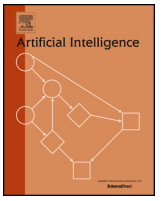
Artificial Intelligence 329 (2024) 104084

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



A stochastic process approach for multi-agent path finding with non-asymptotic performance guarantees

Xiaoyu He [a,b], Xueyan Tang [c,*], Wentong Cai [c], Jingning Li [d]

[a] Singtel Cognitive and Artificial Intelligence Lab for Enterprises, Nanyang Technological University, Singapore
[b] School of Software Engineering, Sun Yat-sen University, China
[c] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[d] NCS Pte Ltd, Singapore

ARTICLE INFO

Keywords:
Multi-agent path finding
Non-asymptotic performance
Stochastic process
Multi-armed bandit problem

ABSTRACT

Multi-agent path finding (MAPF) is a classical NP-hard problem that considers planning collision-free paths for multiple agents simultaneously. A MAPF problem is typically solved via addressing a sequence of single-agent path finding subproblems in which well-studied algorithms such as A* are applicable. Existing methods based on this idea, however, rely on an exhaustive search and therefore only have asymptotic performance guarantees. In this article, we provide a modeling paradigm that converts a MAPF problem into a stochastic process and adopts a confidence bound based rule for finding the optimal state transition strategy. A randomized algorithm is proposed to solve this stochastic process, which combines ideas from conflict based search and Monte Carlo tree search. We show that the proposed method is almost surely optimal while enjoying non-asymptotic performance guarantees. In particular, the proposed method can, after solving N single-agent subproblems, produce a feasible solution with suboptimality bounded by $\mathcal{O}(1/\sqrt{N})$. The theoretical results are verified by several numerical experiments based on grid maps.

1. Introduction

Multi-agent path finding (MAPF) is the task of moving a team of m agents in a common workspace from their source positions to some designated goal positions [1,2]. In its basic form, the object is to optimize the total cost of the planned paths while avoiding collisions with environmental obstacles and among agents. This work concerns the discrete-time case in which the path of each agent is a sequence of location-time pairs. We also assume that the environmental obstacles and source/goal positions are known a priori, so the paths of agents can be planned in an offline manner.

The special case of MAPF with $m = 1$, known as the single-agent path finding (SAPF), has a long history and is considered solved nowadays. Several algorithms for SAPF exist (A* [3] being the most remarkable example) and enjoy both polynomial time complexity and provable optimality. The generic case with $m > 1$, however, has been proved to be NP-hard [4]. Lying at the heart of this hardness is that the possible inter-agent collisions can grow exponentially fast when m increases. Despite this, MAPF has found a wide range of applications in fields like automated warehouses [5], autonomous vehicles [6], and video games [7], as it offers a natural way for modeling many complex control systems.

* Corresponding author.
E-mail addresses: hexy73@mail.sysu.edu.cn (X. He), asxytang@ntu.edu.sg (X. Tang), aswtcai@ntu.edu.sg (W. Cai), jingning.li@ncs.com.sg (J. Li).

https://doi.org/10.1016/j.artint.2024.104084
Received 4 July 2022; Received in revised form 18 January 2024; Accepted 22 January 2024

Available online 1 February 2024

A variety of algorithms for solving MAPF problems have been developed [8–14]. Early works are mostly based on pre-defined motion rules (e.g., [8,10]) or combinatorial search schemes (e.g., [14,13]). Recent algorithms typically rely on iteratively solving a sequence of SAPF subproblems, which have more grounded theoretical basis and better scalability. The conflict-based search (CBS) [9] is a popular example of this class of algorithms. CBS uses a *conflict* to describe when, where, and to which agents a collision happens, and resolves the conflict via re-planning the two involved agents individually. This allows bipartitioning the search space and incrementally updating the solution in the half space with any SAPF solver. The idea behind CBS is intuitive: the inter-agent collisions can usually be avoided after resolving only a few conflicts; so sequentially checking the conflicts in a well-designed order might approach the solution quickly. This intuition has been supported by a lot of empirical evidence [15]. Several reliable heuristics [16–20] have also emerged recently and improved CBS via adjusting the order of conflicts to be resolved. In general, state-of-the-art CBS implementations usually have good empirical performance, scaling well even with hundreds of agents in certain scenarios.

