2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Conflict-Based Search as a Protocol: A Multi-Agent Motion Planning Protocol for Heterogeneous Agents, Solvers, and Independent Tasks

Rishi Veerapaneni[1], Alvin Tang[†1], Haodong He[†2], Sophia Zhao[†1], Viraj Shah[†1], Yidai Cen[†1], Ziteng Ji[†3], Gabriel Olin[‡1], Jon Arrizabalaga[‡1], Yorai Shaoul[‡1], Jiaoyang Li[1], Maxim Likhachev[1]

***Abstract*— Imagine the future construction site, hospital, or office with dozens of robots bought from different manufacturers. How can we enable these different robots to effectively move in a shared environment, given that each robot may have its own independent motion planning system? This work shows how we can get efficient collision-free movements between algorithmically heterogeneous agents by using Conflict-Based Search (Sharon et al. 2015) as a protocol. At its core, the CBS Protocol requires one specific single-agent motion planning API; finding a collision-free path that satisfies certain space-time constraints. Given such an API, CBS uses a central planner to find collision-free paths - independent of how the API is implemented. We demonstrate how this protocol enables multi-agent motion planning for a heterogeneous team of agents completing independent tasks with a variety of single-agent planners including: Heuristic Search (e.g., A*), Sampling Based Search (e.g., RRT), Optimization (e.g., Direct Collocation), Diffusion, and Reinforcement Learning.**

I. Introduction

The rapid progress of robotics technology is lowering robots' costs while enhancing capabilities, leading to a future in which multi-agent robotic systems will be ubiquitous. Such systems may operate across a wide range of domains, including construction, manufacturing, office environments, and even sophisticated household settings. Importantly, these multi-agent teams are unlikely to be homogeneous; rather, they will consist of robots produced by different manufacturers, each performing independent tasks. This raises a fundamental question: how can heterogeneous robots, each potentially equipped with its own motion planning system, coordinate their movements in a shared environment?

In particular, such robots may:

1) Exhibit different embodiments and kinodynamic constraints,
2) Pursue distinct and independent tasks, and
3) Be developed by different companies, each with proprietary motion planning solvers.

While this may seem far-fetched, it is already beginning to appear in practice. Current automated warehouses and manufacturing facilities often integrate diverse platforms such as forklifts, trucks, and mobile carts, which must all operate within the same workspace. However, it remains unclear how to enable such heterogeneous robots to navigate and coordinate effectively in a common environment.

To that end, our main contribution is demonstrating how we can use Conflict-Based Search as a protocol to enable efficient collision-free movement. Conflict-Based Search (CBS) [2] is a foundational method from Multi-Agent Path Finding [3] that finds collision-free multi-agent paths. At its core, we view CBS as defining a protocol that requires one specific single-agent motion planning API; finding a path that avoids space-time locations. Given such an API, CBS uses a central planner to resolve collision between agents - independent of how the API is implemented. Thus, "CBS as a protocol" enables collision-free multi-agent motion planning using different single-agent planners. We demonstrate the effectiveness of this protocol by showing how we can effectively plan for a heterogeneous team with a variety of single-agent planners including: Heuristic Search, Sampling Based Search, Optimization, Diffusion, and Reinforcement Learning. Additionally, the CBS Protocol enables each agent to complete different independent tasks (e.g., coverage, surveillance) instead of just start-goal tasks.

A. Contributions

Taking a step back, to our knowledge the problem of Multi-Agent Motion Planning (MAMP) with algorithmically heterogeneous solvers has not been explored by the MAMP community (discussed more in Section II-B). Thus, our first contribution is to bring attention to this problem which we term Algorithmically Heterogeneous MAMP (AH-MAMP).

Second, the CBS Protocol bridges a gap between single-agent motion planning, multi-agent motion planning, and Multi-Agent Path Finding (MAPF). In particular, our CBS Protocol solution takes existing technology (i.e., existing single-agent solvers and existing MAPF techniques) and shows how they can be put together in a novel way. Thus, our paper is a proof-of-concept paper whose primary contributions are shaped by the reader's context:

(1) Readers focused on single-agent motion planning but not MAMP; For these readers, the key contribution of this paper is showing how we can use their single-agent motion planners in multi-agent systems through the CBS Protocol via defining one single API. The hope is that the protocol more easily enables single-agent motion planning researchers to incorporate their planners into multi-agent systems.

(2) Readers working on real-world MAMP but are unfamiliar with CBS, or only familiar with CBS in its classical

This work was partially supported by National Science Foundation (NSF) grant #2328671, by NSF Graduate Research Fellowship Program grant #DGE2140739, and a gift from Amazon. This research partially used Bridges-2 at PSC [1] through the ACCESS program which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #2138296.

†, ‡ Equal authorship respectively, sorted alphabetically
[1] Carnegie Mellon University, USA
[2] Tongji University, China
[3] UC Berkeley, USA
Corresponding author: vrishi@cmu.edu
Webpage: https://rishi-v.github.io/CBS-Protocol/

(gridworld) context; For these readers, the primary contribution is showing how the CBS Protocol extends beyond gridworld and can incorporate a range of diverse motion planners (e.g., optimization) for kinodynamic motion planning.

(3) Readers familiar with CBS and its extensions past non-gridworld; For these readers, the main contribution is the protocol formulation using heterogeneous solvers at once as well as some intricacies of the single-agent planners in the CBS context (e.g., using an RL policy). We note that the CBS Protocol high-level search does not differ from CBS.

