Research and Optimization of Conflict Search Algorithm for Multi-Agent Path Planning Based on Incremental Heuristic

Yang Li 1,2,3,4*, Jun Wang 1 and Hualiang Zhang 2,3,4

1 School of Computer Science, Shenyang University of Chemical Technology, Shenyang 110142, China
2 Key Laboratory of Networked Control Systems, Chinese Academy of Sciences, Shenyang 110016, China.
3 Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China.
4 Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110169, China.

*Corresponding author’s e-mail: 18340898859@163.com

Abstract. Conflict Based Search (CBS) is used for multi-agent Pathfinding (MAPF) to enable each Agent to reach the target node. The CBS algorithm uses the heuristic algorithm A* search to calculate the MAPF solution, and the path planning uses forward search, which cannot explore the path of the unknown region. On this basis, this paper proposes that, in the case of unknown map and changing environment at any time, when encountering new obstacles, the information obtained from previous search should be used without completely replanning the path. Because of the idea of incremental programming, the number of reprogramming times and the number of affected nodes can be reduced. The optimized algorithm consumes less time and memory, and improves the efficiency of path planning.

1. Introduction

With the continuous development of multi-robot technology, the pathfinding technology for multi-agent is gradually mature. Practical applications of computer games, simulations and robots require real-time solutions. Therefore, more and more attention has been paid to the study of multi-agent path planning algorithm in complex environment [1].

In order to solve the problem of multi-agent road strength planning, multiple agents are transported from their beginning to the target node on the same graph. The MAPF solver must consider the constraint that two agents can be located at the same graph node at the same time.

Single agent heuristic search methods, such as A* [2], use heuristic knowledge in the approximate form of the target distance to focus the search. Pearl et al. [3]. Proposed Lifelong Planning A* (LPA*), which uses two different technologies to reduce its planning time. Focused Dynamic A* (D*) [4] is an ingenious heuristic search method that increases the speed of repeated A* searches by one to two orders of magnitude by modifying previous search results locally.

In order to minimize the cost function, that is, to optimize the sum of the path length of all agents, Sharon et al. [5] proposed the Conflict based search algorithm (CBS), which is an effective MAPF solver. The newly proposed IDCBS[6] algorithm, in order to improve the memory consumption...
problem, uses A* search like IDA* instead of the underlying A* search algorithm in CBS to explore CT with repeated depth first iterations. Andreychuk et al. [7] proposed unbounded and bounded suboptimal MAPF solvers based on CBS, which relaxed the bottom and high-level search and allowed them to return suboptimal solutions. Li et al. [8] introduced an acceptable heuristic to guide the advanced search of CBS. Related researches, such as literature [9,10], respectively focus on incremental or iterative search algorithm, which improves the running time and success rate of the algorithm.

In this paper, based on the traditional CBS algorithm, a complete and time-saving algorithm is proposed. Although we do not guarantee the optimal features, compared with other algorithms, the proposed Conflict search based on incremental heuristic (CBSIH) algorithm has advantages in terms of time and space occupation. The proposed underlying algorithm changes the forward search path method, adopts the reverse search method, assumes that the unknown area is free space, on this basis, the incremental path planning is realized, by finding the shortest distance from the target point to each node. Through this modification, an efficient but suboptimal solution was developed. The proposed method is compared with CBS and some suboptimal algorithms belonging to different solutions, and the results are given in experiments. We used several graphic configurations with different numbers of agents.

2. Materials and Methods

CBS is a two-layer search algorithm. The bottom search finds an optimal path for each agent. When a conflict occurs between paths, the algorithm uses a high-level algorithm to split the conflict between the two known paths, and adds constraints to the agent to avoid conflicts. Perform a search based on the Conflict Tree (CT) between the paths of each agent. Each node in the conflict tree represents a set of constraints on the motion of the agent. CBS will find the path in the best way, otherwise it will fail.