Compared to the progress made in the empirical aspect, the theoretical understanding towards MAPF algorithms is still incomplete. Ideally, for MAPF algorithms based on iteratively solving SAPF subproblems, one would expect a performance guarantee measured as

$$\delta_N := \min_{1\le n\le N} \left\{ \tilde{F}(X_n) - F_* \right\} \text{ with } \tilde{F}(X_n) = \begin{cases} F(X_n) & \text{if } X_n \text{ is feasible,} \\ +\infty & \text{otherwise,} \end{cases} \tag{1}$$

where X_n is an intermediate solution obtained after the n-th iteration, $F(\cdot)$ is the objective function of the MAPF problem, and F_* is the optimal objective value. We say a MAPF algorithm is *asymptotically optimal* if it satisfies $\lim_{N\to\infty} \delta_N = 0$. If, in addition, there exists some function $B(N)$ such that

$$\delta_N \le B(N) = o(1), \quad \forall N \ge 1 \tag{2}$$

holds, then we say this algorithm achieves *non-asymptotic convergence*. Here we refer to $B(N)$ as the *non-asymptotic bound* or *convergence rate*. A MAPF algorithm achieving non-asymptotic convergence ensures producing inexact solutions at any time, while preserving the ability to further improve the solutions if a larger computation time budget is affordable. The non-asymptotic convergence is a standard tool for evaluating optimization algorithms and has been used for highly related topics such as continuous-time multi-robot control; see [21] for an example. We note, however, that it has not been studied in the MAPF context, probably due to the combinatorial complexity of MAPF problems.

To our knowledge, no existing MAPF algorithm has non-asymptotic convergence guarantees. CBS, for instance, achieves asymptotic optimality only. The lack of non-asymptotic guarantees usually results in the difficulty in monitoring the anytime behavior of an iterative algorithm, as there is no smooth trade-off between the accuracy and tractability. For example, when CBS fails to solve a MAPF problem in a reasonable time and has to be terminated manually, it will return nothing and provide no information about how close we are to the final solution. Anytime algorithms [22–25] and suboptimal CBS variants [26,27,20] partially avoid this issue in practice, but they do not necessarily fit into the rigorous definition of non-asymptotic convergence.

This work focuses on improving CBS in the theoretical aspect and presents the first MAPF algorithm with non-asymptotic convergence guarantees. Our start point is to model the MAPF problem as a stochastic process, where each state is a set of constraints, and its action space is made up of constraints for addressing an unresolved random conflict. For a given state and a constraint chosen as an action, the state transition is achieved by including the constraint into the current constraint set. We show that solving a MAPF problem is equivalent to evaluating the initial state of the stochastic process. We then propose a Monte Carlo constraint tree search (MCCTS) algorithm, which can be considered as a hybrid of CBS and Monte Carlo tree search (MCTS) [28], a classical tree-search based state-value estimation tool for stochastic processes. MCCTS preserves the tree structure of CBS for holding constraints and solutions, but does not use its best-first search scheme or try to evaluate a conflicting solution, which we think are the main barriers towards developing a convergence theorem. On one hand, we propose to use any MAPF solver that can accept a set of constraints as a *simulation* to get the reward of performing an action, and this admits producing one feasible solution at each iteration. On the other hand, we introduce the tree policy and backup mechanisms of MCTS, where the former is used to select the nodes for expansion and the latter is to collect necessary statistics for guiding the search. The key challenge here is that the action space is itself stochastic and the rewards of performing actions are non-stationary across iterations; to the best of our knowledge, there exists no rigorous analysis for methods in such settings. We overcome this by designing a new bandit strategy for the tree policy and deriving finite-time convergence guarantees for the proposed method. The main contributions of this work are highlighted as below:

- We show that any solvable MAPF problem can be reduced to a state value evaluation task on a stochastic process.
- We show MCCTS is almost surely optimal. In addition, the following holds with overwhelming probability:

$$\min_{1\le n\le N} F(X_n) - F_* = \mathcal{O}\left(\frac{1}{\sqrt{N}}\right), \tag{3}$$

  where X_n is a feasible solution produced at the n-th iteration, F is the objective function, and F_* is the optimal objective value.
- We discuss two different methods for implementing a simulation. The first uses existing MAPF algorithms, showing the potential of MCCTS as a generic MAPF framework. The second is novel, based purely on randomized schemes. We use this method to provide the first non-asymptotic bound on the complexity for solving MAPF problems, measured by the number of calls to SAPF solvers.

