Received August 26, 2020, accepted September 5, 2020, date of publication September 10, 2020,
date of current version September 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3023200

Cooperative Pathfinding Based on Memory-Efficient Multi-Agent RRT*

JINMINGWU JIANG AND KAIGUI WU
College of Computer Science, Chongqing University, Chongqing 400044, China
Corresponding author: Kaigui Wu (kaiguiwu@cqu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61662083.

ABSTRACT In cooperative pathfinding problems, non-conflict paths that bring several agents from their start location to their destination need to be planned. This problem can be efficiently solved by Multi-agent RRT*(MA-RRT*) algorithm, which is still state-of-the-art in the field of coupled methods. However, the implementation of this algorithm is hindered in systems with limited memory because the number of nodes in the tree of RRT* grows indefinitely as the paths get optimized. This paper proposes an improved version of MA-RRT*, called Multi-agent RRT* Fixed Node(MA-RRT*FN), which limits the number of nodes stored in the tree of RRT* by removing the weak nodes on the path which are not likely to reach the goal. The results show that MA-RRT*FN performs close to MA-RRT* in terms of scalability and solution quality while the memory required is much lower and fixed.

INDEX TERMS Cooperative pathfinding, collision avoidance, multi-agent motion planning, path planning.

I. INTRODUCTION

The problem of planning a series of routes for mobile robots to destinations and avoiding collisions can be modeled as a *cooperative pathfinding* problem. Traditionally, this problem is often simulated in highly organized environments such as grids, which include several obstacles and agents. To find the paths of these agents, the straightforward method is looking for the answer in a joint configuration space which is composed of the state spaces of single agents. Such a space is typically searched using a heuristic guided function such as A* [1]. However, the problem of cooperative pathfinding is proved to be PSAPCE-hard [2].

To solve cooperative pathfinding problems, many works were proposed in the last decades. All these methods can be divided into three categories: decoupled method, coupled method, and hybrid method. Each method has its disadvantages. For example, the computational cost of coupled approaches is sensitive to the increase of agents, while the decoupled methods cannot guarantee their completeness. The hybrid approaches, which inherit the advantages of the coupled and decoupled approaches, seem promising. But when the decoupled planner fails, it may be more time-consuming than just using a single planner.

The associate editor coordinating the review of this manuscript and approving it for publication was Heng Wang.

After Karaman and Frazzoli introduced an asymptotically optimal algorithm, RRT* [3], Čáp combined RRT* with the classical multi-agent motion-planning algorithm and proposed Multi-agent RRT*(MA-RRT*) [4]. MA-RRT* is a coupled algorithm, but, unlike other coupled approaches, it alleviates the increase of computational cost as the number of agents increases by leveraging the idea of the Monte Carlo method. Thus, it can solve the multi-agent path planning problem efficiently. Besides, MA-RRT* comes close to the decoupled planner in terms of the efficiency while still maintaining the completeness and optimality, which makes the MA-RRT* still advanced in the field of coupled techniques.

There are many state-of-the-art works that aim to improve the MA-RRT*, such as [5] in 2019, which improves the efficiency of MA-RRT* at the expense of optimality and completeness. Unlike MA-RRT*, [5] applies RRT* for each agent in turn, and the agents whose paths have been planned by RRT* are treated as moving obstacles.

However, the application of the MA-RRT* is hindered in systems with limited memory, because as the solution gets optimized, the number of nodes in the tree grows indefinitely. The closest work to this problem is the RRT* Fixed Nodes (RRT*FN) proposed by Adiyatov and Atakan Varol [6], which only focuses on improving the memory efficiency of RRT*. Up to now, there is no prior work which limits the memory required for the MA-RRT* algorithm.

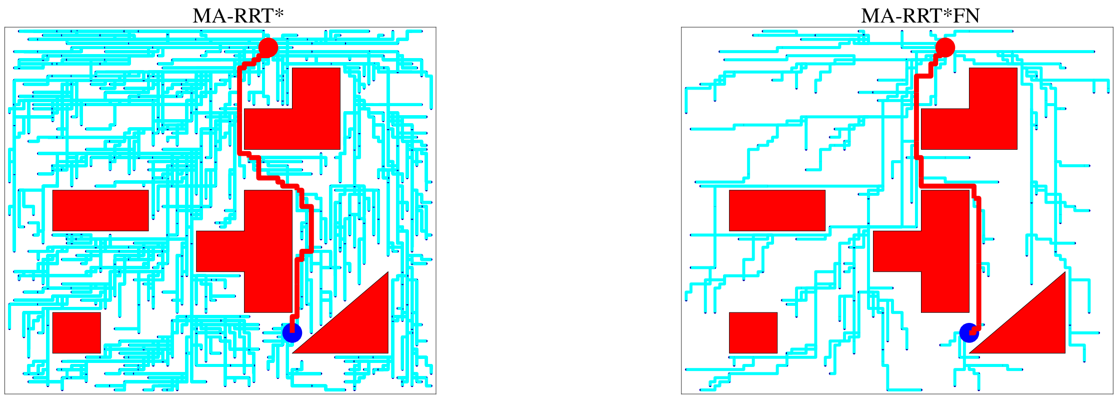

FIGURE 1. Single agent navigation using MA-RRT* and MA-RRT*FN respectively.

