Extending Conflict-Based Search for Optimal and Efficient Quadrotor Swarm Motion Planning

Zihao Wang , Zhiwei Zhang , Wenying Dou , Guangpeng Hu , Lifu Zhang , Meng Zhang *

Posted Date: 16 October 2024

doi: 10.20944/preprints202410.1212.v1

Keywords: Multi-Agent Pathfinding; Conflict-Based Search; Quadrotor Swarm Motion Planning; Motion Primitive

Article

Extending Conflict-Based Search for Optimal and Efficient Quadrotor Swarm Motion Planning

Zihao Wang [1,2], Zhiwei Zhang [3], Wenying Dou [4], Guangpeng Hu [5], Lifu Zhang [3] and Meng Zhang [1,2,*]

1 Institute of Information Engineering, Chinese Academy of Sciences
2 School of Cyber Security, University of Chinese Academy of Sciences
3 Aerospace Information Research Institute, Chinese Academy of Sciences
4 School of Electronics and Information Engineering, Beihang University
5 School of Computer Science, Northwestern Polytechnical University
* Correspondence: zhangmeng@iie.ac.cn

Abstract: Multi-agent pathfinding has been extensively studied in the robotics and artificial intelligence communities. The classical algorithm, Conflict-Based Search(CBS), is widely used in various real-world applications due to its ability to solve large-scale conflict-free paths. However, classical CBS assumes discrete time-space planning and overlooks physical constraints in actual scenarios, making it unsuitable for direct application in Unmanned Aerial Vehicle(UAV) swarm. Inspired by the decentralized planning and centralized conflict resolution ideas of CBS, we propose an optimal and efficient UAV swarm motion planner that integrates State Lattice with CBS, named SL-CBS. SL-CBS is a two-layer search algorithm: (1) The low-level search utilizes an improved State Lattice. We design emergency stop motion primitives to ensure complete UAV dynamics and handle spatio-temporal constraints from high-level conflicts. (2) The high-level algorithm defines comprehensive conflict types and proposes a motion primitive conflict detection method with linear-time complexity based on Sturm's theory. Additionally, our modified Independence Detection(ID) technique is applied to enable parallel conflict processing. We validate the planning capabilities of SL-CBS in classical scenarios and compare it with State-Of-The-Art(SOTA) algorithms, showing great improvements in success rate, computation time, and flight time. Finally, we conduct large-scale tests to analyze the performance boundaries of SL-CBS+ID.

Keywords: multi-agent pathfinding; conflict-based search; quadrotor swarm motion planning; motion primitive

1. Introduction

Swarm motion planning is a fundamental technology for the autonomous flight of Unmanned Aerial Vehicle(UAV) swarm, aiming to solve a dynamically feasible and collision-free flight trajectory for each one. Based on the organizational structure of the swarm, swarm motion planning generally employs either centralized or decentralized methods. Centralized methods struggle to provide feasible solutions within a limited time due to the curse of dimensionality [1]. In contrast, decentralized methods plan independently, offering fast computation but often lacking optimality, resulting in poor solution quality and potential deadlocks(Especially in symmetrical scenarios) [2,3]. Balancing optimality, high-quality solutions, and computational efficiency remains an unresolved challenge in swarm motion planning.

A simplified problem corresponding to swarm motion planning in discrete time-space is Multi-Agent PathFinding(MAPF). MAPF is an NP-hard graph search problem that seeks to find multiple conflict-free paths [4]. The classical algorithm Conflict-Based Search(CBS) and its variants have made great progress in recent years and have been widely applied in fields such as warehouse management, airport towing, and railway planning [5]. CBS consists of two layers: the low-level algorithm plans a collision-free path for each agent in a decentralized manner, like A* or SIPP(Safe

 doi:10.20944/preprints202410.1212.v1

Interval Path Planning). The high-level algorithm centrally detects conflicts among all paths and resolves them by imposing constraints at the low level, iterating until no conflicts are found [6]. However, due to the movement constraints of agents in MAPF and various physical assumptions, such as 4/8-neighbors search space and waiting at the same location, CBS is difficult to apply directly to robots like UAVs that require continuous motion planning.

Inspired by the CBS algorithm for solving MAPF, its decentralized planning and centralized conflict resolution provide a new approach for UAV swarm motion planning. By enhancing the low-level discrete path planning to continuous motion planning that accounts for UAV dynamics, defining new conflict types and implementing conflict detection methods at the high level, the CBS can be extended to swarm motion planning.

Similarly, we integrate State Lattice with CBS to propose an optimal and efficient swarm motion planner suitable for a quadrotor swarm, named SL-CBS. The SL-CBS algorithm is also divided into two layers. The low-level is an improved State Lattice based on motion primitive search. To implement waiting actions, we design an emergency stop motion primitive. Besides, we handle spatio-temporal constraints from high-level conflict resolution. At the high level, we define comprehensive conflict types, particularly motion primitive conflicts. Instead of sampling in the time dimension for conflict detection, we propose a linear complexity conflict detection method based on Sturm's theory. Additionally, our modified Independent Detection(ID) technology applied to the SL-CBS provides the ability to parallel process conflicts, enhancing planning ability in certain scenarios. Our main contributions are as follows:

(1) We extend the CBS algorithm to quadrotor swarm motion planning and propose the optimal and efficient SL-CBS planner integrating State Lattice and CBS.
(2) At the low-level, we improve State Lattice based on motion primitive search that considers a complete dynamic model. We design emergency stop motion primitives and address corresponding spatio-temporal constraints.
(3) At the high-level, we define motion primitive conflicts and propose a conflict detection method with linear time complexity based on Sturm's theory.
(4) We validate the planning capabilities of SL-CBS in classical scenarios such as CrossAndSwap, Cross and Swap.
(5) The benchmarks are designed, and compared to the latest State-Of-The-Art(SOTA) algorithms, SL-CBS demonstrates higher success rates, reduced computation time, and lower flight costs.
(6) We analyze the performance boundaries of SL-CBS+ID in large-scale batch testing.

2. Related Work

2.1. MAPF

Consider an undirected graph $G(V, E)$, where V represents vertices and E represents edges connecting two vertices, assume a set of K agents $\{a_1, a_2, \ldots, a_K\}$ with starting positions $s = (s_1, s_2, \ldots, s_K)$ and goal positions $g = (g_1, g_2, \ldots, g_K)$, where $s_i \in V$ and $g_i \in V$ for $1 \leq i \leq K$. Time is discretized as $t = 1, 2, \ldots$. At each time step, agents can perform one of two actions: move to a neighboring node (4/8-neighbors) or wait at their current position. Each action incurs a unit cost of 1. The objective of MAPF is to determine conflict-free paths for each agent while minimizing the total travel cost. Conflicts typically fall into two categories: vertex conflicts and edge conflicts. A vertex conflict $\langle a_i, a_j, v_i, v_j, t \rangle$ indicates that agents a_i and a_j are in conflict at positions v_i and v_j respectively at time t, typically with $v_i = v_j$. An edge conflict $\langle a_i, a_j, v_i, v_{i+1}, v_j, v_{j+1}, t \rangle$ occurs when agent a_i moves from v_i to v_{i+1} and agent a_j moves from v_j to v_{j+1} at time t, typically with $v_i = v_{j+1}$ and $v_{i+1} = v_j$.

2.2. CBS and Its Developments

The main methods for solving the MAPF include A*-based methods, reduction-based methods, ICTS and CBS, with detailed reviews in [7,8]. In recent years, the CBS algorithm has attracted

 doi:10.20944/preprints202410.1212.v1

significant interest from researchers and scholars due to its powerful conflict resolution capabilities and large-scale conflict-free paths planning abilities.