In the remainder of this paper, we firstly define the problem in Section 2 and review related works in Section 3. Preliminaries on MCTS are discussed in Section 4. Then, in Section 5, we show how to model a MAPF problem as a stochastic process. The detailed algorithm for state value evaluation with this stochastic process is described in Section 6 and empirically verified in Section 7. We conclude this paper and give some remarks in Section 8.

Notations Solutions of a MAPF problem are written as capital letters, X. The path of a particular agent in a solution is written in bold lowercase, $\boldsymbol{x}$, with an optional subscript to denote the index of that agent. Locations of a path are in usual lowercase with a superscript specifying the time step, $x^{[t]}$. We use $|\boldsymbol{x}|$ to denote the length of a path $\boldsymbol{x}$. $\mathbb{E}[\cdot]$ denotes the expectation. $\mathbb{P}\{\cdot\}$ denotes the probability of an event. $\mathbb{I}\{\cdot\}$ denotes the indicator function of an event. $\mathbb{N}$ denotes the non-negative integer set $\{0, 1, 2, \dots\}$. Sets are written in calligraphic font, $\mathcal{A}$, and we use $|\mathcal{A}|$ to denote the cardinality of a set $\mathcal{A}$.

2. Problem definition

The MAPF problem under consideration follows the definition in [1]. We assume each motion of an agent takes one unit of time. So, for each agent i, its path can be formulated as a sequence

$$\boldsymbol{x}_i = (x_i^{[0]}, x_i^{[1]}, \dots),$$

where $x_i^{[j]}$ is its location at time step j and the length of the path needs to be determined together with the locations. We define its cost as the time elapsed before the path is ended, given by

$$f(\boldsymbol{x}_i) = |\boldsymbol{x}_i| - 1,$$

where $|\boldsymbol{x}_i|$ is the length of $\boldsymbol{x}_i$.

We parameterize the workspace of agents using a directed graph $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the vertex set defining all non-obstacle locations, and $\mathcal{E}$ is the edge set characterizing all possible motions which the agents can take.[1] Given the corresponding source locations $\{s_i\}_{i=1}^m \subset \mathcal{V}$ and goal locations $\{g_i\}_{i=1}^m \subset \mathcal{V}$, a MAPF problem with m agents is to find a path set $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ via solving the following problem:

$$\begin{aligned}
&\min\ F(X) = \sum_{i=1}^{m} f(\boldsymbol{x}_i), \\
&\text{s.t. } x_i^{[0]} = s_i, x_i^{[|\boldsymbol{x}_i|-1]} = g_i, && \forall i \in \{1, \dots, m\}, \\
&\quad (x_i^{[j-1]}, x_i^{[j]}) \in \mathcal{E}, && \forall i \in \{1, \dots, m\}, j \in \{1, \dots, |\boldsymbol{x}_i|\}, \\
&\quad C(\boldsymbol{x}_i, \boldsymbol{x}_j) = \emptyset, && \forall i, j \in \{1, \dots, m\}, i \neq j,
\end{aligned} \tag{4}$$

where F is the objective function which is the sum of costs of all the paths, and $C(\boldsymbol{x}_i, \boldsymbol{x}_j)$ denotes the set of all conflicts between agents i and j. The conflict set is given by $C(\boldsymbol{x}_i, \boldsymbol{x}_j) = C_v(\boldsymbol{x}_i, \boldsymbol{x}_j) \cup C_e(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where

$$\begin{aligned}
C_v(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \left\{ t \middle| 0 \le t \le \max\{|\boldsymbol{x}_i|, |\boldsymbol{x}_j|\}, x_i^{[\min\{t, |\boldsymbol{x}_i|\}]} = x_j^{[\min\{t, |\boldsymbol{x}_j|\}]} \right\}, \\
C_e(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \left\{ t \middle| 1 \le t \le \min\{|\boldsymbol{x}_i|, |\boldsymbol{x}_j|\}, x_i^{[t]} = x_j^{[t-1]} \neq x_i^{[t-1]} = x_j^{[t]} \right\}.
\end{aligned}$$

The set $C_v(\boldsymbol{x}_i, \boldsymbol{x}_j)$ contains all time steps when a *vertex conflict* happens, i.e., two agents occupy the same location at the same time. It also captures the case in which an agent blocks another after reaching its goal location. The set $C_e(\boldsymbol{x}_i, \boldsymbol{x}_j)$ includes the time steps of all *edge conflicts* in which two agents try to switch their locations. This definition of conflicts is by convention and can be generalized easily to describe other types of inter-agent collisions. Note that for a pair of agents i and j, a vertex conflict and an edge conflict cannot happen simultaneously at a single time step. Throughout this work we assume the MAPF problem is solvable.

