Decentralized Multi Agent Path Finding with ICBS

1st Anuj Jagetia
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA
ajagetia@wpi.edu

2nd Abhijeet Sanjay Rathi
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA
asrathi@wpi.edu

I. Introduction

The central focus of our investigation pertains to the advancement of Multi-Agent Path Finding (MAPF) by incorporating decentralized planning methodologies with Improved Conflict-Based Search (I-CBS). This fusion is designed to tackle the challenges posed by intricate environments where numerous agents, each governed by dynamic constraints, must navigate while avoiding collisions. The ramifications of this project extend across diverse domains, including robotics, autonomous vehicles, and industrial automation. Resolving MAPF in a decentralized manner, shows significant potential for enhancing the efficiency, safety, and scalability of multi-agent systems operating in dynamic and unpredictable environments.

II. Literature Review

The Multi-agent Path Finding (MAPF) domain has garnered considerable attention for its diverse applications, spanning warehouse management, airport towing, and shopping centers. At the core of MAPF lies the challenge of formulating strategies that enable multiple robots to navigate without encountering collisions. A study by A. Bolu et al. [3] proposes an intriguing solution for mobile robots navigating warehouse grids. This solution involves a modification of the A* algorithm that incorporates turning costs.

In a different vein, G. Sharon et al. [6] introduce the Conflict Based Search (CBS), a two-tiered algorithm that deliberately avoids oversimplifying the multi-agent problem into single-agent models. Instead, it pursues a distinctive path search approach through a conflict tree (CT), focusing on conflicts specific to each agent. The low-level aspect identifies optimal paths for individual agents, and if these paths involve conflicts, the high level, using a split action, imposes constraints on the conflicting agents to circumvent these conflicts.

It is noteworthy that the standard CBS, at each level of the CT, randomly selects a conflict to split, potentially leading to suboptimal choices that significantly impact algorithm performance. To address this, the Improved Conflict-Based Search Algorithm (ICBS) [4] was introduced, incorporating four extensions to enhance standard CBS. These enhancements, namely Meta-agent, Merge & Restart, Prioritizing Conflicts, and Bypassing Conflicts, can be added individually or in combination, with the exception of Merge & Restart, which is specifically relevant to MA-CBS.

III. Proposed Methods

A. ICBS Algorithm

In CBS (conflict based search), constraints are assigned to agents; where the constraints of agent a_i are represented as tuples (a_i, v, t) representing agent a_i is not allowed to remain at vertex v at time step t. An agent a_i path is considered consistent if it satisfies all a_i path constraints. A uniform solution is one that has a single solution that is the same for all agents. It is important to note that in the event of a disagreement between the representative, the same solution will not apply even if the representative's physical limitations are followed.

Advanced CBS works based on constraint tree (CT). CT is a binary tree with nodes (N) containing: (1) a set of constraints on the agent (N.constraints), (2) a solution based on those constraints (N.solution), and (3) Cost (N.cost). The basis of CT starts with an empty bounding space. One of the successors inherits the constraints of its parent node and adds new constraints to the agent. The solution (N.solution) is obtained by the low-level search process described below. When the solution of CT node N is valid, N becomes the goal node, which means that there is no conflict between the paths of each agent. Advanced CBS performs a top-first search for CTs, sorted by cost (N.cost).

Completion of a CT node involves a low-level search of each agent to find the optimal path given the constraints in node N. Any optimal single-agent search algorithm can be applied for low-level search of CBS. When a path is determined to be the same for all agents, the advantages of the agents over other agents can be analyzed by simulating their movements along the plan (N.solution). If all agents reach the goal without conflict, N is declared as the target of the goal and the N^{th} solution is returned. However, if a conflict occurs while verifying two or more agents, the verification is stopped and the node is considered off-target.

To resolve conflicts, specifically conflicts represented as (a_i, a_j, v, t) in a non goal CT node N, it is known that, in any valid solution, only one of the conflicting agents (a_i or a_j) may occupy vertex v at time t. Therefore, at least one constraint (a_i, v, t) or (a_j, v, t) must be satisfied. Therefore, CBS splits Node N, creating two new CT nodes as child nodes; each of these combines one of these constraints into the current set (N.constraints). Essentially, for each non-root CT node, a low-level search is initiated only for operators associated with the newly updated constraints.