CBS is a two-layer search algorithm. The low-level finds optimal paths for each agent that satisfy spatio-temporal constraints, often using the A* algorithm. The high-level maintains a Constraint Tree CT, where each node $N \in CT$ consists of three components: (1) a set of constraints applied to the agents, $N.constraints$; (2) a set of solutions satisfying all constraints, $N.solutions$; (3) the cost of the solutions, $N.cost$. The root node of the CT is constructed from the solutions of single-agent planning without constraints, and the optimal node is selected for expansion based on a greedy search strategy. After selecting a node for expansion, the high-level algorithm checks for conflicts among the agents. If no conflicts exist, the current node is considered as the goal node, and the high-level algorithm terminates, returning the solutions. Otherwise, CBS selects a conflict to resolve, such as $\langle a_i, a_j, v_i, v_j, t \rangle$. Two child nodes are created, inheriting the parent's constraints and adding constraints $\langle a_i, v_i, t \rangle$ and $\langle a_j, v_j, t \rangle$ to each child, respectively. The constraint $\langle a_i, v_i, t \rangle$ prohibits agent a_i from moving to vertex v_i at time t, and similarly for $\langle a_j, v_j, t \rangle$. After adding constraints, the child nodes are replanned using the low-level algorithm and added to the CT.

Subsequent developments in CBS have been significant. [9] classifies conflicts among agents into cardinal, semi-cardinal, and non-cardinal conflicts, prioritizing the resolution of cardinal conflicts. For semi-cardinal and non-cardinal conflicts, agents with alternative paths of equal cost are rerouted to bypass conflicts [10]. In [11,12], symmetry-breaking technique and weighted dependency graph heuristic are proposed for 4-connected grid maps, solving paths for hundreds of agents in dense maps.

2.3. Swarm Motion Planning Using CBS

In existing work, the CCBS algorithm proposed in [13] is an attempt to extend CBS to real-world robots. It defines state conflict types that consider the geometric sizes of robots in the high-level algorithm and computes unsafe time intervals. The low-level search uses the SIPP for continuous-time planning to handle unsafe time interval constraints. Although subsequent works, CCBS+ [14] and CCBS-PGA [15] employ techniques like Prioritizing Conflicts(PC), Disjoint Splitting(DC), High-level Heuristics(HH), and Prioritized Planning(PP) to reduce computing time, they are limited to disc robots and have many physical constraints, such as simple motion models, constant velocity, and ignoring inertia. The CBS-MP proposed in [16] constructs a roadmap using the PRM sampling algorithm during preprocessing, providing feasible paths for all agents. Although roadmap-based CBS efficiently solves paths, the low-level algorithm does not consider dynamic and kinematic models, making the paths suitable only for discrete motion operators.

The work in [17] first integrates the CBS with the Hybrid-State A* algorithm considering dynamics, proposing the CL-CBS for Ackermann-steering vehicles. However, the low-level Hybrid-State A* requires vehicles to move at constant speed, accelerate instantly to constant speed, and brake instantly, which is unrealistic. Moreover, the CBS high-level only considers state conflicts between vehicles, ignoring motion primitive conflicts, leading to a lack of safety guarantees. Although subsequent work HG-CBS [18] improves computational efficiency through grouping and improved search strategies, it does not overcome the original issues. The K-CBS proposed in [19] uses a sampling algorithm considering full dynamics for low-level planning, with high-level algorithms performing state conflict detection after uniform time sampling to determine conflict intervals. Although its multiple motion trajectory results are suitable for real-world swarm motion scenarios, inefficient conflict detection and single-agent planning limit its scalability. In [20], CB-MPC fully considers the dynamic and kinematic models of nonlinear unicycles, using an MPC optimal controller for CBS's low-level algorithm. It solves multiple motion trajectories for complex scenarios through iterative continuous planning, but the large computational load and long optimization time of the MPC optimal control problem result in low overall computational efficiency. The latest db-CBS [21] implements low-level planning based on db-A* [22] , which pre-solves two-point boundary value problems to compute motion primitives meeting dynamic requirements, followed by bounded discontinuous A* search for motion trajectories.

 doi:10.20944/preprints202410.1212.v1

Outside the two-layer CBS algorithm, db-CBS designs a multi-robot trajectory optimization layer to address bounded discontinuities in low-level planning. db-CBS efficiently solves large-scale swarm motion trajectories but relies on time-consuming offline pre-computation of motion primitives.

The aforementioned works either do not consider the complete dynamics of the robots, lack safety guarantees, or suffer from low computational efficiency and require offline pre-computation. Our goal is to overcome these limitations by proposing an optimal and efficient UAV swarm motion planner, which takes into account the full dynamics, based on the CBS.

The organization of this paper is as follows: Section 3 formally defines UAV swarm motion planning problem. In Section 4, we introduce the SL-CBS+ID algorithm framework and its corresponding components. Section 5 presents the experimental results, including tests on classical scenarios, comparisons with baseline algorithms, and performance boundary analysis. Finally, Section 6 provides a summary of the work and suggests potential future research directions.

3. Problem Description

The UAV swarm consists of K identical quadrotors, operating within a workspace W, where $W \subset R^m$ and $m = 2, 3$. Given an obstacle space P^{obs}, the free space is defined as $P^{free} = W \setminus P^{obs}$.

The state of UAV_i is represented as $x_i(t) = \left[p_i^T(t), v_i^T(t), \ldots, p_i^{(q-1)^T}(t)\right]^T$, where $x_i(t) \in \chi \subset R^{mq}$, $1 \le i \le K$, and χ is the state space. The state $x_i(t)$ includes the UAV's position and its derivatives up to order $q-1$. The q-th derivative of the position trajectory $p_i(t)$ is the control input $u_i(t)$, where $u_i(t) = p_i^{(q)}(t)$, and $u_i(t) \in U := [-u_{max}, u_{max}]^m$, with u_{max} being the maximum control input. The UAVs follow the system model:

$$\dot{x} = Ax + Bu \tag{1}$$

where

$$A = \begin{bmatrix} 0 & I_m & 0 & \cdots & 0 \\ 0 & 0 & I_m & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & I_m \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}_{qm \times qm}, B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ I_m \end{bmatrix}_{qm \times m}$$

UAVs are subject to kinematic constraints such as maximum velocity v_{max} and maximum acceleration a_{max}. The safe state space χ^{free} is defined as:

$$\chi^{free} = P^{free} \times [-v_{max}, v_{max}]^m \times [-a_{max}, a_{max}]^m \times \ldots \tag{2}$$

Additionally, UAV_i is modeled as a rigid body, with $S(x_i)$ representing the workspace occupied by the UAV_i in state $x_i(t)$. The UAV is geometrically modeled as a circle or sphere with its center at the UAV's centroid and radius r_c.