II. Related Work

Our work focuses on collision-free multi-agent motion planning (MAMP) for agents with independent tasks.

We first describe what we call "algorithmically homogeneous" MAMP methods that plan for agents using the same method across all agents, even if the agents themselves are heterogeneous. These methods explicitly require access to the internals of the agents and typically extend a single-agent planning technique to multiple agents. We then describe "algorithmically heterogeneous" MAMPF methods that plan for agents which have different single-agent planners.

A. Algorithmically Homogeneous MAMP Methods

We describe a range of algorithmically homogeneous MAMP methods based on their broader taxonomy.

1) Search-Based Planning: Search-Based Planning techniques for MAMP typically focus on a simplified problem set-up called Multi-Agent Path Finding (MAPF) [3]. In the "classical" MAPF set-up, the world is discretized into a graph where each vertex corresponds to a grid cell. Time is also discretized into timestep where each agent can move to an adjacent cell or wait at its current location at each timestep (these define edges between the corresponding vertices). Each agent occupies exactly one vertex/grid cell at a timestep. Search-based MAPF solvers use variants of A* space-time solvers. Operator Decomposition [4] addresses the combinatorial increase in search size by incrementally generating successors by planning individual agents. M* [5] uses a similar approach to plan for up to 100 2D agents.

Conflict-Based Search [2] is a foundational technique in MAPF. We describe CBS more in-depth in Section III-A. Conceptually, CBS is a two-level heuristic search method that iteratively detects and resolves conflicts. CBS utilizes a high-level search that searches over constraints and a low-level search the plans single-agent paths. CBS first plans each agent individually and detects conflicts (i.e., collisions). CBS then proceeds to iteratively resolve conflicts by adding constraints to specific agents and replanning until a collision-free solution is found. Although originally created for MAPF with single-agent A* solvers, CBS has been used with other single-agent solvers as we will see later in this section. Several methods use different variants of A* solvers to handle heterogeneous teams of car-like robots with kinematic constraints [6], [7] or manipulator arms [8].

2) Sampling-Based Planning: Multi-Agent RRT* [9] is an initial attempt that tries to plan for up to 10 2D agents by running RRT* on the combined state space. Multi-Robot Discrete RRT (MRdRRT) [10] is able to plan for up to a combined 60 degrees of freedom over various agents with different degrees of freedom. MRdRRT's main insight is to plan over a discrete implicit graph and to modify the extend operation to move agents sequentially in a priority order.

[11] plans for a heterogeneous team by generating a probabilistic roadmap for each agent and using CBS. Safe Interval Continuous-space CBS [12] recently proposed a novel single-agent Safe Interval RRT* with CBS to plan for up to 100 2D agents in continuous space.

3) Optimization Methods: Optimal Reciprocal Collision Avoidance (ORCA) [13] is able to plan for 1000's of 2D agents by having each agent independently solve a low-dimensional linear program based on the positions and velocities of nearby agents. [14] introduces using safety control barrier certificates via formulating and solving a quadratic programming problem. Recently introduced Space-Time Graphs of Convex Sets [15] utilizes agent priorities and convex optimization to find solutions for up to 10 2D agents.

[16] first uses discrete planning (A*) in CBS and then post-processes the solution with optimization to plan for a heterogeneous drone and ground robot team. Discontinuity-Bound Conflict-Based Search (db-CBS) [17] interleaves discrete search and optimization by using a discontinuity-bounded A* search (e.g., A* with motion primitives) in CBS to generate single-agent approximate solutions which are used as warm-starts for a joint optimization across all agents. If the optimization does not return a feasible solution, then constraints are added and the CBS search continues. Conflict-Based Model Predictive Control (CB-MPC) [18] replaces the low-level A* search in CBS with limited horizon MPC.

4) Machine Learning Methods: Multi-Agent DDPG [19] and Multi-Agent PPO [20] extrapolate single-agent DDPG [21] and PPO [22] respectively for multi-agent systems. These approaches are more general than collision-free MAMP but can be used for MAMP (e.g., [23]).

Several methods train a single agent policy for MAPF using reinforcement learning or supervise learning that learns to avoid collisions [24] and can handle up to 10000 agents in congestion [25]. Other recent work MMD [26] trained a diffusion model as a single-agent planner for CBS and incorporates constraints through guidance terms.

The main restriction with all these algorithmically homogeneous MAMP methods is that they require all agents to run the same planner. Thus, these methods cannot incorporate agents manufactured / programmed externally which limits their application in such instances. The CBS Protocol directly addresses this limitation.

Taking a step back, we note that CBS appears in each category with different types of single-agent solvers, with their emphasis on the single-agent solver. CBS as a Protocol has a different focus and can be seen as encompassing all these methods. Our perspective of CBS using an abstracted single-agent API means that we can recover [2], [6], [7], [8], [11], [12], [16], [18], [26] with different API implementations.

B. Algorithmically Heterogeneous MAMP Techniques

Unlike algorithmically homogeneous MAMP methods, algorithmically heterogeneous MAMP methods do not require each agent to run the same solver. To our surprise, we could

not find any published work that addresses this problem setting. Existing MAMP methods for heterogeneous teams focus on robots with different capabilities but use algorithmically homogeneous solutions (e.g., [7], [11], [16]). On the other hand, existing multi-agent task planning/coordination methods focus on heterogeneous behaviors or task assignment and not on collision-free movement [27], [28].

