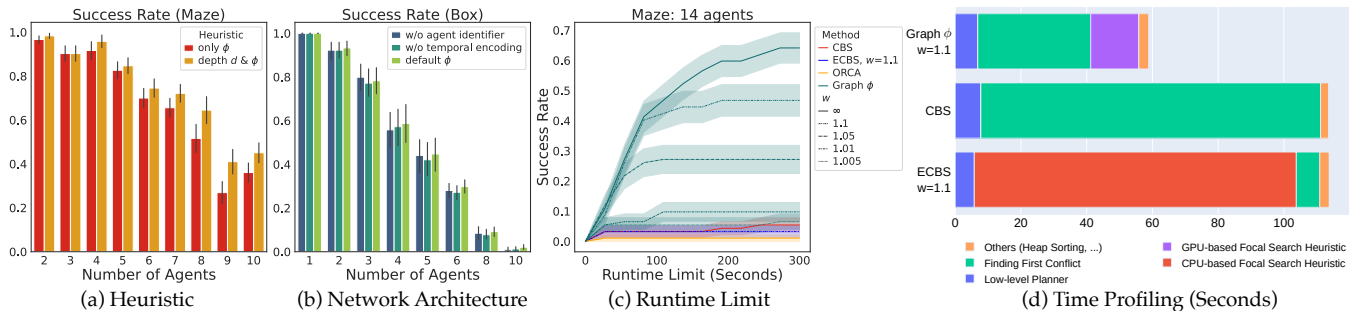
[1]https://github.com/whoenig/libMultiRobotPlanning/

Fig. 5: We conduct 4 various ablation studies to evaluate the proposed method systematically. See Section V-B for more details.

Evaluation Metrics. Our evaluation includes 3 metrics: 1) *Success Rate*, the ratio of the number of successful instances to the total number of test instances. An instance is successful if all agents reach their goals with no collision before the timeout happens. 2) *Computation Time*, the average computational time for each instance, including the failed ones. [2] 3) *Flowtime*, the sum of all the agents' travel time. We only compare the flowtime of the CBS to our method based on the instances where both methods succeed, to show our approach's bounded-suboptimality guarantees.

Overall Performance. We demonstrate the overall performance in Figure 4. Our method significantly outperforms the three baselines in both Maze and Box. It requires much less computation time and achieves significantly higher success rates. Furthermore, though only trained with relatively few numbers of agents, our method generalizes remarkably well to a higher number of agents. For example, our network trained by 2 to 10 agents can be generalized up to 26 agents in Maze, while our policy trained by 1 to 5 agents can be generalized up to 10 agents in Box. In particular, with $w = 1.1$, our method achieves the success rates of $47\%, 23\%, 10\%, 1\%$ in Maze with 14, 18, 22, and 26 agents, and achieves the success rates of $62\%, 30\%.4\%, 0.5\%$ in Box with 4, 6, 8, and 10 agents. On the other side, CBS fails to solve Maze with 18 agents and Box with 10 agents within the timeout. Similarly, ECBS starts to fail in all instances with 20 agents for Maze and with 10 agents for Box. In addition, since the bounded-suboptimality of our algorithm is proved theoretically, it is not surprising to see that the solution qualities (flowtime) of our method are very close to the optimal solutions. Finally, ORCA easily fails in highly dense environments, and such performance is consistent with previous works [45].

B. Ablation Study

In this section, we investigate four questions:

(a) Is incorporating depth information into the heuristic beneficial to the performance? Fig. 5 (a) illustrates the comparison between the heuristic with only ϕ, and with depth d and ϕ, on tests from 2 to 10 agents in Maze. The result shows the effectiveness of introducing depth information. The improvement becomes more noteworthy as the number of agents grows, which validates our choice.

(b) Do the Temporal Encoding and the Agent Identifier improve the performance? Fig. 5 (b) demonstrates the performances in Box with and without the Temporal

Encoding and Agent Identifier. The results show that Agent Identifier and Temporal Encoding improve the success rate. The average improvement of introducing Agent Identifier and Temporal Encoding over the non-default settings is $16\pm33\%$.

(c) How will the runtime limit affect the performance? We take the tests with 14 agents in Maze as an example. We set different runtime limits from 25 seconds to 300 seconds with the interval as 25 seconds. In Fig. 5 (c), we show that our methods outperform the baselines regardless of how the runtime limit changes. When the limit is 300 seconds, our method achieves a success rate of 64% ($w = \infty$), while CBS, ECBS, and ORCA only have 5%, 3%, and 1% respectively.

(d) Time profiling each module of the planners. Finally, we wish to answer the question of why the performances of ECBS considerably degrade once we apply it to non-grid-based MAPF instances, compared to the traditional grid-based settings. We profile each method, and average the results over the Maze tests with 2-10 agents.

Fig. 5 (d) illustrates that ECBS spends considerable computation on the focal heuristic calculation. Such behavior is reasonable, since now for dense graphs, the heuristic is calculated by checking the collisions along edges. Such collision checking is often the main bottleneck for planning and by itself NP-hard in general [46], [47]. Meanwhile, our method uses a learned function for the focal heuristic calculation (the GPU part), which only takes 15% computation cost compared with ECBS. Compared with CBS and ECBS, our method only requires 50% of the total computation time.

VI. CONCLUSION AND FUTURE WORK

We proposed the Graph Transformer as a heuristic function to accelerate CBS by guiding the tree search towards promising nodes with feasible solutions. While achieving significant acceleration, our approach guarantees provable completeness and bounded-suboptimality. We show that in two continuous environments with dense motion graphs, our method outperforms three classical MAPF baselines (CBS, ECBS, and ORCA), while generalizing to unseen tests with a higher number of agents remarkably well. Future works include real-world experiments and improving the method for 10x-100x agents.

