CB-GCS: Conflict-Based Search on the Graph of Convex Sets for Multi-Agent Motion Planning

Shizhe Zhao[1], Allen George Philip[2], Sivakumar Rathinam[2], Howie Choset[3], Zhongqiang Ren[1†]

***Abstract*— Multi-Agent Motion Planning (MAMP) seeks collision-free trajectories for multiple agents from their respective start to goal locations among static obstacles, while minimizing some cost function over the trajectories. Existing approaches include Mixed-Integer Programming (MIP) models, graph-based and sampling-based methods, and trajectory optimization, each with its own limitations. This paper introduces CB-GCS, a new approach that develops a Conflict-Based Search on the Graph of Convex Sets, to solve the MAMP. CB-GCS plans trajectories for agents in continuous workspaces, represented by time-augmented graphs of convex sets (T-GCS), and resolves agent-agent conflicts by adding constraints to the agents in that T-GCS. We test our CB-GCS against various baselines, including a graph-based method that combines search and sampling, as well as a Mixed-Integer Linear Program (MILP) formulation. The numerical results show that solutions found by our approach often have an optimality gap more than 10 times smaller than those found by the baseline when given the same amount of runtime limit.**

I. Introduction

This paper investigates a Multi-Agent Motion Planning (MAMP) problem, which seeks collision-free trajectories for multiple agents from their respective start to goal locations among static obstacles, while minimizing the sum of trajectory lengths of the agents subject to a time bound, within which all agents must arrive at their goals. The agents have speed limits and can move in any direction. This problem is fundamental in robotics and arises in applications such as logistics and surveillance. MAMP is challenging when searching for high-quality obstacle-free trajectories that avoid agent-agent collision [1], [2].

A. Related Work

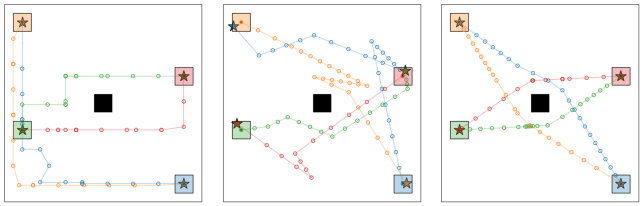
MAMP can be posed as a mixed-integer linear program (MILP) [3], which can be solved by using off-the-shelf solvers such as Gurobi [5] or CPLEX [6]. When MAMP involves nonlinear constraints (e.g., speed limits) or objectives (e.g., Euclidean distance), the problem can no longer be directly posed as a MILP. Although this nonlinearity can be addressed through approximations [7], [8], it may result in highly suboptimal solutions (Fig. 1a). Overall, most mixed-integer programming (MIP) methods for MAMP primarily focus on scalability and on handling complex constraints [9], [10], rather than on solution quality (e.g. Euclidean distance).

[1] Shanghai Jiao Tong University, China. Emails:{shizhe.zhao, zhongqiang.ren}@sjtu.edu.cn

[2] Texas A&M University, College Station, TX 77843-3123. Emails: {y262u297,srathinam}@tamu.edu

[3] Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. Email: {choset@andrew.cmu.edu}

† Corresponding author.



(a) MILP (b) KCBS (c) CB-GCS (ours)

Fig. 1: An illustration of Multi-Agent Motion Planning. Four square agents need to move in an open space to swap their locations. (a) shows the trajectories obtained from the existing MILP model [3] (b) shows the trajectories obtained from KCBS [4] (c) shows the trajectories obtained from our CB-GCS.

Graph-based methods [11]–[15] for MAMP typically involve discretizing the workspace into a graph (such as a state lattice or roadmap), and the agent's action space into a set of motion primitives (i.e., short trajectories connecting two states), to iteratively plan trajectories from their start to goal positions. These methods often find high-quality solutions within the graph. However, generating a graph representation that accurately captures the obstacle-free space and potential agent-agent interactions is challenging: too coarse a discretization may result in no solution, while too fine a discretization may lead to high computational burden.

Sampling-based methods [16], [17] address the aforementioned challenge by iteratively sampling from the state space or the action space of the agents to find collision-free trajectories. These methods quickly find an initial feasible solution and can asymptotically converge to an optimal solution as runtime approaches infinity. However, within a finite runtime, the obtained solution quality can be poor without fine-tuning the sampling process (Fig. 1b). Recent efforts seek to provide solution quality guarantees with a finite number of samples, which is limited to a small number of agents in practice due to the heavy computational burden [16].

Trajectory optimization techniques are used to solve similar multi-agent planning problems by planning the motion of multiple agents with dynamics [18]–[20]. However, these approaches often rely heavily on initialization and can get trapped in local minima or even fail to find a feasible solution in cluttered environments. Local collision avoidance strategies iteratively plan and replan motion around the robots, ensuring they avoid collision with each other in a reactive and decentralized manner [21]–[23]. While these approaches can readily scale to a large number of robots, they

offer no solution quality guarantee due to their myopic local coordination strategy. Finally, other work combines different techniques, such as search and sampling [4], [24], or search and optimization [25], to integrate the benefits of different classes of methods.