Thus, part of this paper's goal is to introduce / bring attention to the Algorithmically Heterogeneous MAMP (AH-MAMP) problem setting. AH-MAMP tries to achieve collision-free motion planning for heterogeneous single-agent solvers without being able to modify the solvers. Solutions for AH-MAMP instead require designing multi-agent *protocols* with well-defined single-agent APIs, with the protocol/API abstraction enabling using heterogeneous single-agent solvers.

To our knowledge, current solutions for algorithmically heterogeneous multi-robot teams in the real world currently plan robots individually viewing other robots as dynamic obstacles. The most common solution is for agents to plan independent paths and rely on local sensing to replan when another agent obstructs their path. However, this reduces their ability to move fast, requires additional planning time during execution, and can lead to deadlock. An alternative scheme could be to use Prioritized Planning [29] which assigns agents priorities and plans agents sequentially starting with the highest priority agent. Each agent plans a path that avoids colliding with the paths of previously planned (higher priority) agents. However, in congestion this can result in no solution being found, which limits its practicality. Taking a step back, the problem with both of these approaches is that they solve the multi-agent problem as greedy single-agent calls viewing other agents as obstacles, resulting in them failing in complex instances. To that end, we introduce CBS as a Protocol as a principled AH-MAMP protocol that does not view agents as obstacles but instead intelligently searches over space-time constraints to find collision-free solutions.

III. CBS as a Protocol

We first describe CBS from classical MAPF and then describe CBS as a protocol.

A. Conflict Based Search

CBS is a two-level heuristic search algorithm for solving MAPF optimally. We focus on the top-level which searches over "Constraint Tree" (CT) nodes and describe its core search components (e.g., nodes, goal condition, etc). Algorithm 1 describes the CBS Protocol (identical to CBS except for the `plan` API) and can be used to follow along.

CT Nodes: At the high level, CBS employs a best-first search over CT nodes. Each CT node contains 1) a set of constraints, and 2) a set of agent paths, one for each agent, that satisfy the constraints. Each constraint prevents an agent from occupying certain space-time locations and is used to avoid collisions with other agents. Conceptually, each CT node represents a candidate solution.

Goal Condition: A CT node is considered a goal node if its agents paths have no collisions (line 7).

Algorithm 1 CBS Protocol

Parameters: Agents a_i with `plan()` API
Output: Collision free paths

```
1:  procedure CBSProtocol({a_i})
2:      node = generateRoot()
3:      OPEN.insert(node)
4:      while OPEN.notEmpty() and not TimeOut() do
5:          node ← OPEN.pop()
6:          conflicts ← detectConflicts(node.volumes)
7:          if conflicts = ∅ then                    ▷ Found solution
8:              return node.paths
9:          constraints ← conflicts[0].getConstraints() ▷ Resolve the first conflict via constraints
10:         for (a_i, p, t) ∈ constraints do         ▷ Simplified
11:             n ← node.copy()                      ▷ New node
12:             n.cons[a_i].append((p, t))
13:             response ← a_i.plan(n.cons[a_i])
14:             if response is None then
15:                 continue ▷ Skip if API did not find solution
16:             n.paths[a_i], n.volumes[a_i], n.costs[a_i] ← response
17:             OPEN.insert(n)
18:     return No solution found
```

Successor Generator: Suppose we have a CT node with a set of agent paths and constraints. If the paths have no collisions, then this is a goal node and CBS terminates. However, if it does have a collision, then CBS chooses one conflict and resolves it by generating two child CT nodes (lines 9-17) with *mutually disjunctive* constraints (a pair of constraints such that a collision occurs if both are violated).

For example, suppose two agents collide at a specific vertex v_0 at timestep t_0, denoted (v_0, t_0). A valid solution cannot have both of those agents at (v_0, t_0), so CBS generates two possible candidate solutions; one with the first agent prevented from visiting (v_0, t_0) and the second with the second agent prevented from visiting (v_0, t_0). This results in two children CT nodes (one for each possibility). Importantly, each children node differs from the parent node by just having an additional constraint on one agent. Thus, generating a children CT node just requires replanning that one agent while keeping the other agents' paths the same. The replanning is typically done by a "low-level" heuristic search algorithm (e.g., A*).

Initial "Root" CT Node: CBS starts by generating the "Root" CT node which contains no constraints (line 2). Generating the Root CT node requires generating paths for each agent individually (this is the only node that requires planning all agents). Once generated, this node is inserted into the CT queue and the high-level CT search is started.