This paper presents a new MA-RRT* based algorithm, called Multi-agent RRT* Fixed Nodes (MA-RRT*FN), which works by employing a node removal procedure to limit the maximum number of nodes in the tree. The properties of our algorithm can be observed in Fig. 1, which shows the two search trees for single-agent navigation using MA-RRT* and MA-RRT*FN respectively in a 2D grid map with the same number of iterations. As shown in Fig. 1, the tree MA-RRT*FN generated is more sparse than MA-RRT*.

In our previous work [7], an extended abstract of our algorithm, we only illustrate our basic ideas and experiments roughly. In this paper, we present a review of related algorithms, a better formulation of the problem, the details of algorithms and pseudocodes, the illustrations of the theoretical principle, the interpretations of the experimental part and the discussions of limitations.

The main contributions of this paper are as follows: 1) The proposed MA-RRT*FN requires a fixed memory, which is much less than MA-RRT* (whose memory cost grows indefinitely), while the scalability and convergence rate of MA-RRT*FN are very close to MA-RRT*. 2) The informed-sampling MA-RRT*FN (isMA-RRT*FN), which is the improved version of MA-RRT*FN, performs very close to informed-sampling MA-RRT* (isMA-RRT*) concerning the suboptimality of solutions, while its convergence rate and scalability are better than isMA-RRT*.

II. RELATED WORK

The methods of cooperative pathfinding can be classified into three categories: coupled method, decoupled method, and hybrid method.

A. COUPLED APPROACHES

In coupled approaches, all agents' routes are computed as a union. The algorithm searches agents' joint configuration space to find the solution, which can provide a stronger guarantee of the feasible path and minimum cost. For example, Standley [8] proposed two techniques which take account of all agents at once, called Independence Detection (ID) and Operator Decomposition (OD). The combination of these two techniques, the ID+OD algorithm, which is capable of solving relatively large problems within milliseconds, is both complete and optimal. Standley then refined the algorithm into an anytime algorithm called Optimal Anytime (OA) [9], which first finds out a solution rapidly, and then utilizes any spare time to improve that solution incrementally.

Alternative methods such as [10] and [11] model the cooperative pathfinding problem as Integer Linear Programming (ILP) and Boolean Satisfiability (SAT) problems, respectively. In [10], Jingjin established a one-to-one solution mapping between multi-robot path planning problems and a special type of multi-flow network and used the Integer Linear Programming to optimize the four goals: the makespan, the maximum distance, the total arrival time, and the total distance.

However, all these coupling methods are sensitive to the increase in agents. The computational cost of these technologies increases dramatically with the increase in robots.

There are also many attempts to use the sampling-based method, such as RRT [12]–[14], PRM [15], to solve multi-agent motion planning problems. These algorithms alleviate the increase of computational cost as the number of agents increases by leveraging the idea of the Monte Carlo method. However, for the reason that they are based on the RRT and PRM, these algorithms are not optimal. After the RRT* is proposed, Čáp combined RRT* with OA and proposed MA-RRT* [4]. Unlike other sampling-based methods, MA-RRT* is both optimal and complete, and it outperforms many classical algorithms such as ID+OD and OA in terms of efficiency and scalability.

B. DECOUPLED APPROACHES

In decoupled approaches, all agents' paths are planned individually. For example, in [16], David Silver introduced three decoupled approaches which decompose the cooperative pathfinding problem into several single-agent navigations: Local Repair A*(LRA*), Hierarchical Cooperative A*(HCA*) and Windowed Hierarchical Cooperative

A*(WHCA*). In [17]–[20], the path of each agent is computed individually based on the pre-assigned priorities. The same case can also be seen in the recent works [5], [21].

Alternative method utilizes conflict based search (CBS) [22] to find the solution. Such as the ByPass-CBS and Continuous-Time-CBS proposed in recent work [23] and [24], which push the performance of CBS further.

Although those decoupled methods can efficiently find the solution, their completeness and optimality cannot be guaranteed.