The agent list includes a group of agents with a starting position and a target position. The goal is to find a path from the starting position to the target position without any conflict. Karabulut et al. [11] proposed that the environment is represented by a graph G: (V, E), where V={v0,…,vn} is a set of vertices, E={e1,…,et} is a set of edges, ei=(vx,vy) defines the edge between two vertices. A={a0,...,ak} is a group of agents, where the initial node si∈V of the agent and the target node gi∈V. Time is discretized into timesteps. In continuous timesteps, the action that each agent can perform is to move to an adjacent vertex or wait at the current node. Except for the agent arriving at the target node, other agents spend unit time costs in the process of moving and waiting. The agent ai moves from the initial node si to the target node gi through a series of movements and waiting. The tuple <ai,aj,v,t> is a vertex conflict, which means that ai and aj are at the same vertex v at time step t, and the tuple <ai,aj,(u,v),t> is an edge conflict, It means that ai and aj traverse the same edge (u, v) in opposite directions between time steps t and t+1 [12]. The goal is to find an effective conflict-free solution that minimizes the sum of individual costs (SIC) of each agent among all the constituent paths.

CBS is used in our multi-agent path finding problem. The working principle is to decompose the problem into a large number of single agent path finding problems, although the number of such single agent paths may increase exponentially. The algorithm is divided into two levels: high-level (search constraint tree) and low-level (constrained single agent path finding).

The bottom search algorithm in CBS can be any pathfinding algorithm, which is consistent with the constraint set of CBS to find the best path for the agent. The traditional underlying algorithm uses the heuristic pathfinding algorithm A*. A* is a complete and time-saving single agent pathfinding algorithm. By using the cost function f(n) = g(n) + h(n), the cost from the starting point n to the target point state is expressed. Among them, g(n) is the cost from the starting node to node n, and h(n) is the cost from node n to the target node [13]. A* takes an OPEN list and a CLOSE list. Select the node with the smallest value of the f(n) function from the OPEN list.

This article studies the use of the reverse search algorithm D* lite in an unknown environment. Unlike A*, D* Lite searches from the end point to the start point, so the definitions of g(n) and h(n) are different: g (n) represents the actual cost from the current point to the end point, and h(n)
represents the estimated value from the current point to the start point. The principle of the D* Lite algorithm initially needs to plan a global optimal path from the target point to the starting point based on the known environmental information and the unknown part as a free space. At this time, a "path field" information is established, which is to increase the quantity close to the target point to provide a basis for selecting the best. The D* Lite algorithm is a reverse search, \( g^*(s) \) records the predecessor node of the grid node, and the calculation formula is as shown in (1):

\[
g^*(s) = \begin{cases} 
0, & \text{if } s = \text{S}_\text{start} \\
\min_{s' \in \text{Pred}(s)}(g^*(s') + c(s', s)), & \text{otherwise}
\end{cases}
\]

RHS \( (s) \) records the \( g(s) \) of the raster-node's successor nodes. The RHS value of a point is the \( g \) value of its parent node plus the minimum value of the cost between these two points, which is equivalent to the minimum cost for a point to get to this point from the parent node. In fact, for most of the algorithm, the \( g \) value and the RHS value are equal. There is formula (2):

\[
\text{RHS}(s) = \begin{cases} 
0, & \text{if } s = \text{S}_\text{start} \\
\min_{s' \in \text{Succ}(s)}(g(s') + c(s', s)), & \text{otherwise}
\end{cases}
\]

In A* algorithm, the priority of A point is judged by the size of \( f(n) \), while in D* Lite, the priority of A point needs to be judged by two key values. The smaller the key value is, the higher the priority is. The first key value is judged first, and the second key value is judged if the first key value is equal. When evaluating the estimated value of the grid point, D*Lite also introduces the value of \( k(s) \) for comparison, where \( k(s) \) contains two values \( [k(s_1); k(s_2)] \), which satisfy the following formulas (3) and (4) respectively:

\[
k_1(s) = \min(g(s), \text{RHS}(s)) + h(s, s_{\text{start}}) + k_m
\]

\[
k_2(s) = \min(g(s), \text{RHS}(s))
\]