High-Level Search: CBS employs a best-first search over the CT nodes by using a priority queue (which is called "OPEN"). There are multiple ways to sort the priority queue based on the MAPF objective, but one standard way is to sort by CT node's cost (typically the sum of each agent's path length). CBS pops out the cheapest node and checks if

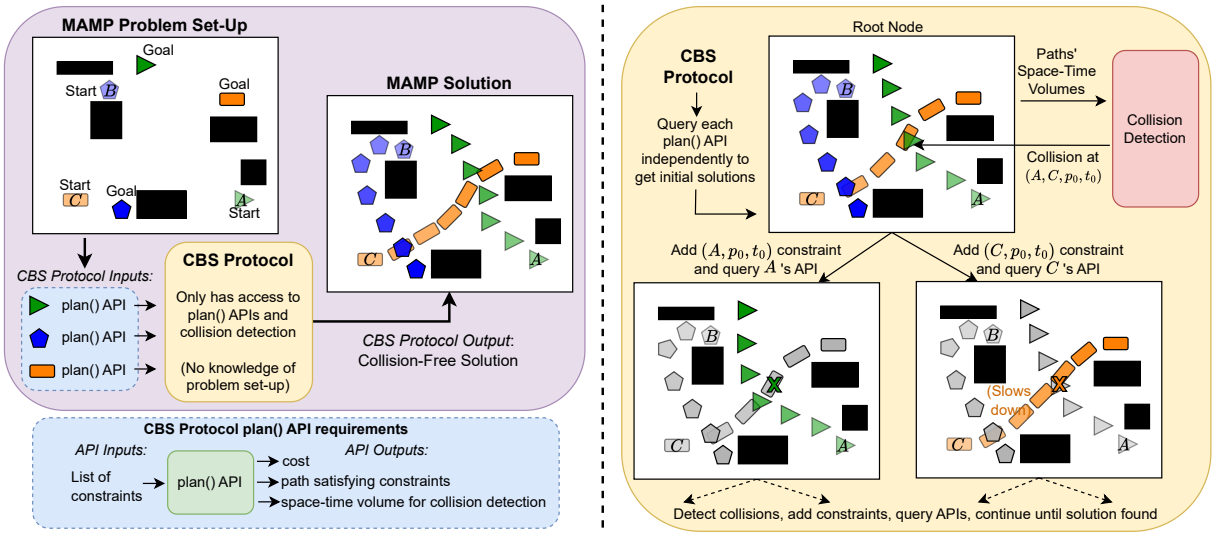

Fig. 1: We depict using the CBS Protocol (left) and its internal process (right). Left: Given a Multi-Agent Motion Planning (MAMP) problem, each agent defines a `plan()` API that satisfies the CBS Protocol requirements (blue dashed box). This is input into the protocol (which has no other knowledge except for collision detection) which uses the APIs and outputs a collision-free solution. Right: The CBS Protocol starts with generating a root node by querying each agent's plan() API without constraints. It detects collisions between agents (e.g., agents A and C collide spatially at point p_0 at time t_0) and generates two child CT nodes. Each node is created by adding a constraint (colored X) that the agent needs to avoid (p_0, t_0) and re-querying the agent's plan() API with the additional constraint. CBS repeats this process of detecting collisions, adding constraints, and querying APIs with additional constraints until a collision-free solution is found.

it is a goal node (line 5). If it is, CBS returns that node's paths as the solution. If not, CBS generates successor nodes and inserts them into OPEN and repeats (line 17).

Summary: CBS searches over candidate solutions (high-level search over CT nodes) by iteratively finding collisions, adding constraints, and replanning individual agents (low-level search). CBS is a centralized planner that plans full-horizon collision-free paths for all agents.

Properties: In the classical MAPF form with optimal high-level A* search and optimal low-level A* planners, CBS is complete and optimal (i.e., it will find the optimal solution given enough time and memory).

B. CBS as a Protocol

As mentioned in Sec II-A, although CBS has predominantly used A* single-agent planners, the general idea of iteratively planning and resolving collisions by searching over a Constraint Tree can be applied to other single-agent planners [17], [12], [18], [26]. However, those publications with CBS focus on developing the single-agent planner.

We come from the AH-MAMP perspective where we do not focus on single-agent solvers but instead want a mechanism to coordinate an algorithmically diverse set of agents. Our insight is that CBS can be used as a protocol across multiple different solvers at once. CBS solely requires each agent to have the ability to plan a path that satisfies the CT node's constraints. Thus, each agent could run their own individual solver with their own kinematic constraints. Since different agents could have different cost functions (other than path length), each solver should also return the solution cost, which can be included in the CT node.

An important caveat is that given a set of agent plans, CBS must detect collisions between agents. Thus, we require the single-agent planner to not just return a path but also the space-time volume occupied by it. Note that compact representations for this are possible (e.g., if the robot's footprint is static, giving the footprint along with the space-time path defines the occupied space-time volume).

CBS as a protocol is exactly the same as CBS except for the change of perspective of the single-agent planner. In particular, instead of the planner being predefined, we view the plan call as an API call that can be written independently.

Single-Agent Solver `plan()` API:

- Input: List of constraints
- Output: 1) Path that satisfies the constraints, 2) Space-time volume occupied by the path (likely through a compact representation), 3) Solution cost
  Or None if no such path found

The CBS protocol can now be used to plan collision-free paths independently of how the single-agent API is defined, enabling using different solvers. Additionally since we only require the output of the solver, this allows using proprietary solvers that do not want to share their internal workings.

In the context of CBS, this API will be repeatedly queried with different constraints. If the API is solving the same task between calls (e.g., start-goal task), this means the API is repeatedly solving the same problem with minor variations (through the constraints). [30] showed how CBS can be sped up by having its single-agent solver reuse "experience" from previous queries. This idea is broadly applicable to different types of single-agent motion planning solvers. Thus single-agent solver APIs are recommended to leverage experience from past queries for faster query times in the CBS Protocol.