3. Related work on MAPF

The standard implementation of CBS [9] performs a best-first search on a tree where each node contains a set of constraints and a solution that is consistent with these constraints. To make the tree nodes comparable, CBS defines a partial order based on the objective values of their associated solutions. This partial order is sometimes further enhanced via incorporating admissible heuristics [17,18] or even inadmissible heuristics [26] (for tie-breaking). This design has an advantage: once CBS finds a feasible solution, it can terminate immediately as the solution found is guaranteed to be optimal. The price to pay for this is that CBS does not guarantee to produce a feasible solution within a polynomial number of calls to SAPF subproblem solvers. This disadvantage is caused by the attempt to evaluate partial solutions (those having collisions), and cannot be avoided by merely using heuristics. Our work addresses this via introducing the concept of simulation, which directly generates a feasible solution that can be output at each iteration.

[1] The set $\mathcal{E}$ may include an edge from each vertex to itself; this admits modeling the action that an agent may wait at some location for avoiding potential conflicts.

Suboptimal CBS variants, such as the extended CBS (ECBS) [27], provide a way to balance the accuracy and complexity. These methods require a user-defined parameter $\epsilon > 1$ (called the suboptimal bound hereinafter) and can produce a solution with the objective value no greater than ϵF_*. Suboptimal CBS variants can spend much less computation time than CBS if ϵ is set properly. But this improvement has no provable guarantees; an improper parameter setting could, in fact, lead to longer computation time. Moreover, the correlation between the suboptimal bound ϵ and the solution quality is usually unknown and seems to be problem-dependent, implying that decreasing ϵ does not always monotonically increase the solution accuracy. Hence, users have to spend extra effort in tuning this parameter with no predictable benefit.

There exist several anytime MAPF algorithms [29,23–25] sharing similar ideas to our proposals. These methods usually produce an initial solution quickly and then iteratively improve it by solving a sequence of MAPF subproblems. To achieve a smoother trade-off between accuracy and tractability, the subproblems are designed to be much simpler than the original one, via techniques such as bounding the graph size or relaxing the suboptimal bound. Existing anytime algorithms, however, do not enjoy non-asymptotic guarantees, since each iteration of improvement is still NP-hard.

The online control of multiple agents via stochastic processes has been studied in [30,31]. The state of the agents is observed from interactions with the environment and their motions are planned using Markov processes. These methods have no optimality guarantees and their applicable regions are slightly different from the MAPF problem discussed in this work. Our work differs from these studies in the state modeling strategy: in our work the state is based on constraints rather than locations. As we will see, this difference leads to a more grounded theoretical foundation.

4. Preliminaries on MCTS

The proposed MCCTS method leverages the MCTS approach, and adapts it for the MAPF problem. Therefore, we present in this section an overview of MCTS and briefly describe its mainstream implementations.

MCTS is a general approach for handling Markov decision processes (MDPs) via Monte Carlo simulations [32]. The method builds a tree incrementally, and uses nodes/edges to store the state/action information of the decision sequence. At each iteration, a tree policy is employed to find the most urgent node using information collected in previous iterations. Then, a simulation is run from the selected node, and the tree is updated according to the simulation result. This typically involves the addition of a child node and an update to the statistics of all ancestor nodes. The most distinguished feature of MCTS is that state transactions during the simulation are made according to some pre-defined Monte Carlo sampling rule, called default policy, which does not need to evaluate the values of intermediate states. This reduces the amount of domain-specific knowledge in decision making, making MCTS very suitable for situations where the value evaluation can only be performed on terminal states.