B. Contributions

This paper introduces Conflict-Based search on the Graph of Convex Sets (CB-GCS), a new approach for MAMP that requires no sampling of the workspace, is not sensitive to initialization, and can provide near-optimal solutions within a finite runtime limit. On one hand, CB-GCS leverages Conflict-Based Search (CBS) for Multi-Agent Path Finding (MAPF) problems, a technique that resolves agent-agent collisions in a discretized graph representation of the workspace. CB-GCS adapts the conflict resolution techniques from CBS to continuous space to resolve agent-agent collisions. On the other hand, CB-GCS leverages the recent concept of the Graph of Convex Sets (GCS) [26], [27], which has been used to plan paths in continuous spaces for a single robot navigating among static obstacles. CB-GCS plans a path for each agent in a time-augmented GCS, which includes an additional time dimension, to avoid agent-agent conflicts. Additionally, CB-GCS leverages anytime focal search [28] to resolve agent-agent conflicts more efficiently while achieving a bounded suboptimal solution for MAMP.

We test our CB-GCS and compare it against a recent sampling-based method, KCBS [4], as well as a MILP formulation [3]. The results show that CB-GCS can find cheaper (better) solutions (i.e., up to 10x smaller optimality gap) than the baseline at the cost of limited scalability.

II. Problem Statement

Let the index set $I = \{1, 2, \dots, n\}$ denote a set of n agents. Let $\mathcal{W} \subset \mathbb{R}^2$ denote a bounded 2D workspace, that is shared by all the agents. In $\mathcal{W}$, there is a set of static obstacles $\mathcal{O} = \{o_1, o_2, \dots, o_s\}$, where each obstacle is a polygon. For any point $w \in \mathcal{W}$, let $w(x)$ and $w(y)$ denote the x and y coordinate of w.

We assume the agents are squares with the same size $l_w > 0$. An agent's location (also referred to as the agent's reference point) is determined by the position of the center of its square. At any time t, let r_t^i denote the reference point of agent i at t, and let A_t^i denote the square area occupied by agent i at t. Agents can move in any direction. Let $s^i, g^i \in \mathcal{W}$ denote the start and goal positions of agent i. All the agents have the same speed limit, denoted as $v_{max} \in \mathbb{R}^+$, and share the same global clock.

Each agent starts its motion at time $t = 0$ from s^i and ends its motion at g^i at a time that is no later than $\tau_{max} \in \mathbb{R}^+$, where τ_{max} is called the *Time Constraint*. The time dimension is uniformly discretized into a set of time points with a step size of δt, and we let $T = \{0, 1, \dots, t_{max}\}$ denote the index of those time points, where $t_{max} = \tau_{max}/\delta t$.

Let $\pi^i = (r_0^i, r_1^i, \dots, r_{t_{max}}^i)$ denote a path of agent $i \in I$, where $r_0^i = s^i$ and $r_{t_{max}}^i = g^i$. The cost of a path $g(\pi^i) = \sum_{k=0}^{t_{max}-1} ||r_{k+1}^i - r_k^i||_2$ is defined as its length[1].

Problem 1 (TB-MAMP): The goal of the Time Bounded Multi-Agent Motion Planning (TB-MAMP) problem is to find a path π^i for each agent $i \in I$ such that for any $t \in T$, (i) each path π^i is collision free with respect to the static obstacles

$$\forall o \in \mathcal{O},\ A_t^i \cap o = \emptyset \tag{1}$$

(ii) there is no agent-agent collision along the trajectories for each pair of agents

$$\forall i \in I, \forall j \in I \backslash \{i\},\ A_t^i \cap A_t^j = \emptyset \tag{2}$$

and (iii) the sum of individual path costs, $\sum_{i \in I} g(\pi^i)$, reaches the minimum.

Remark 1: Arrival time and path length are two common optimization objectives in the MAMP literature, used by different methods. Search-based methods usually minimize the sum of arrival times at the goals [11], [12], [14], while sampling-based methods usually minimize the sum of trajectory lengths [16], [29]. Some recent papers also seek to combine both objectives as a multi-objective planning problem [30]. This work seeks to minimize the sum of trajectory lengths subject to a time constraint.

III. CB-GCS

CB-GCS consists of high-level and low-level planning. The high-level planning leverages the ideas in CBS [31] and introduces new concepts and techniques to resolve conflicts in continuous spaces (Sec. III-A). The low-level planning seeks a path for a single agent in a time-augmented GCS subject to constraints added by the high-level planning (Sec. III-B).

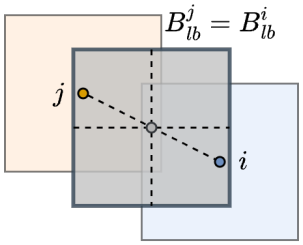
A. High-Level Planning

1) Conflicts and Constraints: For a time step $t \in T$, if two agents collide (i.e., $A^i(r_t^i) \cap A^j(r_t^j) \neq \emptyset$), then we say agents i, j are in conflict. Formally, let $C = (i, j, A_t^i, A_t^j, t)$ denote a conflict between agent i, j at time t. To resolve a conflict $C = (i, j, A_t^i, A_t^j, t)$, a set of four constraints $\Omega = \{\omega_{lb}^i, \omega_{lb}^j, \omega_{ub}^i, \omega_{ub}^j\}$ are generated as follows. Here, lb stands for lower bound and ub stands for upper bound.