The UAV swarm motion planning problem is described as follows: Given each initial state $x_{i0} \in \chi^{free}$ for UAV_i, the goal is to reach the target region $X_i^G \subset \chi^{free}$. Within the given control input U, design the control input $u_i(t)$ for each UAV, adhering to the system model (1) and kino-dynamic constraints (2), to obtain the trajectory $T_i(t)$ such that $T_i(t_0) = x_{i0}$ and $T_i(t_{fi}) \in X_i^G$, where t_0 is the initial time of the swarm and t_{fi} is the arrival time for UAV_i. The trajectories $T_1, T_2, \ldots, T_K$ must be conflict-free at any time t, meaning:

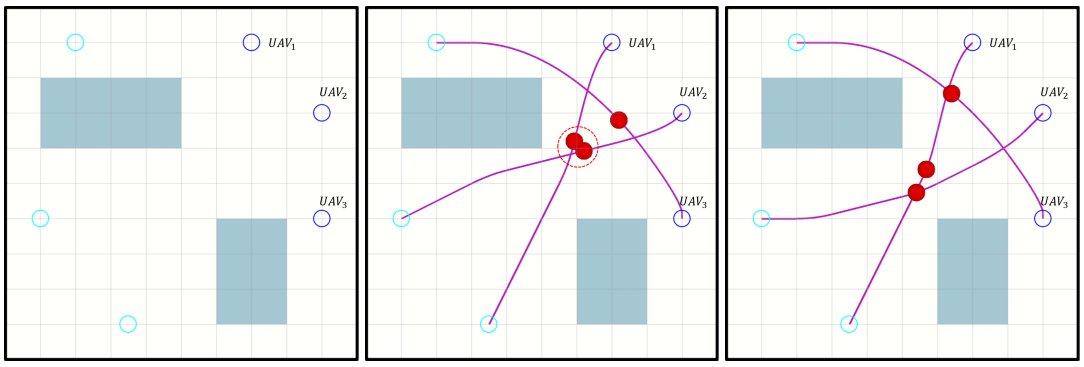
$$S(T_i(t)) \cap S(T_j(t)) = \emptyset, \quad \forall 1 \le i < j \le K \tag{3}$$

$$S(T_i(t)) \cap P^{obs} = \emptyset, \quad \forall 1 \le i \le K \tag{4}$$

 doi:10.20944/preprints202410.1212.v1

4. Algorithm

The SL-CBS algorithm builds upon the classic CBS framework, defining two conflict types: state conflicts and motion primitive conflicts. The low-level employs a State Lattice algorithm based on motion primitive search to effectively handle spatio-temporal constraints, while the high-level introduces a conflict detection method based on Sturm's theory and implements conflict resolution. Figure 1 clearly illustrates the planning process of the SL-CBS algorithm. Figure 1a depicts the initial state of a UAV swarm consisting of three UAVs, showcasing a map with obstacles, as well as the initial positions and target regions of the UAV swarm. Figure 1b presents the planning result of the first iteration of SL-CBS, where each UAV independently plans its trajectory without considering constraints. A collision occurs between the flight trajectories of UAV_1 and UAV_2. Figure 1c shows the planning result of the second iteration. After detecting the conflict in the higher-level algorithm, SL-CBS applies the corresponding constraints to the relevant UAVs and replans the trajectories. It can be observed that the three flight trajectories are now collision-free. The flight trajectories of UAV_1 and UAV_3 remain unchanged from Figure 1b, while UAV_2's trajectory is replanned with the necessary constraints applied to avoid the collision detected in Figure 1b.



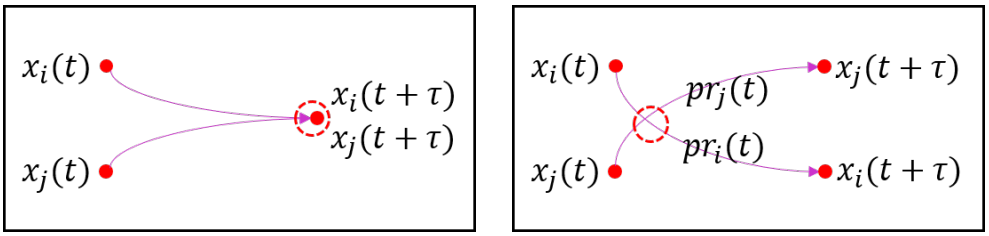
(**a**) initial state (**b**) the first iteration (**c**) the second iteration

Figure 1. Planning process of the SL-CBS algorithm. The black border represents the map boundary, the light green rectangles denote static obstacles, the blue hollow circles indicate the initial positions of the UAVs, and the cyan hollow circles represent the UAVs' target regions. The magenta curves depict the planned UAV flight trajectories, with the solid red circles on the curves indicating the UAV states.

4.1. Definition of Two Conflict Types

We comprehensively define two conflict types: state conflicts and motion primitive conflicts, addressing the limitation in [17] which only considers state conflicts. As shown in Figure 2, a state conflict, corresponding to a vertex conflict in the MAPF problem, is denoted as $\langle x_i(t), x_j(t), t+\tau \rangle$, indicating a conflict between the states $x_i(t+\tau)$ of UAV_i and $x_j(t+\tau)$ of UAV_j occurring at time $t+\tau$. A motion primitive conflict, also akin to an edge conflict in MAPF, is represented as $\left\langle x_i(t), x_j(t), pr_i(t), pr_j(t), t, t+\tau \right\rangle$, indicating a conflict between the motion primitive segment $pr_i(t)$ of UAV_i and $pr_j(t)$ of UAV_j over the interval $(t, t+\tau)$. The exact time and location of the conflict require precise calculation, but it occurs within $(t, t+\tau)$. A detailed description of motion primitive segments $pr_i(t)$ and $pr_j(t)$ is defined in Section 4.2.

 doi:10.20944/preprints202410.1212.v1



(**a**) state conflict (**b**) motion primitive conflict

Figure 2. Two types of conflicts. In (**a**), a state conflict occurs at the extended state at time $t+\tau$, while in (**b**), a motion primitive conflict occurs at some location within $(t, t+\tau)$.

After detecting conflicts and selecting one for resolution, the selected conflict is divided into specific constraints for the low-level algorithm to handle. A state conflict, $\langle x_i(t), x_j(t), t+\tau \rangle$, can be divided into constraints $\left\langle UAV_i, \overline{x_j(t)}, t+\tau \right\rangle$ and $\left\langle UAV_j, \overline{x_i(t)}, t+\tau \right\rangle$, where $\left\langle UAV_i, \overline{x_j(t)}, t+\tau \right\rangle$ prohibits UAV_i from reaching the vicinity of state $x_j(t)$ at time $t+\tau$, and similarly for $\left\langle UAV_j, \overline{x_i(t)}, t+\tau \right\rangle$. A motion primitive conflict, $\left\langle x_i(t), x_j(t), pr_i(t), pr_j(t), t, t+\tau \right\rangle$, can be divided into constraints $\left\langle UAV_i, \overline{pr_j(t)}, t, t+\tau \right\rangle$ and $\left\langle UAV_j, \overline{pr_i(t)}, t, t+\tau \right\rangle$, where $\left\langle UAV_i, \overline{pr_j(t)}, t, t+\tau \right\rangle$ prohibits UAV_i from conflicting with motion primitive $pr_j(t)$ during $(t, t+\tau)$, and similarly for $\left\langle UAV_j, \overline{pr_i(t)}, t, t+\tau \right\rangle$.

4.2. Improved State Lattice

State Lattice is a motion planning method based on motion primitive search, integrating the advantages of considering UAV dynamics and A* search [23]. The State Lattice algorithm first uniformly samples each dimension of the control input U over $[0, u_{max}]$ with a sampling number μ, generating a control input sample set $U_M := \{u_1, u_2, \ldots, u_M\}$, where $M = (2\mu+1)^3$. Given the initial state $x_0 := \left(p_0^T, v_0^T, a_0^T, \ldots\right)^T$, time interval $t > 0$, and control input u_m, the motion primitive $pr(t)$ is generated based on the UAV system model (1):

$$pr(t) = x(t) = e^{At}x_0 + \left[\int_0^t e^{A(t-\sigma)}Bd\sigma\right]u_m \tag{5}$$

The state transition equation $x(t)$ is represented by $pr(t)$ for clarity. According to [23], the motion primitive $pr(t)$ is a polynomial curve of order $2q-1$. Similarly, we design the motion primitive cost J as follows:

$$J = J_s + \rho T = \int_0^T \|u(t)\|^2 dt + \rho T \tag{6}$$

It is a linear combination of smoothness cost J_s and time cost T (Note: T here refers to time interval of motion primitive as described in [23]). Assuming the time interval $t = \tau$, the motion primitive cost is $\left(\|u_m\|^2 + \rho\right)\tau$, where ρ is the time cost weight. After constructing the motion primitive graph, the final trajectory is searched using the A* algorithm.