C. HYBRID APPROACHES

The hybrid approaches, which leverage the strengths of both coupled and decoupled techniques, find the solution by firstly employing decoupled methods, and if the decoupled techniques fail, the coupled approaches will be employed. For example, M* [25] solves the multi-agent path planning problem by taking the decoupled manner first, and when the robots' paths conflict, the conflicting agents will be merged into a meta-agent and planned the path by a coupled planner.

The recent works based on CBS also take the idea of hybrid approaches, for example, MetaAgent-CBS [26], and its improved version [27], which employ the decoupled techniques first to detect conflicts, then merge the conflicting agents and apply coupled methods.

Although it seems that the hybrid approaches inherit the advantages of both coupled and decoupled approaches, it may take more time when the decoupled planner fails.

III. PROBLEM FORMULATION

To make a fair comparison with the MA-RRT* algorithm, which is simulated on graphs, the paper tests both the two algorithms (MA-RRT* and MA-RRT*FN) in a four-connected grid world G_M and uses the following definition: Assume that n agents labeled $1, \ldots, n$ running on a Euclidean space, and each agent, which takes up a single cell $x_i (i \in [1, n])$ of the grid world, has a unique start location s_i and destination d_i. For each timestep, all agents can move to its four neighbor cells $x'_i (x'_i \in children(G_M, x_i))$ if it is free or stay at its current location [9]. Besides, the transitions in which agents pass through each other are prohibited.

A cell is free means that it will not be occupied by an agent at the end of the timestep and does not include an obstacle [9]. The timesteps that a single agent stays on a grid cell are represented as $dur(x_i)$. The total number of timesteps c that the agent has taken from its start state s_i to the goal location d_i is regarded as the cost of the individual agent's path $path_i$. If all the agents can reach their goal without collision, then the sum of each path cost is taken as the cost of the final solution, which is the metric of solution quality. Formally,

$$cost(p) = \sum_{i=1}^{n} \sum_{x_i \in p_i} dur(x_i)$$

where p stands for an n-tuple of paths $(p_1, \ldots, p_n)$. To simplify the representation of nodes in the rapidly random tree, this paper uses x to represent the n-tuple of position $(x_1, \ldots, x_n)$. The starting positions of all agents are given as s, which is an n-tuple $(s_1, \ldots, s_n)$. Similarly, the n-tuple $(d_1, \ldots, d_n)$ is the destination d. Thus, a node in the tree can be denoted as an n-tuple joint state, and each state stands for the position of a single agent.

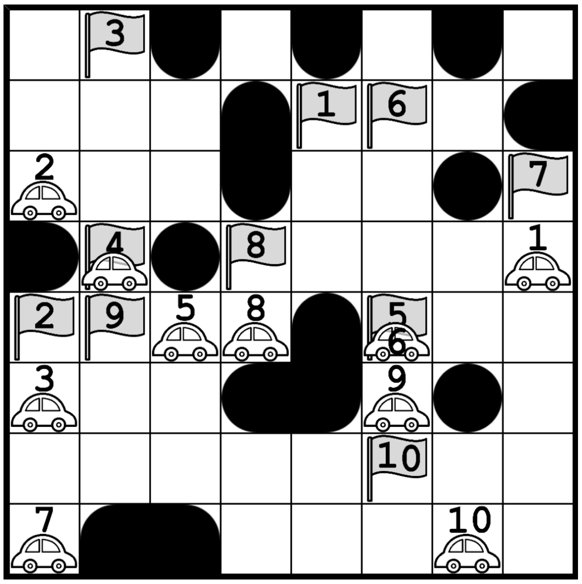

FIGURE 2. A small random grid world instance, in which the cars represent initial agent positions, the flags represent destinations and the obstacles are in black.

IV. MA-RRT* FIXED NODES ALGORITHM

The multi-agent RRT* algorithm is designed based on the RRT* algorithm, which can quickly find a path from a specific start location to a given target region in continuous state space by incrementally building a tree [3]. When the first solution is found, the RRT* algorithm will continue to improve the solution by sampling new random states in the configuration space, which leads to the discovery of a lower-cost path.

The MA-RRT* inherits all the properties of RRT*. For RRT* in a continuous configuration space, if two nodes are mutually visible, then they can be connected. While in a discrete space such as a graph, two nodes can only be connected if a valid path between the two nodes can be found by the heuristic search. Thus, the MA-RRT* is more like a graph version of RRT*(G-RRT*), unless it searches for the shortest path in a configuration space which stands for the joint-state of all agents [4].

Algorithm 1 shows the skeleton of the MA-RRT* algorithm. It begins with a tree that is rooted at the joint initial state x_{init} and continues to sample the random state x_{rand} from the free joint configuration space before extending the tree to x_{rand}. This loop will continue until it is interrupted.