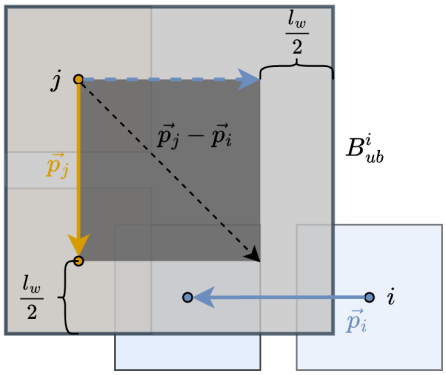
Each constraint $\omega = (i, B^i, t) \in \Omega$ with $B^i \subseteq \mathcal{W}$ representing a region, indicating that the reference point of agent r_t^i is not allowed to enter the region B^i at time t. The four constraints differ in terms of how the forbidden region B^i is generated.

For $\omega_{lb}^i = (i, B_{lb}^i, t), \omega_{lb}^j = (j, B_{lb}^j, t)$, the forbidden area $B_{lb}^i = B_{lb}^j$ is a square centered at the middle point of r_t^i and r_t^j (i.e., centered at $(r_t^i + r_t^j)/2$) with width l_w (Fig. 2a). This pair of constraints are mutually disjunctive [32] in the sense that pair of paths π^i, π^j that violate both of $\omega_{lb}^i, \omega_{lb}^j$ must have a conflict between agent i, j at time t. As a result, if GC-CBS plans by only generating this pair of conflicts, the g cost of any nodes that are generated during planning must be a

[1]We are using the standard L_2-norm for finding the length of any vector.



(a) Lower bound constraint



(b) Upper bound constraint

Fig. 2: An illustration of conflicts. **(a)** Lower bound constraint ensures that there is always a conflict if both r^i_t and r^j_t are inside the constraint region for a given t; thus applying such a constraint never eliminate any feasible solution, thereby preserving completeness and optimality. **(b)** Upper bound constraint expands the constraint region based on relative transition vector of conflicting agents i and j; it eliminates potential solutions where both r^i_t and r^j_t are within the constraint region while not colliding.

lower bound of the true optimum [31]. However, in practice, running CB-GCS with only these two constraints is slow. We therefore introduce two additional constraints, which allow CB-GCS to find a feasible solution more quickly.

For $\omega^i_{ub} = (i, B^i_{ub}, t), \omega^j_{ub} = (j, B^j_{ub}, t)$, the forbidden area B^i_{ub} is a rectangle that bounds the relative "transition vector" of agent i with respect to agent j at time t: Specifically, along the paths π^i, π^j of the two agents, their transition vector at time t can be calculated as $p^i_t = r^i_{t+1} - r^i_t$ and $p^j_t = r^j_{t+1} - r^j_t$ respectively. The relative transition vector of agent j with respect to agent i is $q^{j,i}_t = p^j_t - p^i_t$. Then, B^i_{ub} is an axis-aligned rectangle with lengths along x and y axis equal to $|(q^{j,i}_t(x))| + \epsilon$ and $|q^{j,i}_t(y)| + \epsilon$ respectively, where $\epsilon \geq l_w$ is a positive number representing the additional collision avoidance buffer size. The center of B^i_{ub} is at $r^j_t + q^{j,i}_t/2$ (Fig. 2b). B^j_{ub} is generated in a similar fashion based on the relative transition vector of agent j with respect to agent i. This pair of upper bound constraints is not mutually disjunctive, i.e., there may exist a conflict-free joint path that violates both constraints simultaneously. As a result, $\{\omega^i_{ub}, \omega^j_{ub}\}$ do not guarantee completeness and optimality; the purpose here is to quickly find a feasible solution.

2) Planning Process: The planning process of CB-GCS is shown in Alg. 1. Let $P_k = (\pi_k, g_k, \Omega_k)$ denote a high-level search node, where π_k is a joint path (i.e., a set of paths of all agents) from s^i to g^i for all agents, g_k is the sum of costs of all paths in π_k, and Ω_k is a set of constraints.

To initialize, the shortest path of each agent is planned while ignoring any other agents, and the initial joint path π_0 with cost g_0 is formed. Then, an initial node $P_0 = (\pi_0, g_0, \emptyset)$ is created and added to OPEN, a priority queue that contains a set of high-level nodes and prioritizes these nodes based on their g values from the minimum to the maximum. FOCAL is a subset of OPEN, which will be presented later.

In each planning iteration (Line6-24), a node P_k is popped from OPEN for expansion. Then P_k is first checked for a conflict. If no conflict is found, P_k is returned, whose joint path is a collision-free joint path for all agents. Otherwise, a conflict C is detected, and the aforementioned set of four constraints are generated. For each of those four constraints $\omega^i \in \Omega$, a new corresponding set Ω_l of constraints is formed by adding ω^i to the set of constraints inherited from the current node P_k. Then, low-level planning is invoked for agent i based on the constraint set Ω_l. If no path is found by the low-level planning, the current branch is discarded. Otherwise, the found path π^i_* is used to update agent i's path and all other agents' paths are copied to form a new joint path π_l. A new high-level node $P_l = (\pi_l, g(\pi_l), \Omega_l)$ is created and added to OPEN for future search.