The Conflict-Based Search (CBS) algorithm, employing standard splitting, occasionally makes suboptimal choices when randomly selecting conflicts to split and resolve. These choices can significantly impact its performance. ICBS, utilizing disjoint splitting, augments the standard splitting CBS by expediting the algorithm's execution.

Disjoint splitting presents an alternative to rectify inefficiencies in the standard splitting CBS algorithm. Instead of dividing each conflict into two negative constraints, ICBS with disjoint splitting partitions conflicts into a negative constraint and a positive constraint. The positive constraint, represented as (a_i, v, t), mandates that agent a_i must occupy location v at time t, implicitly preventing any other agent a_j (where j $\neq$ i) from occupying v at time t. The negative constraint where a_i is prohibited from being at v at timestep t (by adding $(\overline{a_i, v, t})$. This approach eliminates redundant plans between the two child nodes generated, potentially improving search efficiency.

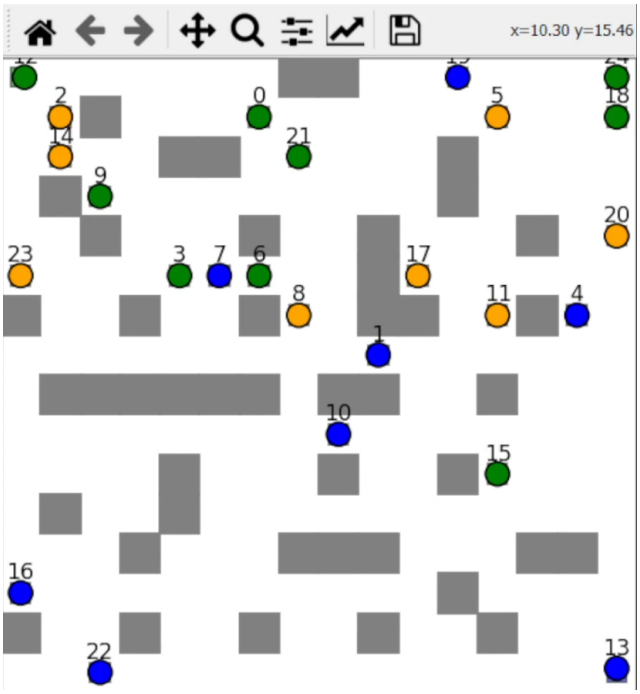

Fig. 1. *Final positions of the Robots*

Improvement 1: The meta-agent represented as a_{ij} is created by combining two agents named a_i and a_j when count of conflicts surpasses a predefined parameter B. If the overlap is less than B, the splitting is complete. A meta-agent is viewed as a composite agent with a state represented by position vectors; each position vector corresponds to an original agent. More importantly, meta-agents are not distributed in the tree below the current CT node, but can be mixed with other (meta)agents to create new meta-agents. The joint method consists of the joint operation of two joint systems. Since no changes occur in other non-participating agents, the downward search for this new meta agent is repeated. The problem of finding the optimal multiperson operator (MAPF) for a single representative of a given dimension can be solved using a MAPF solver (i.e., A*).

Improvement 2: The merge and restart strategy is designed to improve fund utilization by combining agents with meta-agents representing the CT root node (if integration is considered). When a collective decision is made for a group of agents G at CT node N, the current CT is discarded and the search is restarted from a new root. This new baseline represents the initial convergence of agents in group G, thus reducing the recomputation that can occur across multiple IT nodes.

Improvement 3: Conflict tendency is divided into three types of conflict. If you add from C((a1, v, t), (a2, v, t)) to N and repeat the subsearch of the limit operator, this will increase the value of the method by the cost. In N.C, it is semi-cardinal if adding a constraint increases N.cost but adding another constraint does not change N.cost. Finally, if adding two new constraints does not increase N.cost, then C is not cardinal.

Improvement 4: Bypass Conflict (BP) method is suitable for handling partial materiality or non-material conflicts. Instead of choosing a partition function, BP examines the children of N in the CT of a CT node N. If a child node method provides another method that has the same value but has no conflict, N will use that method without doing so. N needs to be allocated and new nodes need to be introduced to the CT. This method simplifies the search process due to the reduced CT size.