The MA-RRT* Fixed Nodes (MA-RRT*FN) utilizes the skeleton of the MA-RRT* algorithm (shown on Algorithm 1, 4) and extends it with some node removing

Algorithm 1 MA-RRT*

```
1: $V \leftarrow \{x_{init}\}$;  $E \leftarrow \emptyset$
2: while not interrupted do
3:     $T \leftarrow (V, E)$;
4:     $x_{rand} \leftarrow SAMPLE$
5:     $(V, E) \leftarrow EXTEND(T, x_{rand})$
6: end while
```

Algorithm 2 MA-RRT*FN

```
1: $V \leftarrow \{x_{init}\}$;  $E \leftarrow \emptyset$;
2: while not interrupted do
3:     if $M = NodesInTree(v)$ then
4:         $(V_{old}, E_{old}) \leftarrow (V, E)$
5:     end if
6:     $T \leftarrow (V, E)$;
7:     $x_{rand} \leftarrow SAMPLE$;
8:     $(V, E) \leftarrow EXTEND(T, x_{rand})$; (See in algorithm 3)
9:     if $M > NodesInTree(v)$ then
10:        $(V, E) \leftarrow ForceRemoval(V, E)$;
11:    end if
12:    if $No\ ForceRemovalPerformed()$ then
13:        $(V, E) \leftarrow RestoreTree()$;
14:    end if
15: end while
```

Algorithm 3 EXTEND(T, x)

```
1: $V' \leftarrow V$;  $E' \leftarrow E$
2: $x_{nearest} \leftarrow NEAREST(T, x)$
3: $(x_{new}, p_{new}) \leftarrow GREEDY(G_M, x_{nearest}, x)$ (See in algorithm 4)
4: if $x_{new} \in V$ then
5:     return $G = (V, E)$
6: end if
7: if $p_{new} \neq \emptyset$ then
8:     $V' \leftarrow V' \cup \{x_{new}\}$
9:     $x_{min} \leftarrow x_{nearest}$
10:    for all $x_{near} \in X_{near}$ do
11:        $(x', p') \leftarrow GREEDY(G_M, x_{near}, x_{new})$
12:        if $x' = x_{new}$ then
13:            $c' \leftarrow cost(x_{near}) + cost(x_{near}, x_{new})$
14:            if $c' < cost(x_{new})$ then
15:                $x_{min} \leftarrow x_{near}$
16:            end if
17:        end if
18:    end for
19:    $parent(x_{new}) \leftarrow x_{min}$
20:    $E' \leftarrow E' \cup (x_{min}, x_{new})$
21:    for all $x_{near} \in X_{near} \setminus \{x_{min}\}$ do
22:        $(x'', p'') \leftarrow GREEDY(G_M, x_{new}, x_{near})$
23:        if $cost(x_{near}) > cost(x_{new}) + cost(x_{new}, x_{near})$ and $x'' = x_{near}$ then
24:            if $onlyChild(parent(x_{near}))$ and $M = NodesInTree(v)$ then
25:                $RemoveNode(parent(x_{near}))$
26:            end if
27:            $parent(x_{near}) \leftarrow x_{new}$
28:            $E' \leftarrow E' \cap \{(x_{parent}, x_{near})\}$
29:            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
30:        end if
31:    end for
32:    return $G' = (V', E')$
33: end if
```

procedures (shown on Algorithm 2, 3). Therefore, the MA-RRT*FN behaves like MA-RRT* before the maximum number of nodes is reached, and after the number of nodes reaches a threshold, it continues to optimize the tree by removing the weak nodes that are not likely on the path to reach the goal while adding a new node.

The skeleton of MA-RRT*FN is shown in Algorithm 2. Initially, the tree grows before the maximum number of nodes **M** is attained, after which the MA-RRT*FN removes a node with one or no child in the tree before adding a new node. The MA-RRT* and MA-RRT*FN use the same skeleton of EXTEND and GREEDY procedure, shown in Algorithm 3 and 4 respectively.

Like the MA-RRT*, in each iteration of MA-RRT*FN, the SAMPLE routine randomly chooses a free state in the joint space. Then, the EXTEND function generates a new node x_{new} in the free space by steering from the nearest node to the new random sample, and then check whether x_{new} is contained in this tree. If so, x_{new} will be deleted from the tree, and the EXTEND function will restart. If not, x_{new} will be added to the tree. After that, the algorithm searches the nodes near x_{new} to construct the node-set X_{near}, and then chooses a node which makes x_{new} have the lowest cost to initial state from X_{near} and $x_{nearest}$ as its parent. Finally, it updates the cost of X_{near} by rewiring to x_{new} if these nodes decrease the total cost by assigning x_{new} as the parent.