CB-GCS also leverages the anytime focal search technique [28] to speed up the planning while looking for a bounded sub-optimal solution. Specifically, let g_{min} denote the smallest cost of all nodes in OPEN. Let $\epsilon_L \in [0, 1]$ denote a constant real number that indicates the maximum optimality gap for the low level planner, and let $\epsilon_F \geq 0$ denote a non-negative real number to track the current sub-optimal bound for the high level planner. FOCAL consists of nodes in OPEN whose g-costs are within range $[g_{min}, (1+\epsilon_F)g_{min}]$. Initially, ϵ_F is set to infinity (Line 4) in order to quickly find a feasible solution. Afterwards, ϵ_F is reduced each time when a better solution is found (Line 12). In FOCAL, nodes are prioritizes based on the following *collision score* from the minimum to the maximum, where a higher score indicates that there are more collision among the agents. FOCAL pops nodes with smaller collision scores for expansion at first.

To compute the collision score $\beta(P)$ of a node P, for each pair of agents $i, j \in I, i \neq j$ and for each time step $t \in T \backslash \{t_{max}\}$, divide $[t, t+1]$ into $T_K(t)$, a set of K time points that uniformly cover the range $[t, t+1]$. For each of those time points $t_k \in T_K(t)$, interpolate the position of the agents i, j based on their positions at t and $t+1$ along paths π^i, π^j, and compute the overlapped area $B^{i,j}_{t_k}$. The collision score $\beta(P)$ is the sum of all areas $|B^{i,j}_{t_k}|$ over all time points in $T_K(t)$ for all $t \in T$ and for all pairs of agents, i.e., $\beta(P) = \sum_{i,j \in I, i \neq j} \sum_{t \in T} \sum_{t_k \in T_K(t)} |B^{i,j}_{t_k}|$.

Remark 2: Alg 1 preserves all feasible solution in the search space, even upper bound constraints may eliminate feasible solutions. We justify this by demonstrating that there is always a high level search node that preserves potential

feasible solutions in its successors. Since lower bound constraints never eliminate feasible solutions, a high-level search node $P_i = (\pi_i, g_i, \Omega_i)$ preserves feasible solutions if there is no upper bound constraints in Ω_i, referred to as a complete node. Clearly, as long as a conflict exists, a complete node always has two successors that are also complete, which means there will always be complete nodes in the open list.

Remark 3: When there is no feasible solution, CB-GCS cannot terminate and requires an external method to determine the feasibility, inheriting the behaviour of classic CBS [31]. When feasible solutions exist, unlike the classic CBS, which guarantees finding the optimal solution in finite time, CB-GCS finds the optimal solution asymptotically. This is because, in continuous space, there are an infinite possible number of possible constraints per time step, so within finite number of iterations, we cannot guarantee generating a set of constraints that defines the optimal solution.

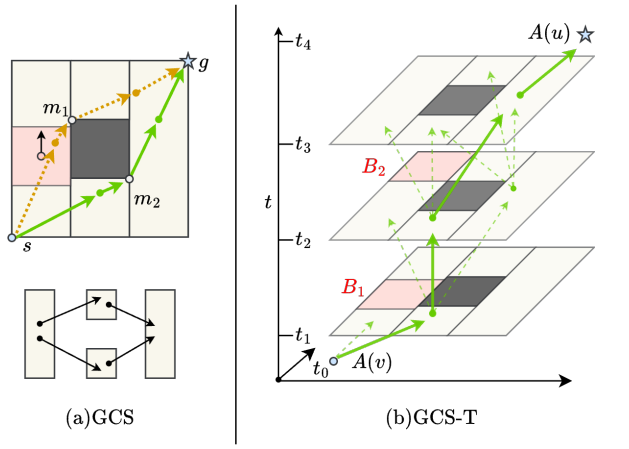
B. Low-Level Planning

1) Time-Augmented Graph of Convex Sets: Let $G = (V, E)$ denote a GCS, where each vertex $v \in V$ is associated with a convex set $A(v) \subseteq \mathcal{W}$, and each edge $(u, v) \in E$ represents a possible transition of an agent from one convex set to another (Fig. 3(a)). Here, we abuse the notation A: unlike the notation A_t^i which indicates the area occupied by agent i at some $t \in T$, we use the notation $A(v)$ to denote the convex set associated with vertex v in GCS. The focus of this work is not on how to generate the convex sets for a cluttered workspace. Instead, the GCS is given, and the focus is on path planning.