Conflict Detection and Constraints: The CBS Protocol detects inter-agent collisions in a CT node's solution by checking for overlaps between all pairs of agents' space-time volume. If a collision is detected between two agents A, C, a representative space-time point (p_0, t_0) in the overlapped region is selected (in 2D this corresponds to a point $p_0 = (x_0, y_0)$). To resolve this collision, we add a $(\text{agent\_id}, p_0, t_0)$ constraint for each colliding agent that forbids the agent from overlapping spatially with (p_0, t_0) (we note prior work [31] introduced this constraint for large agents in gridworld MAPF). We highlight that the constraint is defined in the *workspace* dimension (e.g., 2D or 3D) and is independent of the dimensionality of each agent / solver.

A subtle but important caveat is that in continuous time motion planning, a constraint at a specific instantaneous time could be satisfied by planning a near identical path with an almost instantaneous wait, which will result in another collision at the same location at a negligibly offset time. Likewise for continuous space motion planning, a constraint at a single point could be satisfied by planning a near identical path with an almost negligible deviation. Thus, for practical purposes, the CBS Protocol should use constrain space-time volumes (e.g., forcing the agent to avoid p_0 from $(t_0, t_0 + \delta)$ where δ is the constraint duration) that force meaningful changes in agents' paths. We note that there are extensive improvements in the CBS literature for designing better constraints [8], [32], [33], but this is not related to our main contribution and should be incorporated in the future.

Independent Tasks: CBS for motion planning has been primarily used to plan collision-free start-goal paths. From the API perspective, however, the API simply needs to find a solution that satisfies the constraints. A few prior works have used this flexibility in CBS for planning paths that visit multiple goal locations [34]. Thus more generally, the API is not limited to planning start-goal paths and can be used for arbitrary single-agent tasks (e.g., coverage, exploration).

We highlight that the CBS Protocol requires independent tasks between agents. If agents need to explicitly collaborate (e.g., move an object together) while avoiding collisions with other agents, our protocol is not applicable for two reasons. First, collaborative behavior between two agents is robot / task specific while the CBS Protocol is robot / task agnostic. Second, collaborative behavior is fundamentally not reasoned about in CBS, which only focuses on resolving collisions.

If there are different groups of collaborative agent that need to work together, then the CBS Protocol could be used via meta-agents. Users could specify a single API that plans for all agents in a group (this is a "meta-agent"), which can then be input into the CBS Protocol to ensure that groups do not collide with each-other. Internally, each group API would run their own custom collaborative planning scheme.

Theoretical Properties of the CBS Protocol: The theoretical guarantees of using the CBS Protocol depend on the single-agent APIs and the constraints used. The CBS Protocol is "complete" (i.e., guaranteed to eventually find a valid solution in a solvable instance) given two conditions: 1) Each single-agent API will find a valid solution with bounded cost that satisfies constraints if a solution exists, and 2) The constraints are mutually disjunctive. The proof of completeness follows regular CBS [2].

However, as mentioned earlier, for practical performance constraints should be on space-time volumes, which are not mutually disjunctive. More broadly, in continuous time planning, designing meaningful mutually disjunctive constraints for CBS is an unsolved question [35]. Thus, the CBS Protocol is not theoretically complete in the continuous space-time context, however, it still offers a principled way to resolve inter-agent conflicts for efficient motion planning.

IV. Experiments

Our experiments seek to show the generality of the CBS Protocol by using a variety of solvers from different single-agent motion planning families. Our experiments focus on motion planning in a 2D occupancy map with agents with different kinematic constraints, solvers, and tasks.

A. Single-Agent Solvers

Table I shows a summary of the different solvers which we implemented the required `plan()` single-agent API for the CBS Protocol. We briefly describe each solver here and how they incorporate space-time constraints and experience.

1) Heuristic Search: We implement a single-agent A* search as a representative heuristic search algorithm. We plan in continuous space for robots with dynamic constraints (i.e., Ackermann and Dubins) by using discretized motion primitives. We use a modified version of soft-duplicate detection [36] to speed up search.

Constraints: Incorporating space-time constraints is straightforward; we search over space-time nodes and prune edges/nodes that violate space-time constraints.

Experience: We implement experience by using the technique in [30]. When we find a solution, we store the nodes on the solution path. Then when replanning, if we encounter a node in the solution path, we add as successors the rest of the nodes in the path that are valid under the new constraints.

2) Sampling Based Search: We implement Rapidly-exploring Random Tree (RRT) [37] as a representative of sampling based search algorithms. We use a regular spatial RRT for its simplicity and popularity (with each vertex having an associated time of arrival). We note this means that the RRT solution does not contain wait actions and that the solver is incomplete. However, it can still be used in a plan() API (it may just return failure when solutions exist).

Constraints: Space-time constraints are incorporated in the edge/node validity function.

Experience: Given a new query, we iterate through all nodes/edges in the tree created in the previous query and remove those invalid under the new constraints. We then start the RRT process from this tree. We note more sophisticated versions of replanning using experience exist (e.g., [38]).