Unlike MA-RRT*, the MA-RRT*FN employs a node removing procedure in the EXTEND function, shown on lines 24 and 25 in Algorithm 3. During the EXTEND procedure, the algorithm updates the cost of nodes near the newly added node x_{new}. If a node x_{near} from X_{near} can reach a lower cost to the initial state by reconnecting to the newly added node, then the algorithm will check whether the parent of this node contains only one child and whether the number of nodes in the tree reaches a threshold M. If so, x_{near} will be rewired as a child of x_{new}, and the parent of x_{near} will be deleted.

Fig.3 visualizes this procedure in a simple 2D case. The red node in this figure represents a new node x_{new} which is going to be added to the tree, and the red dotted circle indicates the near domain of x_{new}, represented as X_{near}. If none of the nodes in the near domain of x_{new} has only one child to remove, then the *ForcedRemoval* procedure in Algorithm 2 will be employed, which searches the entire tree, except the x_{new} and the goal node, to find the nodes without children and deletes

Algorithm 4 GREEDY(G_M, **s**, **d**)

```
1:  $x \leftarrow s$;  $c \leftarrow 0$;  $path \leftarrow (\emptyset, \ldots, \emptyset)$
2:  while $x \neq d$ and $c \leq c_{max}$ do
3:      $(path_i, \ldots, path_n) \leftarrow path$
4:      for all $x_i \in x$ do
5:          $N \leftarrow children(G_M, x_i)$
6:          $x' \leftarrow argmin_{x \in children(G_M, x_i)} h(x_i)$
7:          $c \leftarrow c + cost(x_i, x_i')$;  $path_i \leftarrow path_i \cup (x_i, x_i')$;
8:          $x_i \leftarrow x_i'$
9:      end for
10:     if not $COLLISIONFREE(path_1, \ldots, path_n)$ then
11:         return path
12:     else
13:         $path \leftarrow (path_1, \ldots, path_n)$
14:     end if
15: end while
16: return (x,path)
```

Algorithm 5 isMA-RRT*FN

```
1: while not interrupted do
2:     for $i = 1 \ldots n$ do
3:         run the G-RRT* algorithm for agent i
4:     end for
5:     if all agnents find the paths though G-RRT* then
6:         run MA-RRT*FN algorithm based on biased sampling
7:     end if
8: end while
```

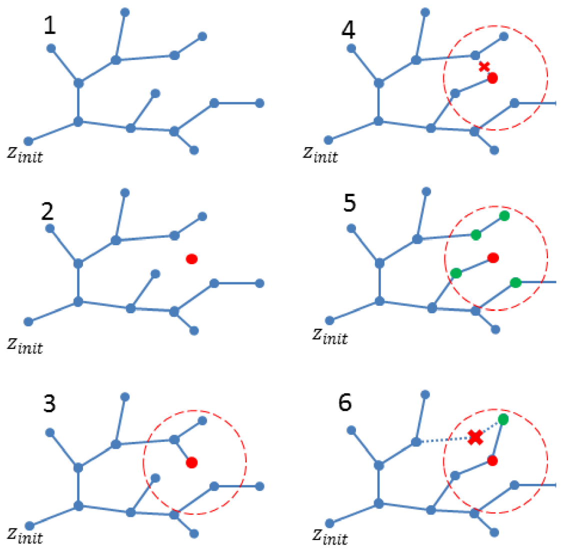

FIGURE 3. Visualization of the MA-RRT*FN for a simple 2D case depiction node insertion, rewiring and removal.

one randomly [6]. In case no nodes are deleted in *EXTEND* and *ForceRemoval* function, x_{new} is removed from the tree.

MA-RRT*FN has the same GREEDY procedure as MA-RRT*. In the GREEDY procedure, the joint state is decomposed into n single-agent states. Thus, the algorithm can steer each agent from its start node **s** to the destination **d** for one timestep separately by merely depending on heuristic guided search, which utilizes Euclidean distance as the metric, and then check the path of agents conflicting or not. If those paths are conflicted, the algorithm will return the path calculated in the prior timestep; if not, the algorithm will check whether all agents reach the target. If they do, the algorithm will return the path of all agents as a series of joint transitional states between **s** and **d**, forming an edge in the tree. If the goal is not attained and the cost of paths exceeds the user-specified threshold c_{max}, the algorithm will return the path between the **s** and the currently arrived node.