For a set of constraints Ω, let Ω^i denote the subset of constraints that are defined for agent i, where agent i is

Algorithm 1 Pseudocode for CB-GCS

```
Compute P_root and insert into OPEN and FOCAL
C ← ∅
P* ← Null                                   ▷ Best result so far
ε_F ← ∞
g_lb ← g_0(1 − ε_L)
while FOCAL not empty and Not timeout do
    P_k = (π_k, g_k, Ω_k) ← FOCAL.top()
    Remove P_k from FOCAL and OPEN
    C ← DetectConflict(π_k)
    if C = ∅ then
        P* = (π*, g*, Ω*) ← P_k             ▷ Record current best plan
        ε_F ← g_k / g_lb                    ▷ Reduce sub-optimal bound
        continue
    Ω ← GenConstraints(C)
    for all ω^i ∈ Ω do
        Ω_l = Ω_k ∪ {ω^i}
        π*_i ← LowLevelPlan(i, Ω_l, ε_L)
        if π*_i = ∅ then
            continue
        π_l ← π_k, replace π^i_l (in π_l) with π*_i
        g_l ← g(π_l))
        P_l = (π_l, g_l, Ω_l)
        Add P_l to OPEN
        Update FOCAL based on ε_F
return P* or Failure
```



(a)GCS (b)GCS-T

Fig. 3: An illustration of GCS and T-GCS, where blue dot and star represents the starting location and goal location of an agent respectively, pink polygon represents a dynamic obstacle and black polygon represents a static obstacle. **(a)** shows the GCS ignoring the dynamic obstacle. There are four convex sets in the workspace, and two optimal paths from s to g colored in brown and green respectively. The brown path collide with the dynamic obstacle. **(b)** shows the T-GCS. Green arrows indicate the adjacent convex sets at the next layer, while solid arrows represent the edges used by the optimal path. The convex sets differs at each time layer, e.g., the number of convex sets are 5,4 and 4 at t_1, t_2 and t_3 respectively.

to be planned by the low-level planning. Given a GCS G, a set of time steps $T = \{0, 1, 2, \cdots, t_{max}\}$ and a set of constraints Ω, we define a time-augmented GCS (T-GCS), $G_T = (V_T, E_T)$, where V_T contains the vertices of a collection of GCSs, organized in $t_{max} + 1$ layers where each layer corresponds to a GCS for time step $t \in T$ (Fig. 3(b)). Let $G_t = (V_t, E_t)$ denote the GCS for a layer $t \in T$. The convex sets in G_t should not intersect with any forbidden areas B_t that are defined in any constraint $\omega^i = (i, B^i, t) \in \Omega^i$ related to agent i. As a result, GCSs at different layers may not be the same. Additionally, for $t = 0$, let $G_{t=0}$ be a graph containing a single vertex v whose corresponding convex set $A(v) = \{s^i\}$ contains only the start location of agent i, and let $G_{t=t_{max}}$ be a graph containing a single vertex u whose corresponding convex set $A(u) = \{g^i\}$ contains only the goal location of agent i.

E_T is a set of directional edges (i.e, arcs) representing the transition from one convex set at layer $t \in T \backslash \{t_{max}\}$ to another convex set at layer $t + 1$. Given V_T, we determine the edge sets by considering the speed limit of the agents: an edge from $v \in G_t$ to $u \in G_{t+1}$ exists if there exists at least one pair of points $p \in A(v), q \in A(u)$ such that $||q - p||_2 \leq v_{max}\delta t$.

2) Bilinear Program: Given a T-GCS G_T, a similar Mixed Integer Conic Program (MICP) as in [26], [27] can be used to find a path for agent i. We first present a bilinear formulation of the problem in this subsection, and then

convert it to a MICP formulation in the next subsection.

Let $q_e \in \{0,1\}$ denote the flow variable associated with an edge $e \in G_T$. Let $E^{out}_{v,t}$ (and $E^{in}_{v,t}$) denote the set of out-going (and in-coming) edges in G_T from a vertex v in G_t, the GCS at the t-th layer. To form a path in G_T, the following flow constraints must be enforced.

$$\sum_{e \in E^{out}_{v,t}} q_e = \sum_{e \in E^{in}_{v,t}} q_e \leq 1, \ \forall v \in G_t, t \in T \setminus \{0, t_{max}\} \quad (3)$$

$$\sum_{e \in E^{out}_{v,0}} q_e = 1, \quad \sum_{e \in E^{in}_{v,t_{max}}} q_e = 1. \quad (4)$$

Let $p_{v,t} \in \mathbb{R}^2$ denote the point selected in the convex set $A(v)$ for some $v \in G_t, t \in T$:

$$p_{v,t} \in A(v), \forall \ v \in G_t, t \in T. \quad (5)$$

For an edge $e = (u, v)$ connecting two adjacent nodes in two adjacent layers of GCS in G_T, the Euclidean distance between the points selected, $p_{u,t}$, $p_{v,t+1}$, and the speed constraints can be formulated as follows:

$$l_{x,t} = p_{v,t+1}(x) - p_{u,t}(x), \quad (6)$$

$$l_{y,t} = p_{v,t+1}(y) - p_{u,t}(y), \quad (7)$$

$$l^2_{x,t} + l^2_{y,t} \leq l^2_e, \quad (8)$$

$$l_e \leq v_{max} \delta t. \quad (9)$$

The objective function to be minimized is:

$$\min \sum_{e \in E_T} q_e l_e. \quad (10)$$

The program is bilinear as the objective function involves non-convexity caused by the product of flow variable q_e and the distance variable l_e.