A standard MCTS implementation is the UCT algorithm [28]. It models the node-selection procedure in the tree policy as a multi-armed bandit problem, and applies the upper confidence bound (UCB) method to solve it [33]. In UCT, each node v has two fields $v.T$ and $v.R$, which record respectively the number of times v has been visited and the estimated value of the state. For a reward-maximization problem, the child node is selected according to the rule

$$\underset{v' \text{ is a child of } v}{\arg\max} \; v'.R + \alpha \sqrt{\frac{\ln v.T}{v'.T}},$$

where α is a hyperparameter. This is known as UCB1 [34], a representative in the UCB family of methods for handling multi-armed bandits. UCT has been shown to provide a state value estimator with errors diminishing at a rate of $\mathcal{O}(1/N)$ [28], where N is the number of iterations. This result, however, relies on exponential concentration of rewards, which is unlikely to hold in practice as the simulations are non-stationary across multiple iterations [35].

To better handle the non-stationary issue, Shah et al. [36] suggested to use a corrected node-selection rule as

$$\underset{v' \text{ is a child of } v}{\arg\max} \; v'.R + \alpha \sqrt{\frac{\sqrt{v.T}}{v'.T}}, \tag{5}$$

and proved that UCT with this new rule achieves an estimation error of $\mathcal{O}(1/\sqrt{N})$. This result is worse than the one given in [28], but it alleviates the exponential concentration assumption. The limitation is, however, that the hyperparameter α used depends on problem-specific knowledge; this implies the new rule may not generalize well on different problems. Moreover, the work [36] did not provide a closed-form setting of α, so how to use the new rule in practice deserves further investigation.

5. From MAPF to stochastic processes

We introduce in this section the *MAPF process*, a Markov stochastic process built from a given MAPF problem. We show that solving the MAPF problem is equivalent to evaluating the value of the initial state in the corresponding MAPF process.

5.1. Constraints, symmetric constraints, and constrained planner

We first discuss several basic concepts. We define two types of constraints, namely the vertex constraint and the edge constraint, to resolve a vertex conflict and an edge conflict respectively. A vertex constraint is a 3-tuple taking the form of

(i,t,l) (Vertex constraint)

where $i \in \{1,\dots,m\}, t \geq 0$, and $l \in \mathcal{V}$. When re-planning agent i, we use this constraint to prevent it from occupying the location l at time step t, which will resolve a vertex conflict of agent i with another agent. Similarly, an edge constraint is characterized by a 4-tuple

(i,t,l,f) (Edge constraint)

where $i \in \{1,\dots,m\}, t \geq 0, l \in \mathcal{V}, f \in \mathcal{V}, (f,l) \in \mathcal{E}$, and $f \neq l$, which is designed to prohibit agent i from reaching l from f at time step t.

For a given solution containing conflicting paths, we can usually generate several different constraints to resolve the conflicts. Among these constraints, we are particularly interested in those satisfying the following condition:

Definition 1 *(Symmetric constraint set).* Let $X = \{\boldsymbol{x}_1,\dots,\boldsymbol{x}_m\}$ be a MAPF solution having conflicts and $\mathcal{A}$ be a non-empty set of constraints. We say the constraint set $\mathcal{A}$ is symmetric with respect to X if the following hold:

- For any vertex constraint $(i,t,l) \in \mathcal{A}$, there exists a vertex constraint $(j,t,l) \in \mathcal{A}$ where $i \neq j$ and $t \in C_v(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- For any edge constraint $(i,t,l,f) \in \mathcal{A}$, there exists an edge constraint $(j,t,f,l) \in \mathcal{A}$ where $i \neq j$ and $t \in C_e(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

For convenience, we also define that an empty constraint set $\emptyset$ is symmetric with respect to solutions that have no conflict.

The above definition requires that the constraints must come in pairs. For example, to resolve a vertex conflict between agents i and j at location l and time step l, one should involve (i,t,l) and (j,t,l) simultaneously in forming the constraint set. This allows for bipartitioning the search space via re-planning agents i and j individually, since no feasible solution can violate these two constraints at the same time.

To utilize the constraints defined above, we need a solution generator which can produce an individual path for each agent that is consistent with a given set of constraints.