4.2.1. Emergency Stop Motion Primitive

In MAPF, the wait action is crucial, offering more possibilities in the motion space. When extended to UAV motion planning, an emergency stop motion primitive corresponding to the wait action is needed. Most existing work assumes that robots can stop instantaneously. However, we design

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 October 2024 doi:10.20944/preprints202410.1212.v1

an emergency stop motion primitive based on a maximum acceleration braking model, which is implemented through the following three steps: (1) Calculate the maximum braking time t_{stop}:

$$t_{stop} = \frac{\|v\|_\infty}{\lambda a_{max}} \tag{7}$$

Here, λ is a factor adjusting braking intensity to ensure completion within τ; (2) Calculate braking acceleration:

$$a_{stop} = -\frac{v}{t_{stop}} = -\frac{\lambda v a_{max}}{\|v\|_\infty} \tag{8}$$

(3) For each axis, calculate the emergency stop motion primitive pr_{stop} based on the braking model d_i:

$$d_i = \frac{v_i^2}{2a_{stop,i}} \tag{9}$$

where d_i is the displacement vector of braking distance in different dimensions. Taking a 2D workspace as an example, $i \in \{x, y\}$, as shown in Figure 3. The cost of the emergency stop motion primitive is designed as $J_{stop} = \rho\tau$, assuming no control input effect.

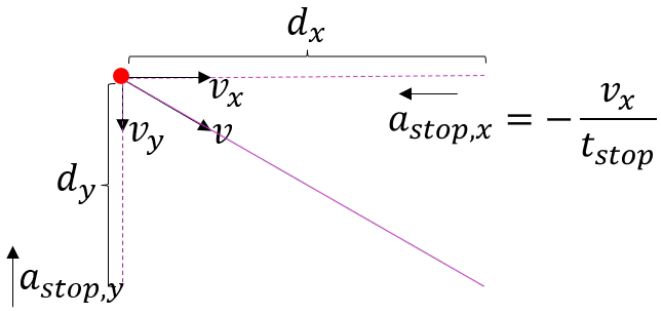


Figure 3. Emergency stop motion primitive model.

4.2.2. Spatio-Temporal Constraints

The high-level algorithm defines state and motion primitive conflicts, dividing them into specific state and motion primitive constraints. In the improved State Lattice algorithm, these constraints are processed to prevent corresponding conflicts. The legality detection process after motion primitive generation involves two steps: (1) Check if the constraint occurrence time overlaps with the current expanded state/motion primitive time. If not, skip the check; otherwise, proceed to (2); (2) For state constraints, check if the current state conflicts with the constrained state. For motion primitive constraints, check if the current motion primitive conflicts with the constrained motion primitive. The specific detection method is detailed in the conflict detection scheme in Section 4.3.

The improved State Lattice pseudo-code is shown in Algorithm 1, with major improvements marked with light yellow areas, while the rest remains consistent with the original algorithm. In line 10, the *getNeighbors** function includes the design of emergency stop motion primitives, and line 18-*20 adds a new check for potential violations of high-level constraints with *chkCons*().

 doi:10.20944/preprints202410.1212.v1

Algorithm 1 SLPlanner*(Improved SLPlanner)

Input: x_0, X^G, constraints
Output: a safe and feasible trajectory T

root $\leftarrow (x_0)$; root.$g \leftarrow 0$; root.$h \leftarrow h(x_0)$;
OpenList.*push*(root);
while OpenList *is not Empty* **do**
  currNode $\leftarrow$ OpenList.*top*();
  OpenList.*pop*();
  if currNode *is near* X^G **then**
    $T \leftarrow$ *traceTrajectory*(currNode)
    return true;
  end if
  neighbors $\leftarrow$ *getNeighbors**(currNode);
  for neighbor *in* neighbors **do**
    nextNode $\leftarrow$ (neighbor.state, neighbor.pr);
    nextNode.$g \leftarrow$ currNode.g + *cost*(neighbor);
    nextNode.$h \leftarrow h$(neighbor.state);
    if *chkDyn*(nextNode) || *chkColl*(nextNode) **then**
      continue;
    end if
    if *chkCons*(nextNode, constraints) **then**
      continue;
    end if
    if nextNode *is not in* OpenList **then**
      OpenList.*push*(nextNode);
    else
      if nextNode.$g <$ OpenList[targetIndex].g **then**
        OpenList.*update*(nextNode);
      end if
    end if
  end for
end while
return false;

4.3. Conflict Detection Method Based on Sturm's Theory

Detecting state and motion primitive conflicts is a frequent operation that directly impacts the efficiency of swarm motion planning. State conflict detection can be performed directly using the geometric model and pose information of the UAVs. Given that UAV_i is in state $x_i(t)$ with position $p_i(t)$ at time t, and UAV_j is in state $x_j(t)$ with position $p_j(t)$ at the same time, a state conflict between UAV_i and UAV_j at time t is determined by the following condition:

$$\left\| p_i(t) - p_j(t) \right\|_2 < 2r_c \tag{10}$$

Detecting motion primitive conflicts is more complex. The motion primitive of UAV_i is $pr_i(t)$, and for UAV_j, it is $pr_j(t)$, with the time interval for motion primitives being $(t, t + \Delta t)$. Due to the presence of emergency stop motion primitives, the time step Δt is less than τ. For simplicity, we normalize the process and check whether the motion primitives $pr_i(t)$ and $pr_j(t)$ conflict over the interval $t \in (0, \Delta t)$ using the following condition:

$$\left\| pr_i(t) - pr_j(t) \right\|_2 < 2r_c,\ t \in (0, \Delta t) \tag{11}$$

[19] indicates that sampling-based detection in the time dimension is inefficient, and its completeness depends on sampling precision. Solving the intersection of two motion primitives analytically from Formula (11) can lead to numerical instability, and finding the roots of high-order polynomials is challenging. Inspired by the method in [24] using Sturm's theory for polynomial constraint satisfaction

 doi:10.20944/preprints202410.1212.v1

problem, which determines the number of roots without calculating their exact values, we apply this approach to conflict detection process because it perfectly suits our needs. Using Sturm's theory, we detect motion primitive conflicts by first constructing a multivariable polynomial function $G(t)$:

$$G(t) = \left\| pr_i(t) - pr_j(t) \right\|^2 - 4r_c^2,\ t \in (0, \Delta t) \tag{12}$$

If $G(t)$ has roots within the interval $t \in (0, \Delta t)$, a conflict occurs; otherwise, there is no conflict. According to Sturm's theory, determining the existence of roots involves constructing the Sturm sequence of $G(t)$ and calculating the difference in the number of sign changes at the endpoints of the interval $(0, \Delta t)$. For detailed information on constructing the Sturm sequence and calculating sign changes at interval endpoints, refer to [24].

4.4. SL-CBS+ID

After detecting all conflicts within the UAV swarm, a conflict graph is constructed for the UAVs. As shown in Figure 4, we observe that conflicts between UAV_1, UAV_2, UAV_3, and UAV_5, as well as those between UAV_4, UAV_6 and UAV_7 can be handled in parallel, thus reducing iteration numbers of SL-CBS algorithm. The Independence Detection (ID) technique addresses this issue by determining that if the optimal solutions for each group of agents do not conflict with one another, the groups are considered independent. We integrate the ID concept from [25] into the SL-CBS algorithm as follows: (1) Construct a conflict graph from the trajectory results of the UAV swarm; (2) Group the UAV swarm using Depth-First Search(DFS); (3) Solve each group in parallel using the SL-CBS algorithm; (4) Check if there are any conflicts in the trajectory results of the entire UAV swarm. If conflicts exist, return to step (1); otherwise, terminate the algorithm. To prevent ID from entering a loop of iterations, we stipulate that if the ID iteration exceeds a specified value, all agents are grouped together. Overall, with ID acceleration, the optimality and completeness of the original SL-CBS algorithm are maintained, while enabling parallel computation to enhance performance.