Formula (5) can be summarized from the above formula:

\[
h(s, s_{\text{start}}) = \begin{cases} 
0, & \text{if } s = \text{S}_\text{start} \\
c(s, s') + h(s', s_{\text{goal}}), & \text{otherwise}
\end{cases}
\]

The definition of \( k_m \) is that when the algorithm is initialized, \( k_m \) is first set to 0, and then when the robot detects a map change, \( k_m = h(s_{\text{last}}, s_{\text{start}}) \), \( s_{\text{last}} \) represents the previous starting point, and \( s_{\text{start}} \) is the current robot location. Because whenever the robot detects a change in the map, it will calculate the heuristic distance between two points (regardless of obstacles) and set the current point as a new starting point, that is, update the position of the starting point. First, the first key, which consists of three things, the minimum value of \( g \) at the current point and the minimum value of RHS plus the estimated value from the current point to the starting point plus one km. The first term is the actual distance to the end point, and the second term is the estimate to the starting point. If \( k_m \) is equal to zero before the robot moves, then the algorithm is equivalent to an A* algorithm that searches backwards from the end point to the starting point. The significance of the introduction of \( k_m \) is that when the robot detects the change of the obstacle, it will plan the path again. At this time, the actual starting point should be the current position of the robot. With the change of the starting point, the H value of each point will change accordingly, and the key value will also change. If this parameter is
not introduced, the key value of all nodes in the priority queue needs to be recalculated, which increases the calculation amount. After the introduction, the consistency of key value can be guaranteed to a certain extent and the calculation amount can be reduced. The second key value is the minimum value of \(g\) and RHS. Its significance is that when the first key value of the two points is equal, the algorithm will preferentially select the point near the end point.

Another important concept in D* Lite algorithm is local consistency, which is used to judge whether the current point needs to be calculated. The definition is as follows: a point is called locally consistent when its G value is equal to its RHS value; otherwise, it is called locally inconsistent. The local inconsistency can also be subdivided into local over-consistency and local under-consistency: when the g value of a point is greater than the RHS value, the point is locally over-consistency, usually when obstacles are deleted or when the algorithm searches for the path for the first time. When the g value of a point is less than the RHS value, the point is locally under-consistent, usually because a new obstacle has been detected.

The high-level handles search binary constraint trees (CT).

Each node \(N \in \text{CT}\) contains the following conditions:

1. Place a set of constraints on the agent \((N.\text{constraints})\). The node of CT contains a set of empty constraints, and the child nodes inherit the constraints of the parent node and add a new constraint for an agent. The constraint imposed on the agent \(a_i\) is a tuple \(<a_i, v, t>\), which means that the agent \(a_i\) is prohibited from occupying the vertex \(v\) at the time step \(t\).
2. A single solution \((N.\text{solution})\) that satisfies all constraints. A group has \(k\) paths, and each agent has one path. The path of the agent must be consistent with the constraints of \(a_i\), and these paths are found through the underlying search.
3. The total cost of the solution \((N.\text{cost})\). Represents the sum of path costs of all agents. The root node contains an empty set of constraints. The upper level performs the best-first search on the constraint tree and sorts the nodes according to their cost [14].

When \(N.\text{solution}\) is valid, node \(N\) in CT is the target node and there is no conflict in the path set between all agents. The high-level performs a best-first search on the CT, where the nodes are sorted by cost size. In CBS, the FIFO approach is broken to facilitate the resolution of CT nodes that contain fewer conflicts.

In the multi-agent pathfinding problem, with the increasing number of agents, the memory is also increased, in the limited time and space requirements, make full use of the computer CPU and memory and other hardware performance. In this paper, a method for solving multiagent pathfinding problem with a large number of agents is designed. To this end, the existing CBS algorithm is optimized, the CBS algorithm A * heuristic algorithm is to search algorithm into reverse search algorithm, realize when detected new obstacles, algorithm does not need to completely rethink path, can use the information gained by the search before, to find A can avoid obstacles path.