Definition 2 *(Constrained planner).* A constrained planner is a function $\mathcal{P}$ receiving a constraint set $\mathcal{S}$ and producing a solution $\mathcal{P}(\mathcal{S}) = X = \{\boldsymbol{x}_1,\dots,\boldsymbol{x}_m\}$, where $\boldsymbol{x}_i$ is an optimal solution to the SAPF problem for agent i, subject to the constraints in $\mathcal{S}$. If some $\boldsymbol{x}_i$ does not exist, we say $\mathcal{P}(\mathcal{S})$ is invalid.

Note that the constrained planner introduced here is for theoretical analysis, and will not be constructed explicitly in practical implementation.

5.2. Stochastic process modeling

We treat solving a MAPF problem as a sequential decision-making task, where at each step a constraint needs to be chosen in order to resolve some conflict. In existing CBS implementations, the conflicts to be resolved are selected by some pre-defined rule, e.g., in the natural order or according to some heuristics. In this work, the order of resolving conflicts is simply random. This leads to a stochastic process over the space of constraints, which we call a MAPF process.

Definition 3 *(MAPF process).* A MAPF process is a tuple $\left(\mathcal{P}, \{(\mathcal{S}_l, \mathcal{A}_l)\}_{l\in\mathbb{N}}\right)$ satisfying the following conditions:

1. $\mathcal{P}$ is a constrained planner.
2. Each $\mathcal{S}_l$ is a set of constraints and $\mathcal{S}_0 = \emptyset$.
3. Each $\mathcal{A}_l$ is a randomly generated constraint set that is symmetric with respect to $\mathcal{P}(\mathcal{S}_l)$.
4. The sequence $\{(\mathcal{S}_l, \mathcal{A}_l)\}_{l\in\mathbb{N}}$ terminates at step l' if $\mathcal{P}(\mathcal{S}_{l'})$ is invalid or has no conflict; otherwise, $\mathcal{S}_{l'+1} = \mathcal{S}_{l'} \cup \{c\}$ for some constraint $c \in \mathcal{A}_{l'}$.

Hereinafter we call $\mathcal{S}_l$ a state and $\mathcal{A}_l$ its associated action set. The elements in an action set are called actions. $\mathcal{S}_0$ is called the initial state. $\mathcal{S}_{l'}$ is called the terminal state if the sequence $\{(\mathcal{S}_l, \mathcal{A}_l)\}_{l\in\mathbb{N}}$ terminates at step l'.

The MAPF process is a special instance of Markov processes. Specifically, each state $\mathcal{S}_l$ is a set of constraints and is associated with a solution generated by a constrained planner, $\mathcal{P}(\mathcal{S}_l)$. The action space of $\mathcal{S}_l$, $\mathcal{A}_l$, could be stochastic but must be symmetric with respect to $\mathcal{P}(\mathcal{S}_l)$. An action $c \in \mathcal{A}_l$ is a constraint and we perform it by including it into $\mathcal{S}_l$, causing a transition from $\mathcal{S}_l$ to a new state $\mathcal{S}_{l+1} = \mathcal{S}_l \cup \{c\}$. In the case that $\mathcal{A}_l$ is empty, which happens when $\mathcal{P}(\mathcal{S}_l)$ is invalid or has no conflict, the state $\mathcal{S}_l$ is considered a terminal state.

To capture the relation between the MAPF process and the original MAPF problem in (4), we need to explicitly measure the quality of the states in the MAPF process. This is achieved by introducing the value function.

Definition 4 *(Value function).* The value function of a MAPF process $\left(\mathcal{P}, \left\{(S_l, \mathcal{A}_l)\right\}_{l\in\mathbb{N}}\right)$ is a function U defined recursively by

$$U(S_l) = \begin{cases} \mathbb{E}\left[\min\limits_{c\in\mathcal{A}_l} U(S_l \cup \{c\})\right] & \text{if } \mathcal{P}(S_l) \text{ is valid but has conflicts,} \\ F(\mathcal{P}(S_l)) & \text{if } \mathcal{P}(S_l) \text{ is valid and has no conflicts,} \\ \infty & \text{if } \mathcal{P}(S_l) \text{ is invalid,} \end{cases} \tag{6}$$

where $\mathbb{E}$ is the expectation taken over $\mathcal{A}_l$.

Directly computing the expectation in $U(S_l)$ is intractable. In the following we define a helper function to address this.