Sum of Costs SAT Solver: Let's express (Δ) as the difference between h_{SIC} and the value of the best cost (e.g. $\Delta = C^* - h_{Temperature}$). Also, let μ_o represent the length of the longest path of the shortest path. The main idea revolves around the claim that a solution whose value is equal to C^* should be reached in a maximum time step of $\mu = \mu_o + (\Delta)$. This is because, at worst, all (Δ) moves are attributed to workers with the shortest path of length μ_o. In this case, the SAT model is designed to determine how many solutions (Δ) over μ_o add edges in time $(\Delta) + \mu = \mu_o$. This method involves an outer loop that iterates over various values of (Δ).

This innovative method leverages the relationship between the heuristic h_{SIC}, the optimal sum-of-costs C^*, and the longest path μ_o to formulate a SAT formula. The formula addresses the question of whether a solution meeting the specified criteria is feasible within the stipulated time bounds. The outer loop, iterating over different (Δ) values, encapsulates the exploration of potential solutions with varying degrees of additional edges beyond the longest individual path.

B. A Algorithm*

The algorithm takes a map (graph) M, a set of agents A, and potential constraints C that limit agent movements, such as collision avoidance. It utilizes two crucial data structures: an OPEN list, which tracks frontier nodes for future expansion, and a CLOSED list, which logs nodes already explored. To facilitate tracking when all agents reach their goals, a meta-agent is introduced, encapsulating all individual agents within A.

The primary loop involves expanding the next node from the OPEN list, assessing whether all agent goals have been attained. If not, the algorithm generates successor nodes for each agent still in search.

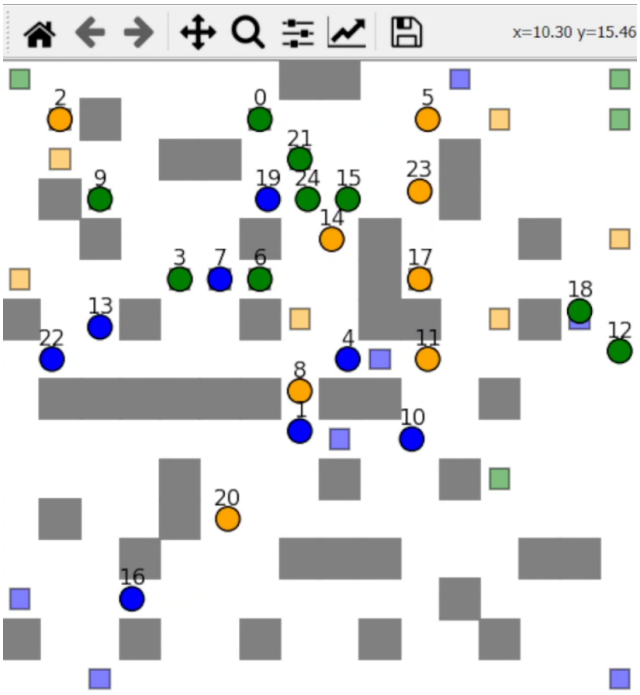

Fig. 2. *Initial positions of the Robots*

During successor generation, external constraints from C are considered, and any node violating a constraint is discarded. A heuristic h-value is computed for each agent, aiding in prioritizing nodes by cost. The generated child nodes are then incorporated into OPEN or CLOSED based on whether they have been previously encountered or if their cost is lower than the current node.

This iterative process continues, simultaneously exploring coupled state spaces for each agent until a joint goal state is reached or there are no remaining nodes to expand. The approach ensures the coordinated navigation of agents while adhering to specified constraints, contributing to the successful and constraint-compliant exploration of the environment.

IV. Platform and Evaluation

We have opted to utilize matplotlib as a tool for evaluating our project, with the intention of visually representing the connection between the number of agents and the associated CPU processing time needed for determining the optimal path. Furthermore, we plan to generate a second graph to illustrate the relationship between the number of agents and the success rate achieved in our project.

Throughout the project standard and disjoint splitting are compared for various test cases that is in the same environment if we keep increasing the number of robots, the results are analyzed in 3 graphs as below :

Result 1 : In result 1, the comparison of nodes generated for standard and disjoint method of CBS search are compared the results clearly showed that the disjoint splitting was effective than standard splitting and it reduced the nodes generated for all the cases till test case 5, the reference was not prominent but as the number of robot increased the results were clearly