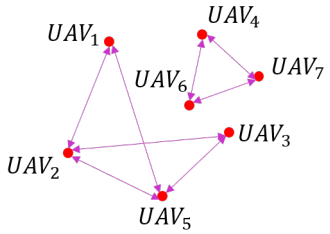


Figure 4. UAV swarm conflict graph. The magenta double-headed arrows represent the existence of conflicts between the trajectories of the UAVs.

The complete pseudo-code SL-CBS+ID algorithm for is shown in Algorithm 2. Lines 5–25 outline the SL-CBS algorithm process, while the remaining lines implement the ID. In line 2, a conflict graph is constructed based on the initial solution from line 1. Line 4 involves grouping the UAV swarm based on the conflict graph. Within lines 5–19, line 6 constructs the root node of the CT tree based on the initial states and target regions, where the CT tree is maintained by the priority queue CBSPriorityQueue. In line 9, a greedy search retrieves the current optimal node from the CBSPriorityQueue. Lines 10–14 assess whether the current node has any conflicts; if none are present, the solution is returned. Otherwise, line 15 extracts the earliest occurring conflict for processing. Subsequently, line 16 partitions the relevant constraints for the current conflict. Lines 17–23 handle these constraints, with line 18 constructing new child nodes that inherit the constraints of the parent node. Line 19 adds new constraints, and line 20 resolves the current conflict through replanning based on the improved State Lattice. Finally, in line 22, the newly created child nodes are added to the CBSPriorityQueue.

 doi:10.20944/preprints202410.1212.v1

Algorithm 2 SL-CBS+ID

Input: K, $\{x_{i0}\}$, $\{X_i^G\}$
Output: multiple collision-free trajectories $\{T_1, T_2, \ldots, T_K\}$
$\{T_1, T_2, \ldots, T_K\} \leftarrow$ **SLPlanner**$^*(x_{i0}, X_i^G)$;
conflicts $\leftarrow findAllConflicts(\{T_1, T_2, \ldots, T_K\})$;
while conflicts.$size() > 0$ **do**
  groups $\leftarrow divideGroups$(conflicts);
  for group *in* groups **do**
    root $\leftarrow generateRootNode$(group, $\{x_{i0}\}, \{X_i^G\}$);
    CBSPriorityQueue.$push$(root);
    while CBSPriorityQueue *is not Empty* **do**
      currNode $\leftarrow$ CBSPriorityQueue.$top()$;
      CBSPriorityQueue.$pop()$;
      if $validateSolutions$(currNode.sols) **then**
        $\{T_1, T_2, \ldots, T_K\} \leftarrow$ currNode.sols;
        break;
      end if
      conflict $\leftarrow chooseConflict$(currNode);
      constraints $\leftarrow divideConstraints$(conflict);
      for constraint *in* constraints **do**
        childNode $\leftarrow$ currNode;
        childNode.$addConstraints$(constraint);
        sol $\leftarrow$ **SLPlanner**$^*(x_{i0}, X_i^G$, childNode.cons);
        childNode.$update$(sol);
        CBSPriorityQueue.$push$(childNode);
      end for
    end while
  end for
  conflicts $\leftarrow findAllConflicts(\{T_1, T_2, \ldots, T_K\})$;
end while
return $\{T_1, T_2, \ldots, T_K\}$;

5. Experiment

Our proposed SL-CBS+ID algorithm runs on a system equipped with an Intel Core i7-8700@3.2GHz CPU and 16GB RAM, utilizing Ubuntu and ROS environments. The complete C++ project code is open-sourced at https://github.com/ucas-zihaowang/SL-CBS.git. The complexity of the environment, along with the initial and target states of the UAV swarm and kino-dynamic models, directly affect the results of swarm motion planning. We strictly establish a unified benchmark and consistent UAV kino-dynamic models to ensure the accuracy and fairness of the experimental results. Ultimately, we conduct three types of experiments: classic scenarios testing, comparison with baseline algorithms, and analysis of the algorithm's performance boundaries.

5.1. Test on Classical Scenarios

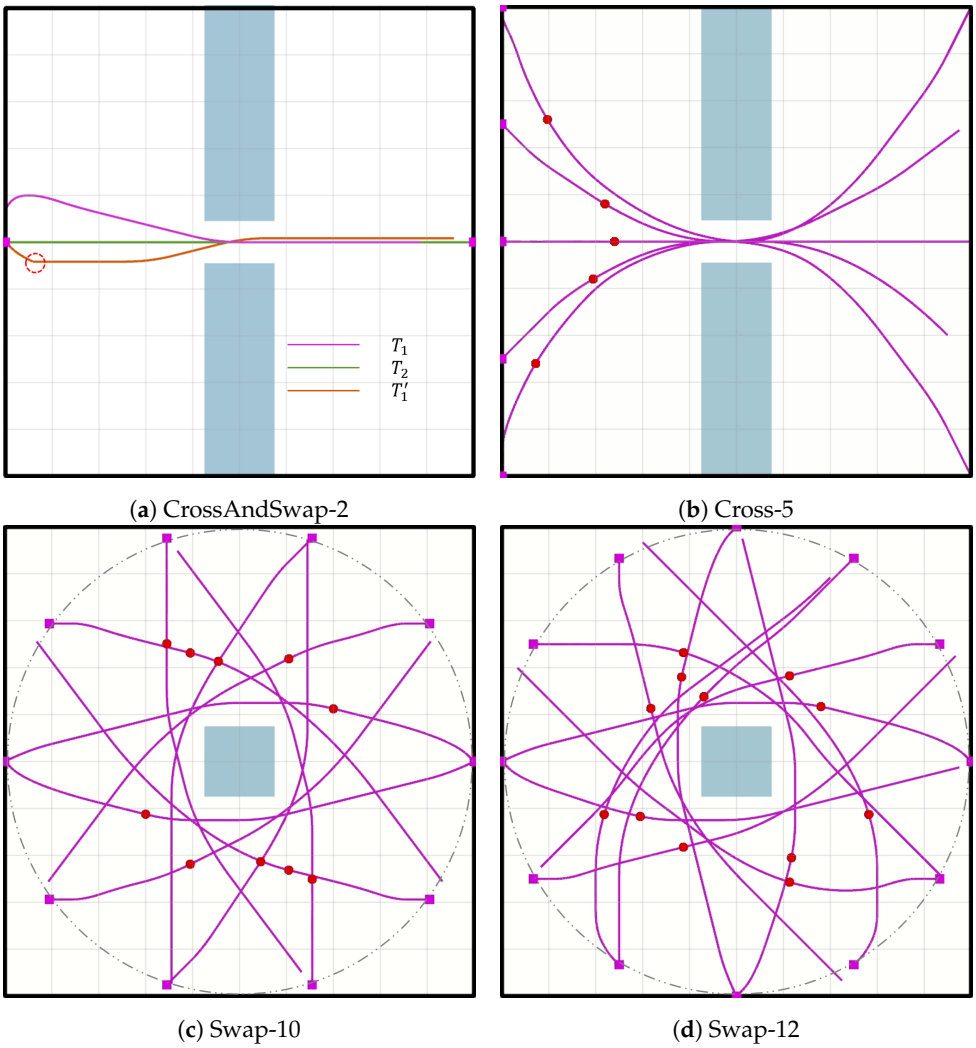
5.1.1. Setup

The Tunnel Configuration and Star Configuration are two classic test scenarios [2]. Figure 5a,b illustrate the Tunnel Configuration, while Figure 5c,d show the Star Configuration, both measuring 10m by 10m. The light green rectangular areas represent static obstacles. We conduct three types of tests in these scenarios: CrossAndSwap, Cross, and Swap. The UAV's maximum speed v_{max} is 2m/s, with acceleration-based control and a maximum control input u_{max} of 1. The control space sampling number μ is set to 1, the time interval τ is 0.5s, and the time weight of the motion primitive cost function ρ is 10. Additionally, the UAV's geometric model is a circle with a radius r_c of 0.1m.