3) Mixed Integer Conic Program: For an edge $e = (u, v) \in E_T$ and its flow variable q_e, let $z_e = q_e p_u$ and $z'_e = q_e p_v$, where z_e (and z'_e) be the product of the flow variable of edge e and the position variable p_u (and p_v) in the convex set at the source vertex u (and target vertex v) of edge e (respectively).

The flow constraints in Eq.3 can be reformulated as:

$$\sum_{e \in E^{out}_{v,t}} (q_e, z_e) = \sum_{e \in E^{in}_{v,t}} (q_e, z'_e), \ \forall v \in G_t, t \in T \setminus \{0, t_{max}\}, \quad (11)$$

$$\sum_{e \in E^{out}_{v,t}} q_e \leq 1. \ \forall v \in G_t, t \in T \setminus \{0, t_{max}\}. \quad (12)$$

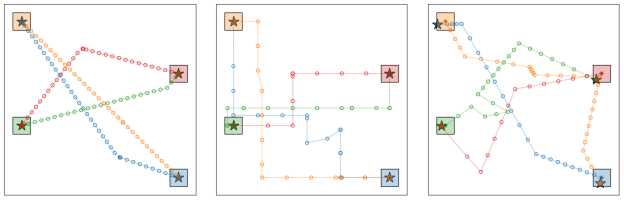
Define $\tilde{l}_e = \tilde{l}_e(z_e, z'_e, q_e)$ as a perspective function: When $q_e \neq 0$, $\tilde{l}_e(z_e, z'_e, q_e) := l_e(z_e/q_e, z'_e/q_e)q_e = l_e(x_u, x_v)q_e$. Otherwise, when $q_e = 0$, the perspective function returns 0. The speed constraint in Eq. 9 can be reformulated as:

$$\tilde{l}_e(z_e, z'_e, q_e) \leq v_{max} \delta t. \quad (13)$$

Let $\tilde{A}(v)$ denote the perspective of convex set $A(v)$ (which is still a convex set). Then, z_e, z'_e of edge $e = (u, v), u \in G_t, v \in G_{t+1}$ belongs to the following sets:

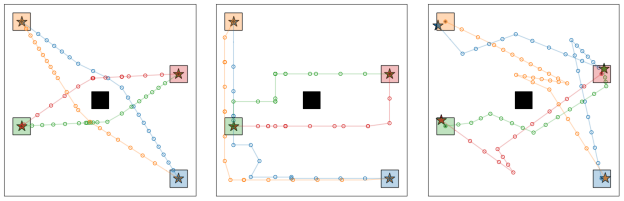
$$(z_e, q_e) \in \tilde{A}(u), \quad (14)$$

$$(z'_e, q_e) \in \tilde{A}(v). \quad (15)$$



(a) CB-GCS (ours) (b) MILP (c) KCBS

Empty



(d) CB-GCS (ours) (e) MILP (f) KCBS

Mid-blocked

Fig. 4: Test environments and sample solution paths returned by different methods.

The objective function to be minimized is:

$$\min \sum_{e \in E_T} \tilde{l}_e. \quad (16)$$

The validity of this reformulation follows from Theorem 5.7 in [26].

Remark 4: This paper assumes time steps T and the step size δt are given. In practice, T and δt can impact the low-level planning and should be chosen properly. Increasing δt influences high level planning by allowing agents to move over a longer range per time step, leading to a larger upper bound constraint regions for the agents, and may reduce the solution quality. Conversely, decreasing δt slows down the low level planning, since a smaller δt leads to more layers of T-GCS, and increases the number of variables and constraints in the bilinear program.

IV. Experimental Results

A. Test Settings

We consider two environments, *Empty* and *Mid-blocked*. Both environments are 10 by 10 workspace, where agents locate at the border and need to swap their locations, i.e., each agent's goal is the starting location of another agent. These environments describe a scenario where all single-agent shortest paths cross at the same position. We aim to use *Empty* to evaluate the high-level planner, and *Mid-blocked* to demonstrate the impact of obstacles on the low-level planner. Fig 4 illustrates the environments.

B. Implementation and Baselines

We implement CB-GCS in Python and use Gurobi 11 [5] as the solver. In our tests, we let Gurobi terminate if the optimality gap (i.e., the gap between the lower bound and the upper bound computed by the solver) is within 5%, i.e.,

	n	g_{lb}	CB-GCS			MILP			KCBS		
			D_{1st} (time)	D^*	$\triangle\%$	D_{1st} (time)	D^*	$\triangle\%$	D_{1st} (time)	D^*	$\triangle\%$
Empty	2	24.2	25.3 (0.25s)	24.8	2%	35.7 (0.0s)	35.7	32%	32.7 (0.1s)	32.7	26%
	4	42.2	43.7 (5.9s)	43.3	2%	64.3 (0.0s)	64.3	34%	71.8 (1.0s)	59.4	29%
	6	61.8	70.1 (3.9s)	69.9	12%	97.1 (0.0s)	92.9	33%	111.0 (2.8s)	84.3	27%
Mid-blocked	2	23.9	25.4 (1.4s)	25.0	4%	35.4 (0.0s)	35.4	32%	43.9 (0.0s)	43.9	46%
	4	41.7	43.8 (38.1s)	43.8	5%	73.0 (0.0s)	73.0	43%	137.1 (0.6s)	65.4	36%
	6	61.2	inf	inf	inf	100.1 (0.0s)	90.4	32%	110.9 (3.4s)	83.9	27%

TABLE I: Experiment results. g_{lb} is the lower bound computed by Alg. 1 line 5, D_{1st} is the solution quality of the first feasible solution, D^* is the best solution quality after termination, and $\triangle$ is the optimality gap.