Algorithm 1 High Level ICBS Algorithm

```
function MAIN(()MAPF problem instance)
    Init R with low-level paths for the individual agents
    while OPEN not empty do
        N ⟵ best node from OPEN // lowest solution cost
        Simulate the paths in N and find all conflicts.
        if N has no conflict then
            return N.solution // N is goal
        end if
        C ⟵ find cardinal / semi-cardinal conflict (N)
        if C is semi or non-cardinal then
            if Find-Bypass (N, C) then
                continue
            end if
            if should-merge (a_i, a_j) then
                a_ij = merge (a_i, a_j)
                if MR active then
                    Restart search
                end if
                Update N.constraints()
                Update N.solution by invoking low-level (a_ij)
                Insert N back into OPEN
            end if
        end if
    end while
end function
function GENERATE CHILD(Node N, Constraint C)
    A.constraints ⟵ N.constraints + (a_i, s, t)
    A.solution ⟵ N.solution
    Update A.solution by invoking low level (a_i)
    A.cost ⟵ SIC (A.solution) return
end function
```

seen for test case 17, the generated node for standard splitting were 162 and that of disjoint splitting were less than 150. The results are close to expected outcome of the project.

Result 2 : In result 2, the nodes expanded are compared for both algorithm extensions, the graph for result 2 shows that disjoint splitting showed effective performance in node expansion, here as well the disjoint splitting outperforms than standard splitting after test case 7 the difference between the nodes expansion of these 2 extensions were clearly visible still there was a case for test 15 when both algorithm expanded same number of nodes and on other side for test case 20 the difference in number of nodes expanded was 11.

Result 3 : In result 3, the time utilization for individual algorithm extensions are compared, the average time required for both algorithm extensions was about 0.3 seconds for maximum 25 agents which shows both the algorithm extension were time effective while considering the comparison between the standard and disjoint splitting as per the expectations the disjoint splitting showed better results than the standard splitting. For test case 7 the standard splitting took 0.05 second while disjoint splitting took 0.01 second, which shows a clear difference in the performance.

Along with this considering standard parameters of motion

Algorithm 2 A* Algorithm

```
function COUPLED A*(Map M, agents A, constraints C)
    initialize OPEN and CLOSED
    h —value ⟵ 0
    constraint-table constraints ⟵ for each agent in A
    for agent in A do
        initialize root node
        push root to OPEN
        CLOSED (loc, timestep) ⟵ root
        while OPEN not empty do
            curr ⟵ pop node from OPEN
            for agent a in meta-agents do
                if a in its goal then
                    a.reach —goal ⟵ True
                end if
            end for
            if agents reaches its goal then
                return Path
            end if
            for each agent that is searching do
                for each direction do
                    if curr[loc] is goal or invalid move then
                        Continue
                    end if
                    check external constraints
                    if constrained then
                        Continue
                    end if
                    Compute h-value for each agent
                    generate Child node
                    if Child is in CLOSED then
                        if Child is better than curr then
                            insert Child into CLOSED
                            Push Child into OPEN
                        end if
                    else
                        insert Child into CLOSED
                        Push Child into OPEN
                    end if
                end for
            end for
        end while
        print "NO SOLUTION"
    end for
end function
```

planning algorithm, the results clearly show that the path were optimum and the algorithm shows completeness for the path if one exists.

V. Application Impacts

Collision-free path planning is a critical aspect of managing safety, efficiency, and effectiveness in a multi-agent robot system. It serves as a preventative measure against collisions, ensuring the well-being of both the robots and their surroundings. This approach facilitates synchronized movement in situations

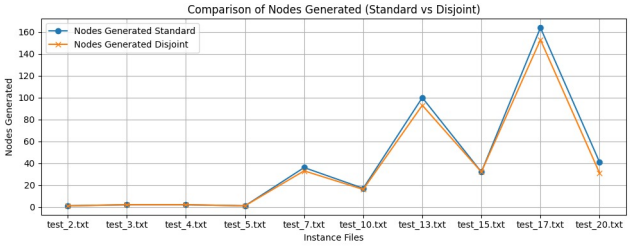

Fig. 3. *Initial positions of the Robots*

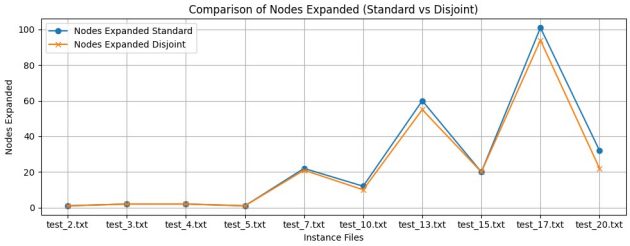

Fig. 4. *Initial positions of the Robots*

where multiple robots operate together, enabling them to complement each other's actions. This project focuses on studying multi-agent robot scenarios in non-grid-based environments to address dynamic and intricate settings, including:

1) Real-world Navigation: Maneuvering through crowded and unpredictable environments such as busy streets or shopping malls with uneven terrain.