When the CBSIH high-level algorithm deals with conflicts, it calls the low-level search algorithm for each agent. Given CT node \(N\), it returns the optimal path consistent with each constraint in \(N\). The single agent path searched by the underlying algorithm verifies other agents based on a single solution \((N.\text{solution})\) that satisfies all constraints. If all agents have a path from the initial node to the target node, and there is no conflict between these paths. Then mark this CT node \(N\) as the target node and return \(N.\text{solution}\). If two (or more) agents \(a_i\) and \(a_j\) conflict \(<a_i,a_j,v,t>\) during the verification conflict, the verification conflict stops and the node is marked as a non-target node.

For a given non-target CT node \(N\), solution \(N.\text{solution}\) contains conflicts \(<a_i,a_j,v,t>\). In the effective solution, at most one agent \(a_i\) or \(a_j\) with conflicts may occupy vertex \(v\) at time \(t\). at least one of \((a_i, v, t)\) or \((a_j, v, t)\) constraints must be satisfied. CBS generates two new CT nodes as children of \(N\), and each node adds these constraints to the previous set of constraints \((N.\text{constraints})\).
Algorithm 1 Conflict based jump point search
Input: MAPF instance
1: function CBSIH
2: \texttt{R.constraints} = \emptyset
3: \texttt{R.solution} = find individual paths for all agents using D*lite()
4: \texttt{R.cost} = SIC\texttt{(R.solution)}
5: insert \texttt{R} into OPEN
6: while OPEN is not empty do
7: \texttt{P}←best node from OPEN with the smallest cost
8: if \texttt{P}\texttt{.collisions} = 0 then
9: return \texttt{P}\texttt{.solution}
10: \texttt{C}←first conflict(ai,aj,v,t) in \texttt{P}
11: foreach agent ai in \texttt{C} do
12: \texttt{A}←new node
13: \texttt{A.constraints}←\texttt{P}\texttt{.constraints}+(ai,s,t)
14: \texttt{A.solution}←\texttt{P}\texttt{.solution}.
15: Update \texttt{A}\texttt{.solution} by invoking low-level\texttt{(ai)}
16: \texttt{A.cost} = SIC\texttt{(A.solution)}
17: Insert \texttt{Q} into OPEN

Figure 1 (a) is the target node to the initial node. The corresponding CT is shown in Figure 1(b) [13]. The root contains an empty set of constraints. The underlying algorithm returns the best solution for each agent (line 2 of algorithm 1). <S1,A1,C,G1> is the path of agent a1, and <S2, B1, C, G2> is the path of agent a2. The total cost of this node is 6. The root is inserted into the OPEN list, and the two-agent solution given by two separate paths (line 7) is verified during the expansion of the node, when the two agents reach vertex C at time step 2. A conflict (a1, a2, C, 2) is generated, the root is declared as a non-target, and two child nodes are generated to resolve the conflict (line 11). Add constraints (a1,C,2) to the left child node, and add constraints (a2, C,2) to the right child node. Then call the underlying algorithm search (line 15) to find the best path that also satisfies the new constraints. For the left child, a1 must wait for a time step at S1 (or A1), and the path <S1, A1, A1, C, G1> returns to agent a1 and agent a2, <S2, B1, C, G2 The path of> remains the same for the left child. The total cost of the left child node is 7. In the same way, the right child node is generated at a cost of 7. Both child nodes are added to the OPEN list (line 17). Finally, the left child node is selected for extension and the underlying path is verified. Since there is no conflict, the left child node is declared as the target node (line 9), and its solution is returned as the optimal solution.