Both MA-RRT* and MA-RRT*FN evenly sample the random states in agents' joint configuration space, which would cause a relatively low convergence rate. To improve the speed of MA-RRT*FN in finding the solution, we take the idea from isMA-RRT*, the improved version of MA-RRT* proposed in [4]. The improved algorithm is called informed sampling MA-RRT*FN (isMA-RRT*FN), shown in Algorithm 5, which runs G-RRT* for every single agent to find some high-quality solutions and then runs MA-RRT*FN for all agents together with biased sampling, which samples states near the single-agent optimal path.

V. EXPERIMENTS AND RESULTS

This paper chooses MA-RRT* and isMA-RRT* as the benchmark and compares the ability of MA-RRT*, MA-RRT*FN, isMA-RRT* and isMA-RRT*FN in terms of performance, convergence rate and memory cost. All experiments are performed on *matlab 2018a 64-bit* in a common program framework and tested on *intel core i7 8700k* 3.7 GHz CPU.

To make a fair comparison between these four algorithms, this paper utilizes the problem instance set of [4], mentioned as follows: The agents run in a grid-like square-shaped world, where each agent occupies a single cell. At each timestep, all agents can stay on the cell waiting for other agents or move to the 4-neighborhood cell of its location if these cells are free. Ten percent of the grids are removed to represent obstacles or barriers. A unique start location and destination are selected randomly for every agent.

The problem instance set varies in the following two parameters. The grid sizes: 10×10, 30×30, 50×50, 70×70, 90×90. Number of agents: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, which are the same as in [4]. The two parameters are combined in each grid size and number of agents. For each combination, this paper randomly sets 120 instances based on different obstacles and destinations of agents. Therefore, the first experiment contains 6000 different problem instances in total. All algorithms are implemented on the same instance set, and the runtime of each instance is limited to 5 seconds. For MA-RRT*FN and isMA-RRT*FN algorithm, the maximum number of nodes was set to 200. To speed the procedure of spanning towards the target, all algorithms choose the final goal state as the new random sample with the probability

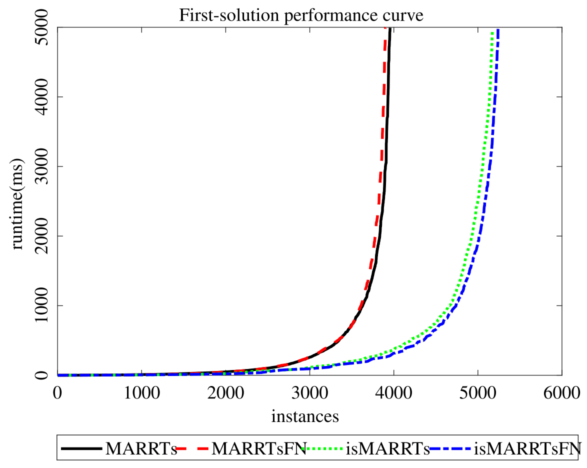

FIGURE 4. Performance curve.

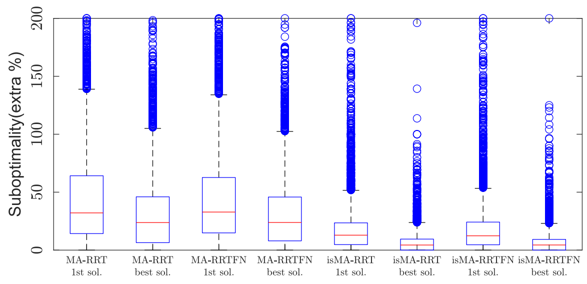

FIGURE 5. Suboptimality.

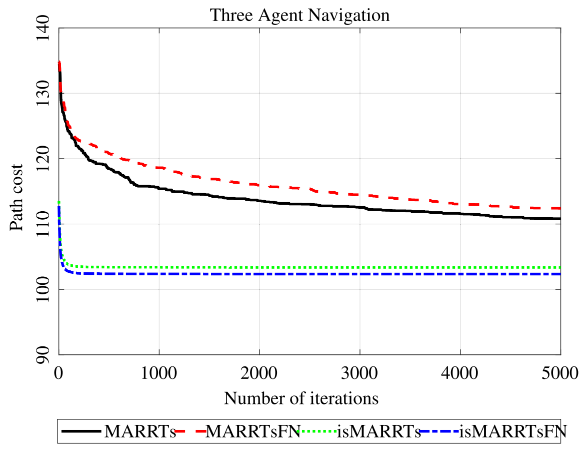

FIGURE 6. Solution quality.

of **p**, which is the user-specified parameter in the sampling procedure.

One more thing we need to clarify is that we tried various different maximum numbers of nodes, ranging from 20 to 2000 in the experiment. The results show that the different parameters have little influence on the quality of the final solution and the speed of finding the first solution under the settings of this experiment. We finally chose 200 as the uniform parameter for different scales of configuration space (ranging from 1 to 10 agents in cooperative navigation). The reason is that we found that when the maximum number of nodes in the tree increases to a certain level (approximately $>= 100$), the continuing increase of nodes did not help to improve the final solution a lot (compare to the unlimited version), but cost a lot of memory.