3) Optimization: We formulate the motion planning problem as a trajectory optimization problem, whose objective is to drive the system from the initial state to the goal in minimum time. To this end, we transcribe the problem into a nonlinear program, with a cost function that minimizes a weighted objective and constraints that enforce both dynamic feasibility and spatial validity. Dynamic constraints

Solvers	Space-Time Constraints	Experience	Dynamic Constraints	Tasks
A*	Edge validity	Successor Generator [30]	Ackermann, 4-connected, Dubins	Start-goal, Coverage
RRT	Edge validity	Re-use partial valid tree	4-connected, None	Start-goal, Coverage
Direct Collocation	Constraint	Warm start	Ackermann	Start-goal, Surveillance
Diffusion	Penalty	Warm start [26]	None	Start-goal, Motion patterns
RL	Action masking	Caching	Unicycle	Start-goal, Coverage

TABLE I: Various Single-Agent Solvers used in the experiments with CBS-Protocol. We note that these solvers cover different motion-planning paradigms (heuristic search, sampling, optimization, diffusion, RL respectively).

are implemented using direct collocation [39], while collision avoidance is addressed through a duality-based exact representation [40]. We initialize the nonconvex nonlinear program with an A* solution and solve it using IPOPT [41].

Constraints: We handle space–time constraints explicitly, in the same way as dynamics and spatial constraints.

Experience: We incorporate experience by warm starting the optimization with the previous solution.

4) Diffusion: We implement a diffusion solver by modifying Multi-robot Multi-model planning Diffusion (MMD) [26] for our map and tasks.

Constraints: MMD incorporates space-time constraints by adding a guidance term into the diffusion process that penalizes the distance between the agents path and the (x, y) location of an active constraint.

Experience: MMD generates paths by initializing a path with complete random noise and denoising it. MMD uses experience by replacing the random noise initialization with the path of a previous solution with some slight noise added.

5) Reinforcement Learning: We implement an Reinforcement Learning (RL) solver using PPO [22]. To our knowledge, we are unaware of existing papers that have used an RL single-agent solver with CBS. The model takes in a local observation and outputs a categorical probability distribution of the next action. Concretely, the model inputs are:

1) An estimate of distance to goal computed via a backward Dijkstra assuming gridworld dynamics (so unaware of the agent's footprint or dynamics),
2) Local obstacle distances by ray casting at 30 degree increments (e.g., simulating a lidar scan), and
3) Previous action indices

The model outputs a 1-hot probability distribution corresponding to a discrete set of motion primitives for a robot with Dubins dynamics. We utilize motion primitives to handle kinodynamic constraints and as it also enables using action masking to speed up training. To generate a full path, we rollout the policy, greedily following its actions and backtracking if necessary when hitting a deadend (i.e., DFSing based on the agent's policy).

Constraints: We incorporate space-time constraints during test time by masking actions that violate constraints. We also tried training a constraint aware version (i.e., constraints were also input into the model) but that performed worse.

Experience: We cache the previous solution and given a new call, reuse the previous solution up to the earliest violated constraint, minus a small slack (5 simultation seconds).

B. CBS Protocol Settings

We primarily care about the success rate (as opposed to solution cost) in our experiments and therefore use a "Greedy" CBS variant [42] for the CBS Protocol that prioritizes CT nodes by number of pairs of agents with conflicts and tiebreaks by solution cost. We use discrete collision checking every 0.1 simulation seconds to detect conflicts. Each robot has a fixed footprint, so the APIs output space-time volumes by outputting their space-time path and footprint, which is then used by CBS for collision detection. Constraints for a collision at (x_0, y_0, t_0) block a spatial square of size 0.1 containing x_0, y_0 for 2.5 seconds. These numbers were semi-arbitrarily picked for performance and to emphasize continuous space / continuous time constraints and planning.

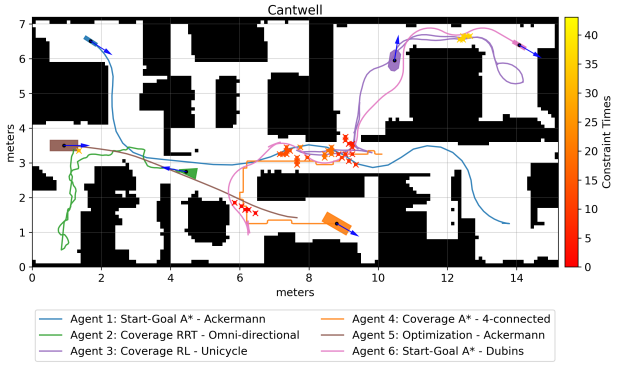

Fig. 2: An example of the CBS Protocol with 6 heterogeneous agents with different solvers, dynamics, and tasks. Constraints for the solution are plotted as colored x's.

C. Experimental Results

We test our method on the Cantwell (7.2x15.2, Fig 2), Ribera (8.7x6.3), and Mosquito (23.4x11.1) maps from the IN2D dataset [43] that realistically model a house or office space, as well as an empty map (10x15) and large (32x32) map with 10% obstacles (size in meters). Each robot has a footprint between 0.1 to 0.8 meters (some circles, rectangles, and triangles). All single-agent APIs/solvers, collision-checking, and the CBS Protocol were written in Python and experiments were run on a machine with an i7-11800H@2.30GHz x 16 CPU and 3050 Ti Laptop GPU.

Start-Goal Planning: We evaluate the CBS Protocol's ability to handle a variety of different solvers at once. Each solver was given a per-query time limit of 5-10 seconds with an overall CBS Protocol time limit of 120 seconds. The A* and RRT solvers worked on most instances on all the maps, while RL and Collocation struggled on the IN2D maps, and Diffusion worked well only on the Ribera map. For each multi-agent problem instance with N agents, we semi-randomly select N start-goal pairs, and then sample a random solver that is able to solve that pair. This means that the distribution of solvers and maps in our experiments is not

uniform. We generated instances with 2-10 agents through this mechanism which resulted in 675 instances.