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{fig1.jpg}
\caption{(A) MAPF instance (b) Constraint tree (CT)}
\end{figure}

3. Results & Discussion
This experiment is all run in Linux system, equipped with Intel Core I7-8650U CPU with running frequency of 1.9ghz. Both algorithms are implemented in the same C++ code base. Our experiment on the MAPF benchmark consisted of 32 grids with different attributes (city map, grid with random
obstacles, maze, warehouse map, etc.), each with a digital scene (starting and target locations for up to 7,000 agents). The number of agents in the first scene of each grid is increased for each algorithm until the running time limit of 1h or the memory limit of 8GB is reached.

According to the experimental results, Figure 2 compares the running time of the two algorithms (solid line represents CBS and dotted line represents CBSIH), respectively showing the comparison of the running time under three different maps, and the result shows that the conflict search algorithm based on incremental heuristics consumes less time.

Experiments were carried out on three MAPF reference maps, and the scheme compared the two algorithm solvers in less than 1min. Table 1 shows the number of instances successfully solved by the two algorithms for different maps, that is, the data of all solved instances and difficult solved instances. A difficult instance is defined as requiring more than 30 seconds of allocation time to resolve. Experimental results show that the incremental heuristic conflict search algorithm can solve more problems.

| Group            | All instances | Hard instances |
|------------------|---------------|----------------|
|                  | CBS | CBSIH | CBS | CBSIH |
| 16x16_empty      | 39  | 58   | 25  | 50    |
| warehouse        | 60  | 90   | 40  | 64    |
| den520d          | 48  | 64   | 38  | 56    |

4. Conclusions
In this paper, CBSIH algorithm is proposed. By modifying the search strategy in the underlying algorithm and adopting a new fast reprogramming method, the target vertex is searched to the current vertex of the robot, the heuristic algorithm is used to focus the search, and similar methods are used to minimize the cost. Experimental results show that CBSIH performs better than CBS algorithm in these three maps. The search algorithm based on incremental heuristics is simple and the pathfinding time is short. Provides quantifiable robustness to MAPF problems while minimizing costs. In the future research work, it will be further combined with other advanced technologies to achieve a better multi-agent path planning algorithm.

References
[1] Wang W. Research and application of multi-robot path planning method based on multi-agent system [D]. Chengdu: University of Electronic Science and Technology of China, 2015.
[2] Nilsson, N. 1971. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill.
[3] Pearl, J. 1985. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley.
[4] Stentz, A. 1995. The focussed D* algorithm for real-time replanning. In Proceedings of the International Joint Conference on Artificial Intelligence, 1652–1659.
[5] Sharon G, Stern R, Felner A, et al. Conflict-based search for optimal multi-agent pathfinding[J]. Artificial Intelligence, 2015, 219: 40-66.

[6] Boyarski E, Felner A, Harabor D, et al. Iterative-Deepening Conflict-Based Search[C]//IJCAI. 2020: 4084-4090.

[7] Andreychuk A, Yakovlev K, Atzmon D, et al. Multi-agent pathfinding with continuous time[C]// Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. Macao: IJCAI Press, 2019:39-45.

[8] Li Jiaoyang, Felner A, Boyarski et al. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search[C]// Proceedings of the Twenty-Eighth International Joint Conference on artificial Intelligence. Macao: IJCAI Press, 2019: 442-449.

[9] Boyarski E, Felner A, Harabor D, et al. Iterative-Deepening Conflict-Based Search[C]//IJCAI. 2020: 4084-4090.

[10] Wan Q. Research on Multi-AGV Path Planning of Logistics Sorting Based on CBS Algorithm[D]. Harbin Institute of Technology, 2018.

[11] Karabulut İ Y. Suboptimal conflict based search for multi agent pathfinding[D]. Middle East Technical University, 2019.

[12] Matteo B, Nicola B, Francesco A. Multi-Agent Path Finding in Configurable Environments[C]// Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems.[s.n]2020: 159–167.

[13] Chen Mengxia. Research on multi-AGV path planning in large-scale scenarios of logistics sorting system[D]. Harbin: Harbin Institute of Technology, 2020.

[14] Barer M, Sharon G, Stern R, et al. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem[C]//Seventh Annual Symposium on Combinatorial Search. 2014: 961 – 962.