The results are plotted in Fig. 4 and Fig. 5. In Fig.4, the values in the x-axis are the index of instances which are sorted according to the runtime needed when the first valid solution is found. The values in the y-axis are the runtime when the algorithm finds the first solution. For each algorithm, the ordering can be different. The last point of x-position in the performance curve indicates how many instances are solved within 5 seconds. It can be seen that MA-RRT* resolves 66% of the instances, MA-RRT*FN 65%, is MA-RRT* 86% and isMA-RRT*FN 87%, from the problem instance set. It can be seen that MA-RRT*FN and isMA-RRT*FN come close to MA-RRT* and isMA-RRT* respectively in terms of the performance.

The relative solution quality is shown in Fig. 5. The experiment compares all algorithms in terms of the first returned solution and the best solution found within 5 seconds runtime limit. The suboptimality is calculated by the following formula:

$$\text{suboptimality} = \left(\frac{\text{the cost of returned solution}}{\text{the cost of optimal solution}} - 1\right) \cdot 100. \quad (1)$$

As shown in Fig. 5, the suboptimality of first and best solution of MA-RRT*FN and isMA-RRT*FN are very close to MA-RRT* and isMA-RRT*FN respectively, which indicates that the convergence rate of MA-RRT*FN and isMA-RRT*FN comes close to MA-RRT* and isMA-RRT*FN.

In order to further visualize the memory cost and convergence process, we did a second experiment which shows their gap in a specific instance in which 3 agents navigate in a 30×30 grid. This problem instance set also randomly sets 120 instances based on different obstacles and locations of agents. To facilitate the observation of solution quality corresponding to the different iterations, we extend the running time and utilize the iterations as the running time criterion. All algorithms will be terminated as the iterations reach a threshold. The iterations of each instance are limited to 5000. For MA-RRT*FN and isMA-RRT*FN, the maximum number of nodes is set to 1000.

Fig.6 and Fig.7 shows the average minimum path cost and the average number of nodes in the tree versus the iterations of all algorithms in terms of the solutions of 120 instances, respectively. The first point of the x-position in the path cost curve can be interpreted as the solution quality at the first iteration. The last point of the x-position in the path cost curve indicates the final solution quality after 5000 iterations. It can be observed from Fig.6 that MA-RRT*FN has a similar convergence rate to MA-RRT* while its number of nodes in the tree is much less. As shown in Fig.7, memory required for MA-RRT* grows linearly with the iterations increase, while

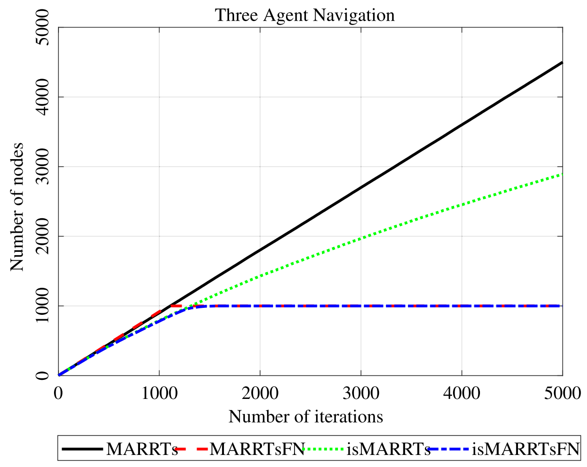

FIGURE 7. **Memory required.**

the number of nodes stored in MA-RRT*FN is much lower and fixed after the iterations reach about 1000 (which is to say the number of nodes in MA-RRT*FN reaches the maximum).

The results also indicate that the isMA-RRT*FN performs better than isMA-RRT* concerning the convergence rate to the optimal path, while it also has a lower and fixed memory. Finally, MA-RRT* is proved to be convergent in [4]. Although the experimental results strongly imply that the MA-RRT*FN and isMA-RRT*FN also have the theoretical guarantee of converging to the optimal path, the optimality of MA-RRT*FN and isMA-RRT*FN remains to be proved theoretically.