We select sequential planning and decentralized planning from [2] as baseline algorithms. Sequential planning is a centralized algorithm that plans flight trajectories for each UAV in order of

 doi:10.20944/preprints202410.1212.v1

priority, treating the trajectories of previously planned UAVs as dynamic obstacles. In decentralized planning, each UAV plans independently, treating others as dynamic obstacles, and iteratively replanning based on a receding horizon control. Both algorithms maintain consistent UAV models and parameters, using computation time, trajectory flight time, and total trajectory cost as evaluation metrics.



(**a**) CrossAndSwap-2 (**b**) Cross-5

(**c**) Swap-10 (**d**) Swap-12

Figure 5. Flight trajectories of UAV swarm in classic scenario test instances. The black border represents the map boundary, while the light green areas indicate static obstacles. The magenta curves depict the flight trajectories of the UAVs. In Figure 5a, the green and orange curves also represent UAV flight trajectories. In Figure 5b–d, the solid red circles indicate the flight states of the UAV at intermediate time.

5.1.2. Result

In Figure 5a, two UAVs cross a narrow gap to reach each other's positions, where only one UAV can pass through at a time. T_1 is the flight trajectory of UAV_1 without using an emergency stop, while $T_1{}'$ includes the emergency stop motion primitive, and T_2 is the trajectory of UAV_2. The flight trajectory $T_1{}'$ shows a noticeable emergency stop at the red circle, with a significantly reduced detour. According to Table 1, using the emergency stop motion primitive reduces both flight time and total trajectory cost. However, it should be noted that the addition of emergency stop motion primitives increases the search space, leading to longer computation time. Typically, the increase in computation time is not substantial, but our CrossAndSwap-2 test represents an extreme case.

 doi:10.20944/preprints202410.1212.v1

In Figure 5b–d, we conduct tests for Cross-5(5 UAVs crossing a gap), Swap-10(10 UAVs swapping positions), and Swap-12(12 UAVs swapping positions), respectively. According to Table 2, our proposed SL-CBS shows a great reduction in both trajectory flight time and total trajectory cost across all test instances compared to sequential planning and decentralized planning. This is because sequential planning sacrifices trajectory quality to obtain suboptimal solutions, while decentralized planning focuses on local optima for each UAV, thereby losing the global optimal solution. In terms of computation time, sequential planning is highly efficient as it plans the trajectory for each UAV only once, but it is prone to unsolvable situations in complex environments. Decentralized planning, on the other hand, requires iterative computation of each UAV's flight trajectory from its current state to the target region, resulting in longer computation time. As shown in Table 2, SL-CBS generally takes longer computation time than sequential planning in most test instances but performs better than decentralized planning. Additionally, we observe that in the Swap-12 test, due to the dispersed distribution of conflicts, the use of ID technique greatly reduces computation time by grouping.

Table 1. Evaluation metrics results of CrossAndSwap-2.

Planner	Cross and Swap-2		
	Computation Time	**Flight Time**	**Total Trajectory Cost**
SL-CBS (no pr_{stop})	706ms	13.5s	141
SL-CBS (pr_{stop})	1665ms	13s	136.5

Table 2. Evaluation metrics results of classic scenario test instances.

Planner	Cross-5			Swap-10			Swap-12		
	Comp. Time	**Flight Time**	**Total Traj. Cost**	**Comp. Time**	**Flight Time**	**Total Traj. Cost**	**Comp. Time**	**Flight Time**	**Total Traj. Cost**
Seq Plan [2]	1384ms	35s	381	953ms	58.5s	628	1304ms	70s	758
Dec Plan [2]	4861ms	43.04s	465.46	2464ms	60.45s	660.65	2684ms	74.13s	808.93
SL-CBS	1211ms	34s	371.5	2128ms	58s	621	27533ms	68s	735
SL-CBS+ID	1220ms	34s	371.5	2294ms	58s	621	2653ms	68s	736

5.2. Compared to Baseline Algorithms

5.2.1. Setup

Most benchmarks for MAPF are based on 4/8-connected grid maps, which cannot be directly applied to continuous space motion planning for UAV swarm. To address this, we design a continuous map with dimensions of 5m by 5m, featuring 20 randomly generated circular obstacles with a radius of 0.2m, resulting in an obstacle density of approximately 10%. The initial and target positions of the UAV swarm must satisfy the following requirements: (1) The initial and target positions of the UAVs must not collide with each other or with the obstacle regions; (2) The Euclidean distance between each UAV's initial and target position is greater than 1/4 of the map width and less than 3/4 of the map width. Here, the maximum speed of the UAVs v_{max} is 0.5 m/s, with acceleration-based control and a maximum control input u_{max} of 2, Additionally, the geometric model of the UAVs is a circle with a radius r_c of 0.15m.

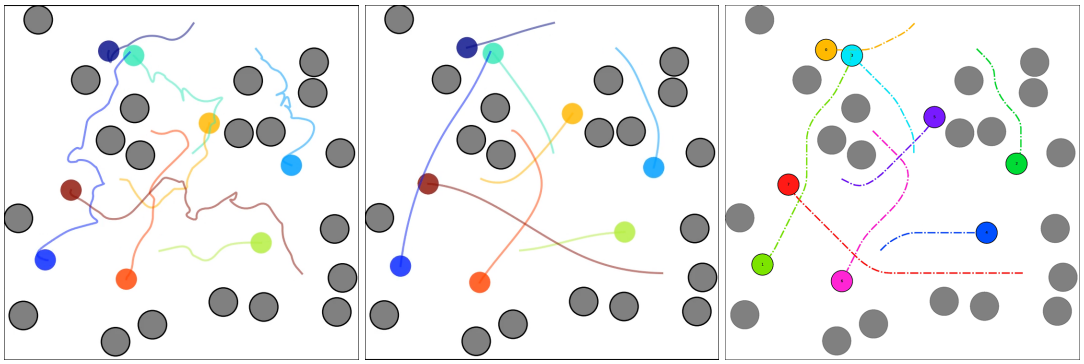
 doi:10.20944/preprints202410.1212.v1

We select the latest algorithms suitable for UAVs, K-CBS [19] and db-CBS [21], which share the same technical approach as SL-CBS, as baseline algorithms. The control interval length is crucial for a fair comparison. K-CBS employs a sampling-based low-level algorithm with control interval lengths varying between $[0.1, 1]$. The db-CBS low-level algorithm is based on db-A*, with pre-computed motion primitives consisting of states and actions at time intervals of 0.1s. Analysis of db-CBS local motion primitives shows a basic time intervals length greater than 0.5s. Therefore, we set the time interval τ for SL-CBS to 0.5s. Given the high obstacle density and complex environment of our benchmark, the SL-CBS control input sampling number μ is set to 4.

We conduct batch tests with swarm sizes $K = 2, 4, 6, 8, 10$, testing each scenario with 10 trials, with a time limit of 300s. Consistent with the db-CBS project [21], we use success rate, average computation time, and average trajectory flight time as evaluation metrics.