Recall from Section III-A that the initial "root node" created in CBS requires planning each agent individually without constraints. Thus, the number of "root conflicts" (number of pairs of agents that conflicts) serves as an approximation for problem difficulty as having more conflicts implies a more congested problem instance that requires more work for the CBS Protocol to resolve. Additionally, a root node with 0 conflicts indicates that individually planning and executing would result in a successful instance. Of the 675 instances, 175 instances fell into this category.

Fig 3 shows the success rate (bottom) and number of CT Nodes generated (top) based on the number of root conflicts. The CBS Protocol solved 54% of instances with non-zero conflicts. As the number of conflicts increases, the number of CT Nodes generated for successful instances increases (e.g., the median of the blue dots increases). We also observe that the overall success rate decreases as root conflicts increase.

We emphasize that the CBS Protocol is tightly coupled to the success rate and speed of the underlying single-agent APIs as it repeatedly calls them during its search process. For example, the RL and Collocation solvers had relatively low success rates with constraints (35%, 46% respectively) compared to A* with Dubin's or 4-connected dynamics (87%, 94%). Additionally, some solvers/APIs took up to 10 seconds to satisfy a single query, which limited the number of CT nodes CBS could explore within the 2 minute timeout. Collision checking also took a non-negligible amount of time for larger amounts of agents.

We conducted an ablation study by removing "experience" from each single-agent API and rerunning experiments. This did not noticeably change the overall CBS protocol success rate, but substantially increased the median API query time, ranging from 10% to 300% increases depending on the solver, except for Collocation which took 20x longer.

Importantly, the raw results should not be interpreted as a fundamental limitation of the CBS Protocol. With optimized single-agent APIs and collision checking (e.g., planning times under 0.1 seconds), CBS is capable of generating thousands of nodes and achieving substantially higher success rates. Thus, our findings highlight the dependency of CBS performance on the single-agent API and illustrate the trend of CBS's CT node generation and conflict resolution.

Baseline: As mentioned in Section II-B, algorithmically heterogeneous baselines are non-existent. We did not implement a local replanning baseline as our individual APIs are too slow to be called often during execution (which may also be an issue in real applications). Instead, we let each agent plan individually and implemented a local priority-based collision avoidance policy. When a lower-priority agent is within a radius of 0.1 meters to a higher-priority agent, the lower-priority agent attempts to reverse direction and backtrack. If agents are within 0.05 meters, then they freeze. This baseline worked on 87% of instances without root conflicts and 9% of instances with root conflicts.

Heterogeneous Tasks: We do a proof-of-concept and replace the start-goal task in the plan API with different tasks. For A* and RRT we implement coverage planners

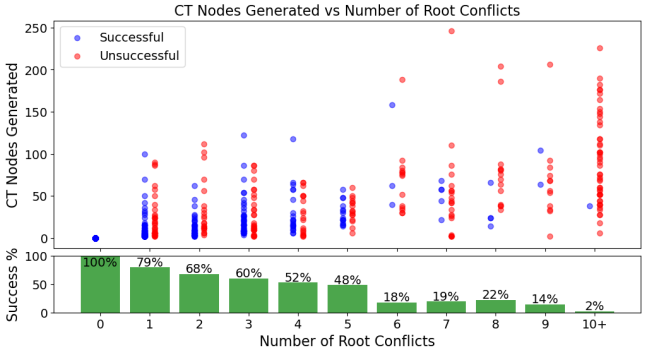

Fig. 3: We show results on MAMP start-goal problems with different solvers and categorize them by the number of conflicts in the root node (a proxy for problem difficulty). The top plot shows the number of CT nodes that the CBS Protocol generates for successful (blue) and unsuccessful (red) instances while the bottom shows the success rate.

that need to visit a set of points in any arbitrary order. For optimization (direct collocation), the agent attempts to maintain a radius around a given "surveillance" point. For diffusion, we train a policy to mimic motion patterns on data where the robot visits an intermediate point before proceeding to the goal. Finally, for the RL solver we implement a Roomba-like policy that tries to sweep the entire environment. Due to space constraints, Figure 2 shows a single example solution with various APIs (more examples are provided on the website). We highlight that we only need to input the different plan APIs into the CBS Protocol without any additional changes.

V. Conclusion and Future Directions

The future of robotics will require multiple robots from different companies completing individual tasks to efficiently find collision-free paths in shared environments. It is unclear how to effectively do this due to each robot having its own interfaces and motion planning software. To that end, we show how we can use Conflict-Based Search [2] as a protocol. The CBS Protocol solely requires agents to specify a single `plan()` API that takes in certain space-time constraints and returns a path with corresponding cost and space-time volume. We show how the CBS Protocol enables finding collision-free paths for a variety of solvers (A*, RRT, Direct Collocation, Diffusion, Reinforcement Learning), and is not limited to start-goal tasks but can work with arbitrary independent motion planning tasks.

Future Work: From an industry perspective, we would like to see the CBS Protocol adopted and deployed on real heterogeneous systems with efficient solvers.