$\epsilon_L = 0.05$. All experiments[2] were run on a desktop with a 16-core i7-13700 CPU and 32GB RAM on Ubuntu 22.04. Baseline methods are as follows:

a) KCBS [4]: It follows the workflow of Conflict-Based Search (CBS) [31]. *KCBS* employs a sampling-based method (e.g. RRT [33]) at the low-level to find path for each agent in the continuous space. The original *KCBS* incorporates kinodynamic constraints into the low-level planner, which is not required in this work. In addition, *KCBS* does not consider the time constraint τ_{max} (Sec. II) while our MICP does. Finally, the original *KCBS* prioritizes the search by the number of collisions rather than the traversed distance and terminates after the first solution is found. We adapted *KCBS* to align with our problem setting by allowing it to keep running CBS-like search for further improvement after finding the first solution. This will persist until either the time limit is reached or all CBS search nodes are explored.

b) MILP [3]: MILP uses the sum of L_1-norm as the objective function:

$$L_1 = \sum_t |r_t^i, r_{t+1}^i|, \tag{17}$$

and we approximate the speed limits by

$$|r^i, r_{t+1}^i| \leq \sqrt{2}\delta t v_{max}. \tag{18}$$

Details of this MILP formulation can be found in [3].

We refer to a (problem) instance as a specific number of agents in a specific environment. We set a 100 seconds runtime limit (i.e., *TLimit*=100s) for each instance, and fix the parameters $\delta t = 0.5, v_{max} = 2, \tau_{max} = 10$.[3]

We vary the number of agent $n \in \{2, 4, 6\}$ for each environment. For each instance, we examine the following metrics: the sum of cost of the first feasible solution D_{1st} and the corresponding time, the best solution after termination, denoted as D^*, and the optimality gap $\triangle$. For both metrics, smaller values are better. We consider $g_{lb} = g_0(1 - \epsilon_L)$ as the lower bound of an instance, where g_0 is the initial sum of costs in CB-GCS that ignores all other agents.

C. Numerical Results

Table I shows the results. We can see that CB-GCS takes more time to find the first solution but consistently provides better solutions than MILP and KCBS. Ultimately CB-GCS terminates with a smaller optimality gap. For example, for $n = 4$ in Empty, the optimality gap of CB-GCS is 2%, which is 10 times smaller than others (near 30%). MILP can instantly find a feasible solution (within 0.1s), but it rarely improves the solution further in the remaining time. This is because the L_1-norm in the objective function doesn't align with our Euclidean distance objective. This misalignment causes early termination (MILP mostly finish within 10s) as MILP reaches the optimality gap defined by L_1-norm. The L_1-norm objective also causes the MILP to prefer rectilinear-like paths (i.e., movements are vertical or horizontal as shown in Fig. 4), which impacts the solution quality. KCBS can quickly find the first feasible solution and continues to improve it over time, but it cannot effectively reduce the optimality gap. For example, in *Mid-blocked* $n = 4$, it finds feasible solution in 0.6s, but ends up with $\triangle = 36\%$ after 100s. This is because the low-level sampling-based planner is unbounded, preventing it from improving the overall solution quality by applying high-level constraints.

We can also observe that instances from *Empty* have higher g_{lb} than those from *Mid-blocked*. This indicates a drawback of our low-level planner (T-GCS): it is significantly impacted by the presence of obstacles. In fact, the static obstacle at the center of *Mid-blocked* causes at least 4 convex sets per layer (depending on the convex generation method). The number of edges in *Mid-blocked* is 10x more than in the *Empty*, which significantly slows down the program (Eq (3),(4)). As the result, for the same number of agents, the runtime for finding the first feasible solution in *Mid-blocked* is much higher than in *Empty*, and it failed to find a solution on *Mid-blocked* when $n = 6$, which is also the reason we test with up to 6 agents.

V. Conclusion and Future Work

This paper introduces CB-GCS for MAMP that combines time-augmented Graph of Convex Sets (T-GCS) and Conflict-Based Search (CBS). CB-GCS can find high quality solution in continuous space. Numerical results show that it provides solutions with an optimality gap more than 10x smaller compared to other competitors (MILP and KCBS). However, the size of T-GCS can be significantly impacted by the topology of the environment, which slows down the low-level planning.

Future work includes improving the convex set generation to reduce the size of the graph in CB-GCS. Given that

[2]Our implementation is available online `https://github.com/rap-lab-org/public_CB-GCS`

[3]We use "τ_{max}" to denote the time constraint in the TB-MAMP problem formulation as described in Sec. II, and use "runtime limit $TLimit$" to denote the runtime limit for each instance.

runtime of T-GCS significantly increases in the presence of obstacles, one could consider adapting implicit graph search [34] to T-GCS, to leverage techniques such as parallelization and anytime planning.