5.2.2. Result

Figure 6 illustrates the flight trajectories of SL-CBS and its baseline algorithms for a swarm size of $K = 8$. The trajectories planned by K-CBS are relatively rugged and contain numerous bends, primarily because its low-level search relies on a sampling algorithm and lacks back-end optimization. Although the low-level of db-CBS uses db-A* to search discontinuous trajectories, these become smooth and continuous after trajectory optimization. Our SL-CBS employs an improved State Lattice at the low level, which considers complete dynamics, resulting in smooth and continuous trajectories.



(**a**) K-CBS (**b**) db-CBS (**c**) SL-CBS

Figure 6. Flight trajectories planned by K-CBS, db-CBS and SL-CBS for swarm size $K = 8$. The black border represents the map boundaries, while the 20 gray circles indicate obstacle regions. The solid circles in various colors depict the current states of the UAV swarm, and the curves in different colors represent the trajectories already flown by the UAV swarm.

Table 3 presents the batch test results for SL-CBS and baseline algorithms across different swarm sizes. Due to the inefficient search of the sampling algorithm and the ineffective conflict interval detection method, K-CBS can only solve most instances for swarm size of 2-6, consistent with the experimental results in [19]. The db-CBS low-level, utilizing pre-computed motion primitives for search, can solve most instances for swarm size of 2-10. However, because it includes an additional layer of trajectory optimization and its conflict detection method in the high level is less effective than that of SL-CBS, its computation time is generally longer than SL-CBS. Only for a swarm size of 10 does SL-CBS take longer time to compute than db-CBS, as our method for calculating average computation time includes all successfully solved instances. SL-CBS, with its optimal two-layer search, substantially outperforms K-CBS and db-CBS in terms of trajectory flight time. Therefore, based on the test results and the above analysis, SL-CBS demonstrates clear improvements over K-CBS and db-CBS in success rate, computation time, and flight time.

 doi:10.20944/preprints202410.1212.v1

Table 3. Evaluation metrics results of K-CBS, db-CBS and SL-CBS.

Swarm Size	K-CBS			db-CBS			SL-CBS		
	Success Rate	Average Comp. Time	Average Flight Time	Success Rate	Average Comp. Time	Average Flight Time	Success Rate	Average Comp. Time	Average Flight Time
$K = 2$	1	2.62s	28.66s	1	0.38s	11.61s	1	0.24s	9.5s
$K = 4$	1	21.89s	50.46s	1	4.77s	24.89s	1	0.26s	20.65s
$K = 6$	0.7	55.3s	67.66s	0.9	11.48s	38.73s	1	0.62s	30.6s
$K = 8$	0.5	116.13s	88.94s	0.8	36.05s	53.69s	1	10.02s	41.5s
$K = 10$	0	-	-	0.8	43.91s	67.93s	0.9	63.98s	52.38s

5.3. Performance Boundaries

5.3.1. Setup

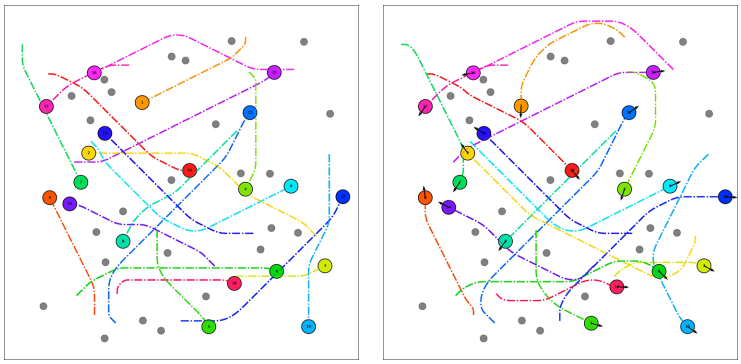
Similar to CL-CBS [17] used in the car-like robots, we conduct batch testing on SL-CBS to analyze its performance boundaries across different swarm sizes. Here, the map size is 50m by 50m, with 25 randomly generated circular obstacles, each with a radius of 0.5m. The initial and target positions of the swarm are randomly generated, consistent with the requirements in Section 5.2. The maximum speed of the UAVs is 2m/s, controlled by acceleration, with a maximum control input u_{max} of 1. Additionally, we introduce yaw control to achieve full state-space planning for the UAVs, with a maximum yaw of 0.27 radian and a maximum yaw control input of 0.5. We also set the control space sampling number μ to 1 and the time interval τ to 1. The geometric model of the UAV is a circle with a radius r_c of 1m.

For each randomly generated scenario, we test it with 30 trials, with the computation time limited to 90s. We conduct batch tests using four algorithm configurations: SL-CBS(Without yaw), SL-CBS+ID(Without yaw), SL-CBS(With yaw), and SL-CBS+ID(With yaw), across swarm size of 5, 10, 15, and 20. The evaluation metrics are success rate, average computation time, average total trajectory flight time, and average makespan, where makespan is defined as the maximum flight time within the UAV swarm.

5.3.2. Result

Figure 7 shows the planned trajectories for SL-CBS with and without yaw for a swarm size of 20. It demonstrates the algorithm's planning capability to solve large-scale, collision-free trajectories in dense environments. Figure 8 illustrates the relationship and trends between various evaluation metrics and swarm size. According to Figure 8a, when the swarm size is 10 or less, SL-CBS+ID can solve all instances, while SL-CBS solves over 90% of them. As the swarm size exceeds 10, the performance of SL-CBS declines, which aligns with the trend in Figure 8b where computation time increases linearly with swarm size. When the swarm size is 20, SL-CBS+ID(Without yaw) still maintains a success rate of over 50%. From Figure 8b, SL-CBS can solve problems within 5s for swarm sizes of 10 or less, and requires 10 seconds for a swarm size of 15. It is noteworthy that for SL-CBS+ID(With yaw), the computation time decreases at a swarm size of 20 due to its lower success rate, which includes only a limited sample of successful instances.

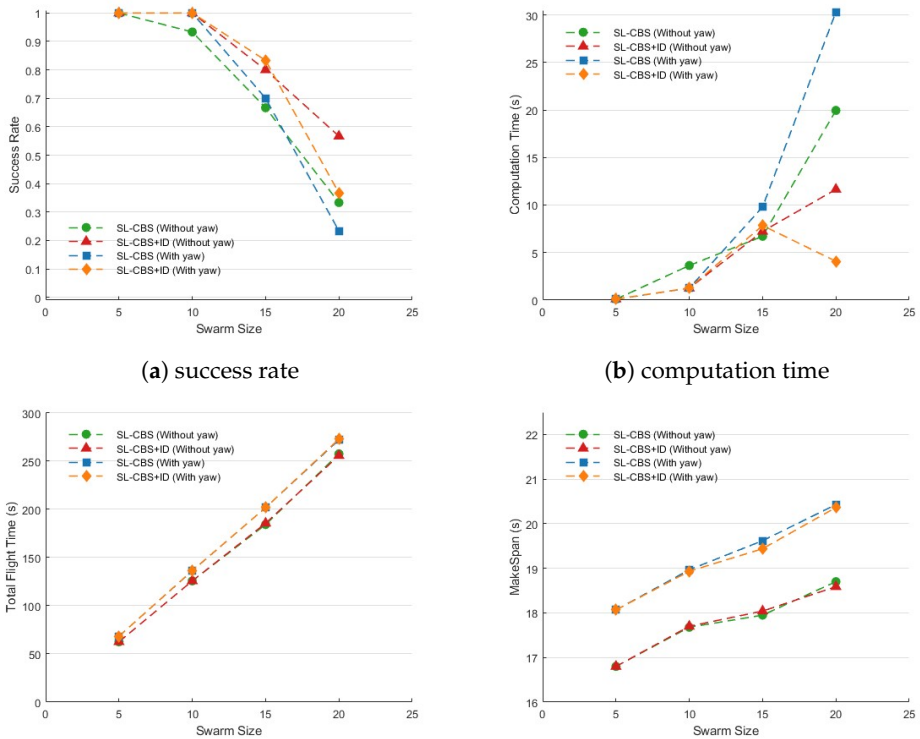
Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 October 2024 doi:10.20944/preprints202410.1212.v1



(**a**) $K = 20$ (Without yaw) (**b**) $K = 20$ (With yaw)

Figure 7. Flight trajectories planned by SL-CBS for swarm size $K = 20$. The black border represents the map boundary, while the 25 gray circles indicate obstacle regions. The solid circles in various colors represent the current states of the UAV swarm, with black arrows on the circles indicating the yaw direction of each UAV. The curves in different colors illustrate the trajectories already traveled by the UAV swarm.