From the single-agent `plan()` API perspective, there is significant potential in developing better solvers that handle CBS space-time constraints and replanning. In particular, these space-time constraints are a special subcategory of "dynamic" obstacles whose structure could be exploited for efficient planning. More effective versions of reusing experience in respect to these constraints is also promising.

From the AH-MAMP and multi-agent protocol perspective, the use of heterogeneous `plan()` APIs opens up

interesting questions on how to best leverage their strengths and weaknesses. For example, in our experiments, different APIs had varying success rates and query times. Enabling the CBS Protocol to intelligently reason about this would likely boost performance. Additionally, as a proof of concept, the current version of the CBS Protocol retools CBS from classical MAPF but does not include additional improvements in the MAPF literature. Achieving this would likely require expanding the single-agent `plan()` API inputs.

From the most broad perspective, we hope this work bridges a gap between single-agent motion planning, multi-agent motion planning, and MAPF, as well as encourages future development of multi-agent motion planning protocols that can handle heterogeneous single-agent solvers.

REFERENCES

[1] S. T. Brown, P. Buitrago, E. Hanna, S. Sanielevici, R. Scibek, and N. A. Nystrom, "Bridges-2: A platform for rapidly-evolving and data intensive research," in *Practice and Experience in Advanced Research Computing*, 2021.

[2] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, 2015.

[3] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Symposium on Combinatorial Search (SoCS)*, 2019.

[4] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[5] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*, 2011.

[6] L. Wen, Y. Liu, and H. Li, "Cl-mapf: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints," *Robotics and Autonomous Systems*, 2022.

[7] Y. Bai, S. Kotpalliwar, C. Kanellakis, and G. Nikolakopoulos, "Multi-agent path planning based on conflict-based search (cbs) variations for heterogeneous robots," *Journal of Intelligent & Robotic Systems*, 2025.

[8] Y. Shaoul, R. Veerapaneni, M. Likhachev, and J. Li, "Unconstraining multi-robot manipulation: Enabling arbitrary constraints in ecbs with bounded sub-optimality," in *Proceedings of the International Symposium on Combinatorial Search*, 2024.

[9] M. Čáp, P. Novák, J. Vokrínek, and M. Pěchouček, "Multi-agent rrt: sampling-based cooperative pathfinding," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[10] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning," *The International Journal of Robotics Research*, 2016.

[11] I. Solis, J. Motes, R. Sandström, and N. M. Amato, "Representation-optimal multi-robot motion planning using conflict-based search," *IEEE Robotics and Automation Letters*, 2021.

[12] J. Sim, J. Kim, and C. Nam, "Safe interval rrt* for scalable multi-robot path planning in continuous space," *arXiv:2404.01752*, 2024.

[13] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed autonomous robotic systems: The 10th international symposium*, 2013.

[14] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, 2017.

[15] J. Tang, Z. Mao, L. Yang, and H. Ma, "Space-time graphs of convex sets for multi-robot motion planning," in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025.

[16] M. Debord, W. Hönig, and N. Ayanian, "Trajectory planning for heterogeneous robot teams," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[17] A. Moldagalieva, J. Ortiz-Haro, M. Toussaint, and W. Hönig, "db-cbs: Discontinuity-bounded conflict-based search for multi-robot kinodynamic motion planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[18] A. Tajbakhsh, L. T. Biegler, and A. M. Johnson, "Conflict-based model predictive control for scalable multi-robot motion planning," in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2024.

[19] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, 2017.

[20] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," *Advances in neural information processing systems*, 2022.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[23] S. Peng, "Multi-robot path planning combining heuristics and multi-agent reinforcement learning," *arXiv:2306.01270*, 2023.

[24] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, 2019.

[25] H. Jiang, Y. Wang, R. Veerapaneni, T. Duhan, G. Sartoretti, and J. Li, "Deploying ten thousand robots: Scalable imitation learning for lifelong multi-agent path finding," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[26] Y. Shaoul, I. Mishani, S. Vats, J. Li, and M. Likhachev, "Multi-robot motion planning with diffusion models," in *International Conference on Learning Representations (ICLR)*, 2025.

[27] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, 2019.

[28] M. Bettini, A. Shankar, and A. Prorok, "Heterogeneous multi-robot reinforcement learning," *arXiv:2301.07137*, 2023.

[29] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, 1987.

[30] Y. Shaoul, I. Mishani, M. Likhachev, and J. Li, "Accelerating search-based planning for multi-robot manipulation by leveraging online-generated experiences," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2024.

[31] 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*, 2019.

[32] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, 2021.

[33] A. Andreychuk, K. Yakovlev, E. Boyarski, and R. Stern, "Improving continuous-time conflict based search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[34] Z. Ren, S. Rathinam, and H. Choset, "Conflict-based steiner search for multi-agent combinatorial path finding." in *Proceedings of Robotics: Science and Systems*, 2022.

[35] A. Li, Z. Chen, D. Harabor, and M. Vered, "Cbs with continuous-time revisit," *arXiv preprint arXiv:2501.07744*, 2025.

[36] W. Du, S.-K. Kim, O. Salzman, and M. Likhachev, "Escaping local minima in search-based planning using soft duplicate detection," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[37] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Technical Report*, 1998.

[38] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2006.

[39] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM review*, 2017.

[40] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, 2020.

[41] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, 2006.

[42] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[43] M. Dobrevski and D. Skočaj, "Adaptive dynamic window approach," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.