VI. ACKNOWLEDGMENT

The main ideas in this paper originated during Dr. Ren and Allen's work at CMU and TAMU respectively. This material is partially based on the work supported by the National Science Foundation (NSF) under Grant No. 2120219 and 2120529. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect views of the NSF.

Dr. Ren and Dr. Zhao in the last few months were also partially supported by the Natural Science Foundation of Shanghai under Grant 24ZR1435900, and the Natural Science Foundation of China under Grant 62403313.

REFERENCES

[1] K. Solovey and D. Halperin, "On the hardness of unlabeled multi-robot motion planning," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1750–1759, 2016.

[2] J. K. Johnson, "On the relationship between dynamics and complexity in multi-agent collision avoidance," *Autonomous Robots*, vol. 42, no. 7, pp. 1389–1404, 2018.

[3] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *2001 European Control Conference (ECC)*, 2001, pp. 2603–2608.

[4] J. Kottinger, S. Almagor, and M. Lahijanian, "Conflict-based search for multi-robot motion planning with kinodynamic constraints," in *IROS 2022, Kyoto, Japan, October 23-27, 2022*. IEEE, 2022, pp. 13 494–13 499.

[5] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[6] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[7] M. G. Earl and R. D'andrea, "Iterative milp methods for vehicle-control problems," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1158–1167, 2005.

[8] D. Ioan, I. Prodan, S. Olaru, F. Stoican, and S.-I. Niculescu, "Mixed-integer programming in motion planning," *Annual Reviews in Control*, vol. 51, pp. 65–87, 2021.

[9] K. Leahy, Z. Serlin, C.-I. Vasile, A. Schoer, A. M. Jones, R. Tron, and C. Belta, "Scalable and robust algorithms for task-based coordination from high-level specifications (scratches)," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2516–2535, 2021.

[10] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1189–1206, 2019.

[11] L. Cohen, T. Uras, T. S. Kumar, and S. Koenig, "Optimal and bounded-suboptimal multi-agent motion planning," in *SOCS*, 2019.

[12] A. Andreychuk, K. S. Yakovlev, P. Surynek, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," *Artif. Intell.*, vol. 305, p. 103662, 2022.

[13] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *SoCS*, vol. 10, no. 1, 2019, pp. 151–158.

[14] Z. Ren, S. Rathinam, and H. Choset, "Loosely synchronized search for multi-agent path finding with asynchronous actions," in *2021 IROS*. IEEE, 2021, pp. 9714–9719.

[15] J. Chen, J. Li, C. Fan, and B. C. Williams, "Scalable and safe multi-agent motion planning with nonlinear dynamics and bounded disturbances," in *Thirty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, 2021, pp. 11 237–11 245.

[16] D. Dayan, K. Solovey, M. Pavone, and D. Halperin, "Near-optimal multi-robot motion planning with finite sampling," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3422–3436, 2023.

[17] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "A general task and motion planning framework for multiple manipulators," in *IROS*, 2021, pp. 3168–3174.

[18] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics Autom. Lett.*, vol. 5, no. 2, pp. 604–611, 2020.

[19] J. Tordesillas and J. P. How, "Mader: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, 2022.

[20] L. Ferranti, L. Lyons, R. R. Negenborn, T. Keviczky, and J. Alonso-Mora, "Distributed nonlinear trajectory optimization for multi-robot motion planning," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 2, pp. 809–824, 2022.

[21] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The international journal of robotics research*, vol. 17, no. 7, pp. 760–772, 1998.

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

[23] S. Ruan, Q. Ma, K. L. Poblete, Y. Yan, and G. S. Chirikjian, "Path planning for ellipsoidal robots and general obstacles via closed-form characterization of minkowski operations," in *Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020.

[24] D. Le and E. Plaku, "Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1868–1875, 2019.

[25] B. Senbaslar, W. Hönig, and N. Ayanian, "RLSS: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations," *Auton. Robots*, vol. 47, no. 7, pp. 921–946, 2023.

[26] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.

[27] K. Sundar and S. Rathinam, "A* for graphs of convex sets," 2024. [Online]. Available: https://arxiv.org/abs/2407.17413

[28] L. Cohen, M. Greco, H. Ma, C. Hernández, A. Felner, T. S. Kumar, and S. Koenig, "Anytime Focal Search with Applications." in *IJCAI*, pp. 1434–1441.

[29] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "drrt*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3, pp. 443–467, 2020.

[30] Z. Ren, C. Zhang, S. Rathinam, and H. Choset, "Search algorithms for multi-agent teamwise cooperative path finding," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1407–1413.

[31] 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.

[32] J. Li, P. Surynek, A. Felner, H. Ma, T. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7627–7634.

[33] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[34] R. Natarajan, C. Liu, H. Choset, and M. Likhachev, "Implicit graph search for planning on graphs of convex sets," in *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, D. Kulic, G. Venture, K. E. Bekris, and E. Coronado, Eds., 2024.