VI. CONCLUSION AND FUTURE WORK

This paper proposes MA-RRT*FN, an anytime algorithm that has lower demands on the memory requirements, to solve the multi-agent path planning problem in the systems with limited storage. Unlike MA-RRT*, whose memory cost is indefinite as the solution converges to the optimal path, our techniques employ some node removing procedures to limit the number of nodes stored in the tree and keep on optimizing the path when finding the solution in agents' joint-state space. We compare the capability of our algorithm with MA-RRT* and isMA-RRT*. The experimental results show that the MA-RRT*FN, which has a fixed number of nodes in the tree, performs as well as MA-RRT* in terms of scalability, solution quality and convergence rate in solving multi-agent path planning problems. Besides, the improved version, isMA-RRT*FN, has a better convergence rate and scalability than isMA-RRT* while its memory requirement is much lower and fixed.

This paper simulates the algorithm on a motion graph, which connects the states in the tree by a valid path. However, the algorithm can also be extended to continuous space by using the straight-line visibility approach in place of the GREEDY function.

In the future, we will continue to explore the theoretical proof of the convergence rate of MA-RRT*FN and the tradeoff between the performance and the maximum nodes. Another area we would like to explore is the application of the MA-RRT*FN algorithm in a more dense environment.

REFERENCES

[1] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Dec. 1968.

[2] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; PSPACE- hardness of the 'Warehouseman's Problem," *Int. J. Robot. Res.*, vol. 3, no. 4, pp. 76–88, Dec. 1984.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[4] J. Vokrãnek, and M. Pächouäek, "Multi-agent RRT: Sampling-based cooperative pathfinding," in *Proc. Int. Conf. Autonomous*, 2013, pp. 1263–1264.

[5] P. Verbari, L. Bascetta, and M. Prandini, "Multi-agent trajectory planning: A decentralized iterative algorithm based on single-agent dynamic RRT," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2019, pp. 1977–1982.

[6] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2013, pp. 354–359.

[7] J. Jiang and K. Wu, "Multi-agent path planning based on ma-rrt fixed nodes," in *Proc. 19th Int. Conf. Auto. Agents MultiAgent Syst.*, pp. 1875–1877, 2020.

[8] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.

[9] T. S. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 668–673.

[10] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1163–1177, Oct. 2016.

[11] E. Erdem, D. G. Kisa, U. Oztok, and P. Schüller, "A general formal framework for pathfinding problems with multiple agents," in *Proc. Conf. Artif. Intell.*, Jun. 2013, pp. 1–8.

[12] S. Carpin and E. Pagello, "On parallel RRTs for multi-robot systems," in *Proc. 8th Conf. Italian Assoc. Artif. Intell.*, 2018, pp. 834–841.

[13] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Citeseer, Tech. Rep., 1998.

[14] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 5369–5375.

[15] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Dec. 1996.

[16] D. Silver, "Cooperative pathfinding," in *Proc. AIIDE*, vol. 1, 2005, pp. 117–122.

[17] A. Geramifard, P. Chubak, and V. Bulitko, "Biased Cost Pathfinding," in *Proc. AIIDE*, Marina Del Rey, CA, USA, 2018, pp. 112–114.

[18] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams with complex constraints," *Auto. Robots*, vol. 32, no. 4, pp. 385–403, May 2012.

[19] R. Regele and P. Levi, "Cooperative multi-robot path planning by heuristic priority adjustment," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 5954–5959.

[20] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 430–435.

[21] M. Ragaglia, M. Prandini, and L. Bascetta, "Multi-agent poli-rrt," in *Proc. Int. Workshop Model. Simul. Auto. Syst.*, 2016, pp. 261–270.

[22] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, Feb. 2015.

[23] E. Boyrasky, A. Felner, G. Sharon, and R. Stern, "Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding," in *Proc. Int. Conf. Autom. Planning Schedule*, 2015, pp. 47–51.

[24] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding (MAPF) with continuous time," 2019, *arXiv:1901.05506*. [Online]. Available: https://arxiv.org/abs/1901.05506

[25] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artif. Intell.*, vol. 219, pp. 1–24, Feb. 2015.

[26] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Meta-agent conflict-based search for optimal multi-agent path finding," in *Proc. SoCS*, 2012, pp. 39–40.

[27] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony, "ICBS: Improved conflict-based search algorithm for multi-agent pathfinding," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 1–5.

JINMINGWU JIANG received the B.S. degree from the Department of Mechanical and Power Engineering, Chongqing University of Science and Technology, Chongqing, China, in 2016. He is currently pursuing the M.S. degree with the School of Computer Science, Chongqing University, China. His main research interests include mobile robot motion planning and multi-agent motion planning.

KAIGUI WU received the bachelor's degree from the Department of Mathematics, Sichuan Normal University, in 1989, the master's degree from the Department of Computer Science, Chongqing University, in 1993, and the Ph.D. degree in power system and automation from the School of Electrical Engineering, Chongqing University, in December 1999. He is currently a Professor and a Doctoral Tutor with Chongqing University. His research interests include cryptography and information security.