2)Search and Rescue: Collaborating in disaster-stricken areas with irregular structures and obstacles, where agents work together to locate and rescue survivors.

3)Underwater Exploration: Exploring the ocean floor with multiple autonomous underwater vehicles, managing currents, varying depths, and unknown terrain.

4)Precision Agriculture: Coordinating multiple robotic agents in agricultural fields with irregular crop layouts, varying soil conditions, and unexpected obstacles.

5)Warehouse Logistics: Autonomous robots collaborating in warehouses with changing inventory, dynamic obstacles, and diverse item shapes and sizes.

VI. Challenges

(1) There are opportunities for enhancing MAPF algorithms, such as the development of hybrid approaches, optimization of existing algorithms, and exploration of admissible heuristics. There is potential for modifying optimal algorithms to achieve a better trade-off between runtime and solution quality.

(2) A comprehensive comparison of MAPF algorithms is essential to gain insights into their strengths and weaknesses. Creating a standardized set of benchmark instances would facilitate this comparison.

(3) Theoretical research on Multi-Agent Path Finding (MAPF) is currently hindered by a lack of clarity regarding how parameters like the number of agents, graph size, agent density, and conflict distribution impact problem difficulty.

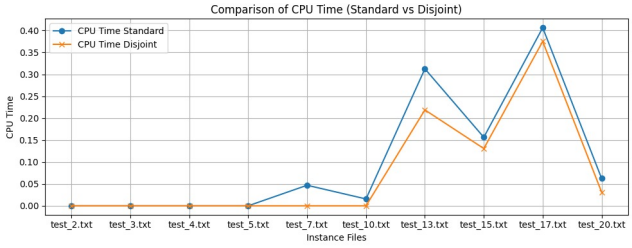

Fig. 5. *Initial positions of the Robots*

(4) The MAPF field is experiencing a division between sum-of-cost and makespan variants, emphasizing the need for cohesive research efforts that compare algorithms for both cost functions.

(5) Despite recent advancements in handling real-world scenarios, further research is required to effectively apply existing MAPF algorithms to domains with features such as directed edges, non-unit edge costs, uncertainty in actions, and dynamic appearances/disappearances of agents.

VII. Future Scope

In the context of upcoming advancements, our project aims to make substantial progress by empowering each mobile robot to autonomously navigate toward dynamically defined pick-up locations while safely interacting with fellow robots that serve as dynamic obstacles. This ambitious objective introduces heightened complexity, requiring the development and implementation of sophisticated algorithms to facilitate seamless collaboration among multiple robotic entities.

Additionally, we envision that the selection of pick-up locations will be intelligently determined based on the spatial arrangement of robots and their assigned tasks. This task-oriented approach seeks to enhance system efficiency by strategically choosing pick-up points, minimizing travel distances, and maximizing the utilization of each robot's capabilities.

VIII. Contributions

Abhijeet Rathi
Implementation of ICBS Algorithm
Compilation of the codes
Standard Splitting
Report Writing

Anuj Jagetia
Implementation of A* Algorithm
Disjoint Splitting
Report Writing
Overleaf document

Worked on everything mostly together

References

[1] Ariel Felner, Roni Stern Solomon Eyal Shimony; Eli Boyarski; Meir Goldenberg, Guni Sharon, Nathan Sturtevant; Glenn Wagner; Pavel Surynek (2017). *Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges*.

[2] Gloria Yoon; Yizhong Wang; Ash Peng (2021). *Improved Conflict-based Search (ICBS) with Disjoint Splitting*.

[3] A. Bolu and ¨O. Korc¸ak (2019) *Path Planning for Multiple Mobile Robots in Smart Warehouse*.

[4] E. Boyarski et al (2015) *ICBS: The Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding*.

[5] J. Li; D. Harabor; P. J. Stuckey; A. Felner; H. Ma, and S. Koenig (2021) *Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search*

[6] G. Sharon; R. Stern; A. Felner; N. Sturtevant (2012) *Conflict-Based Search for Optimal Multi-Agent Path Finding*