Figure 8c,d reveal that the total trajectory flight time and makespan cost curves for SL-CBS and SL-CBS+ID nearly overlap, further indicating that our ID technique does not affect actual trajectory results. Combined with Figure 8a,b, SL-CBS+ID consistently outperforms SL-CBS. Particularly for larger swarm size, when the swarm size is 20, the success rate improves by approximately 13.4%, and computation time is reduced by about 8.3s.



(**a**) success rate (**b**) computation time

(**c**) total flight time (**d**) makespan

Figure 8. Evaluation metrics results of SL-CBS under different swarm sizes.

6. Conclusions

We extend Conflict-Based Search from the MAPF domain to UAV swarm motion planning, formally defining the swarm motion planning problem and proposing the optimal and efficient SL-CBS+ID algorithm.

 doi:10.20944/preprints202410.1212.v1

SL-CBS+ID eliminates the need for motion constraints and physical assumptions, making it fully suitable for autonomous UAV swarm flight. It features a two-layer search algorithm: the low-level uses an improved State Lattice algorithm tailored for UAVs, incorporating emergency stop primitives and effectively handling spatio-temporal constraints. The high level considers complete state and motion primitive conflicts, introducing a linear time complexity motion conflict detection method based on Sturm's theory. Additionally, our modified ID is applied to enable parallel conflict processing.

We conduct tests on classic scenarios like swap and cross to validate SL-CBS+ID's planning capabilities. Compared to SOTA algorithms, our SL-CBS demonstrates great advantages in success rate, computation time, and trajectory flight time. Finally, we perform batch testing of SL-CBS+ID to analyze its performance boundaries.

In the future, we will explore the application of CBS enhancement techniques, such as conflict classification and high-level heuristics, to SL-CBS, aiming to improve its solving capabilities without compromising its optimality and completeness.

References

1. Chen, J.; Li, J.; Fan, C.; Williams, B.C. Scalable and safe multi-agent motion planning with nonlinear dynamics and bounded disturbances. Proceedings of the AAAI conference on artificial intelligence, 2021, Vol. 35, pp. 11237–11245.
2. Liu, S.; Mohta, K.; Atanasov, N.; Kumar, V. Towards search-based motion planning for micro aerial vehicles. *arXiv preprint arXiv:1810.03071* **2018**.
3. via Prioritized, N.h.M.R. Efficient Trajectory Planning for Multiple Non-holonomic Mobile Robots via Prioritized Trajectory Optimization **2020**.
4. Yu, J.; LaValle, S. Structure and intractability of optimal multi-robot path planning on graphs. Proceedings of the AAAI Conference on Artificial Intelligence, 2013, Vol. 27, pp. 1443–1449.
5. Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T.; Uras, T.; Xu, H.; Tovey, C.; Sharon, G. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515* **2017**.
6. Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N.R. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence* **2015**, *219*, 40–66.
7. Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; others. Multi-agent pathfinding: Definitions, variants, and benchmarks. Proceedings of the International Symposium on Combinatorial Search, 2019, Vol. 10, pp. 151–158.
8. Felner, A.; Stern, R.; Shimony, S.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; Surynek, P. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. Proceedings of the International Symposium on Combinatorial Search, 2017, Vol. 8, pp. 29–37.
9. Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Betzalel, O.; Tolpin, D.; Shimony, E. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. Proceedings of the International Symposium on Combinatorial Search, 2015, Vol. 6, pp. 223–225.
10. Boyarski, E.; Felner, A.; Sharon, G.; Stern, R. Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. Proceedings of the International Conference on Automated Planning and Scheduling, 2015, Vol. 25, pp. 47–51.
11. Li, J.; Harabor, D.; Stuckey, P.J.; Ma, H.; Koenig, S. Symmetry-breaking constraints for grid-based multi-agent path finding. Proceedings of the AAAI conference on artificial intelligence, 2019, Vol. 33, pp. 6087–6095.
12. Li, J.; Felner, A.; Boyarski, E.; Ma, H.; Koenig, S. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. IJCAI, 2019, Vol. 2019, pp. 442–449.
13. Andreychuk, A.; Yakovlev, K.; Surynek, P.; Atzmon, D.; Stern, R. Multi-agent pathfinding with continuous time. *Artificial Intelligence* **2022**, *305*, 103662.
14. Andreychuk, A.; Yakovlev, K.; Boyarski, E.; Stern, R. Improving continuous-time conflict based search. Proceedings of the AAAI Conference on Artificial Intelligence, 2021, Vol. 35, pp. 11220–11227.

 doi:10.20944/preprints202410.1212.v1

15. Park, C.; Lee, S.; Yang, H.; Shin, D.; Kang, S.; Kim, Y. Conflict-Based Search with Partitioned Groups of Agents for Real-World Scenarios. 2023 20th International Conference on Ubiquitous Robots (UR). IEEE, 2023, pp. 986–992.
16. Solis, I.; Motes, J.; Sandström, R.; Amato, N.M. Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics and Automation Letters* **2021**, *6*, 4608–4615.
17. Wen, L.; Liu, Y.; Li, H. CL-MAPF: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints. *Robotics and Autonomous Systems* **2022**, *150*, 103997.
18. Liu, W.; Wang, J.; Zhang, K.; Yu, H.; Zheng, Z.; Lu, G. HG-CBS Planner: Heuristic Group-based Motion Planning for Multi-robot. 2023 IEEE 18th Conference on Industrial Electronics and Applications (ICIEA). IEEE, 2023, pp. 687–692.
19. Kottinger, J.; Almagor, S.; Lahijanian, M. Conflict-based search for multi-robot motion planning with kinodynamic constraints. 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2022, pp. 13494–13499.
20. Tajbakhsh, A.; Biegler, L.T.; Johnson, A.M. Conflict-based model predictive control for scalable multi-robot motion planning. 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024, pp. 14562–14568.
21. Moldagalieva, A.; Ortiz-Haro, J.; Toussaint, M.; Hönig, W. db-cbs: Discontinuity-bounded conflict-based search for multi-robot kinodynamic motion planning. 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024, pp. 14569–14575.
22. Hönig, W.; Ortiz-Haro, J.; Toussaint, M. db-A*: Discontinuity-bounded Search for Kinodynamic Mobile Robot Motion Planning. 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2022, pp. 13540–13547.
23. Liu, S.; Atanasov, N.; Mohta, K.; Kumar, V. Search-based motion planning for quadrotors using linear quadratic minimum time control. 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017, pp. 2872–2879.
24. Wang, Z.; Zhou, X.; Xu, C.; Chu, J.; Gao, F. Alternating minimization based trajectory generation for quadrotor aggressive flight. *IEEE Robotics and Automation Letters* **2020**, *5*, 4836–4843.
25. Standley, T. Finding optimal solutions to cooperative pathfinding problems. Proceedings of the AAAI conference on artificial intelligence, 2010, Vol. 24, pp. 173–178.