---

[2] Since CBS and ECBS call different low-level planners (A* and A*-ϵ), we consider the computation time to be the fairest metric to evaluate the efficiency, instead of counting the number of expanded nodes, for instance.

REFERENCES

[1] J. Enright and P. R. Wurman, "Optimization and coordinated autonomy in mobile fulfillment systems." in *AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, 2011, pp. 33–38.

[2] A. Prorok and V. Kumar, "Privacy-preserving vehicle assignment for mobility-on-demand systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 1869–1876.

[3] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas, "Cooperative air and ground surveillance," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 16–25, 2006.

[4] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," in *Robotics Research*. Springer, 2017, pp. 41–58.

[5] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[6] J. Banfi, N. Basilico, and F. Amigoni, "Intractability of time-optimal multirobot path planning on 2d grid graphs with holes," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 1941–1947, 2017.

[7] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 3612–3617.

[8] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *European Conference on Artificial Intelligence*, 2014, pp. 961–962.

[9] J. Li, W. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, 2021, pp. 12 353–12 362.

[10] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 11 785–11 792.

[11] T. Huang, S. Koenig, and B. Dilkina, "Learning to resolve conflicts for multi-agent path finding with conflict-based search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 246–11 253.

[12] T. Huang, B. Dilkina, and S. Koenig, "Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding," in *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2021.

[13] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[14] E. Lam and P. Le Bodic, "New valid inequalities in branch-and-cut-and-price for multi-agent path finding," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 184–192.

[15] E. Lam, P. Le Bodic, D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent path finding," *Computers & Operations Research*, vol. 144, p. 105809, 2022.

[16] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. S. Kumar, and S. Koenig, "Adding heuristics to conflict-based search for multi-agent path finding," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018, pp. 83–87.

[17] J. Li, A. Felner, E. Boyarski, H. Ma, and S. Koenig, "Improved heuristics for multi-agent path finding with conflict-based search." in *IJCAI*, vol. 2019, 2019, pp. 442–449.

[18] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. R. Sturtevant, J. Schaeffer, and R. Holte, "Partial-expansion a* with selective node generation," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5036

[19] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, M. Fox and D. Poole, Eds. AAAI Press, 2010. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1926

[20] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*. IEEE, 2011, pp. 3260–3267. [Online]. Available: https://doi.org/10.1109/IROS.2011.6095022

[21] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 430–435.

[22] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 1928–1935.

[23] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[24] Q. Sajid, R. Luna, and K. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," in *International symposium on combinatorial search*, vol. 3, no. 1, 2012.

[25] M. Čáp, P. Novák, A. Kleiner, and M. Selecký, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 835–849, 2015.

[26] J. P. Van Den Berg and M. H. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 885–897, 2005.

[27] Q. Li, W. Lin, Z. Liu, and A. Prorok, "Message-aware graph attention networks for large-scale multi-robot path planning," *IEEE Robotics and Automation Letters*, 2021.

[28] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932–6939, 2020.

[29] C. Yu, H. Yu, and S. Gao, "Learning control admissibility models with graph neural networks for multi-agent navigation," 2022. [Online]. Available: https://arxiv.org/abs/2210.09378

[30] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to run heuristics in tree search." in *Ijcai*, 2017, pp. 659–666.

[31] R. Zhang, C. Yu, J. Chen, C. Fan, and S. Gao, "Learning-based motion planning in dynamic environments using gnns and temporal encoding," *CoRR*, vol. abs/2210.08408, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.08408

[32] J. Song, R. Lanka, A. Zhao, A. Bhatnagar, Y. Yue, and M. Ono, "Learning to search via retrospective imitation," *arXiv preprint arXiv:1804.00846*, 2018.

[33] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4274–4289, 2021.

[34] K. Okumura, R. Yonetani, M. Nishimura, and A. Kanezaki, "Ctrms: Learning to construct cooperative timed roadmaps for multi-agent path planning in continuous spaces," *arXiv preprint arXiv:2201.09467*, 2022.

[35] J. Pearl and J. H. Kim, "Studies in semi-admissible heuristics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 4, pp. 392–399, 1982.

[36] M. Ghallab and D. G. Allard, "A: An efficient near admissible heuristic search algorithm," in *Proceedings of International Joint Conference on Artificial Intelligence*. Citeseer, 1983, pp. 789–791.

[37] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[38] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*, 2017, pp. 1263–1272.

[39] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 77–85. [Online]. Available: https://doi.org/10.1109/CVPR.2017.16

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[41] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[42] J. Kim, T. D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong, "Pure transformers are powerful graph learners," *arXiv preprint arXiv:2207.02505*, 2022.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[44] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[45] S. H. Arul, A. J. Sathyamoorthy, S. Patel, M. W. Otte, H. Xu, M. C. Lin, and D. Manocha, "Lswarm: Efficient collision avoidance for large swarms with coverage constraints in complex urban scenes," *IEEE Robotics Autom. Lett.*, vol. 4, no. 4, pp. 3940–3947, 2019. [Online]. Available: https://doi.org/10.1109/LRA.2019.2929981

[46] B. Chazelle, "Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm," *SIAM Journal on Computing*, vol. 13, no. 3, pp. 488–507, 1984.

[47] P. Jiménez, F. Thomas, and C. Torras, "Collision detection algorithms for motion planning," *Robot motion planning and control*, pp. 305–343, 1998.