Definition 5 *(Conditioned value function).* Consider a MAPF process $\left(\mathcal{P}, \left\{(S_l, \mathcal{A}_l)\right\}_{l\in\mathbb{N}}\right)$. Define the value function for a state S_l conditioned on $\mathcal{A}_l$ as

$$V(S_l; \mathcal{A}_l) = \begin{cases} \min\limits_{c\in\mathcal{A}_l} U(S_l \cup \{c\}) & \text{if } \mathcal{P}(S_l) \text{ is valid but has conflicts,} \\ F(\mathcal{P}(S_l)) & \text{if } \mathcal{P}(S_l) \text{ is valid and has no conflicts,} \\ \infty & \text{if } \mathcal{P}(S_l) \text{ is invalid.} \end{cases} \tag{7}$$

When the context is clear, we will call both $V(S_l; \mathcal{A}_l)$ and $U(S_l)$ the value of S_l.

By comparing (6) and (7), we can observe that

$$U(S_l) = \mathbb{E}\left[V(S_l; \mathcal{A}_l)\right]. \tag{8}$$

To see how this helps in quantifying a MAPF process, substitute (8) into the first case of (7) and it follows that

$$V(S_l; \mathcal{A}_l) = \min_{c\in\mathcal{A}_l} \mathbb{E}\left[V(S_l \cup \{c\}; \mathcal{A}_{l+1})\right] \tag{9}$$

holds if $\mathcal{P}(S_l)$ is valid but has conflicts, where $\mathcal{A}_{l+1}$ is the action set associated with the state $S_l \cup \{c\}$, and the expectation here is taken over $\mathcal{A}_{l+1}$. Equation (9) is of particular interest as it is analogous to the *Bellman optimality equation* for characterizing the optimal value function of an MDP [37]. This gives a hint that we can evaluate the state values using tools such as MCTS, which is designed for handling MDPs.

It remains to show the relation between a MAPF problem and the corresponding MAPF process:

Theorem 1. *Given a MAPF problem* (4) *with the optimal objective value* F_*. *The value of the initial state in the MAPF process* $\left(\mathcal{P}, \left\{(S_l, \mathcal{A}_l)\right\}_{l\in\mathbb{N}}\right)$ *satisfies*

$$F_* = U(S_0) = V(S_0; \mathcal{A}_0) \tag{10}$$

for every constraint set $\mathcal{A}_0$ *that is symmetric with respect to* $\mathcal{P}(S_0)$.

Proof. See Appendix A.

Remark 1 *(MAPF as a state value evaluation task).* Theorem 1 shows that every MAPF problem corresponds to a MAPF process and the optimal value of the MAPF problem is exactly the value of the initial state S_0. It means we can solve a MAPF problem via evaluating the value of S_0, i.e., $U(S_0)$. However, as mentioned above, evaluating $U(S_0)$ is still a difficult task as it involves taking expectation over all instances of $\mathcal{A}_0$. The second equality of (10) suggests a way to avoid this issue: we can simply choose any instance of $\mathcal{A}_0$ and use the conditioned value $V(S_0; \mathcal{A}_0)$ instead.

6. MCCTS: a state value estimation framework for MAPF processes

In this section we provide the MCCTS algorithm for evaluating $V(S_0; \mathcal{A}_0)$ given a MAPF process $\left(\mathcal{P}, \left\{(S_l, \mathcal{A}_l)\right\}_{l\in\mathbb{N}}\right)$, which is equivalent to solving the corresponding MAPF problem as shown in Theorem 1. We assume every action set $\mathcal{A}_l$ has a fixed size $K \geq 2$ if S_l is nonterminal. This is a generalization to existing CBS implementations where K is fixed to 2.

6.1. Motivation for the algorithm design

The underlying idea of our approach is to use MCTS to handle a given MAPF process as if it was an MDP. This is motivated by the similarity between the recursive relation (9) and the Bellman equation for MDPs. However, the MAPF process is in fact not an MDP: in the former the action set is itself random whereas in the latter the action set is fixed. Therefore, when performing MCTS

on a MAPF process, the stochasticity comes from not only Monte Carlo simulations but also the action set generation step. To our knowledge, existing analysis methods for MCTS such as [28,36] are not applicable in this setting.

We overcome this difficulty in two steps. 1) For a nonterminal state S_l and its associated action set $\mathcal{A}_l$, we propose to compute $V(S_l;\mathcal{A}_l)$ using Monte Carlo estimations of $U(S_l \cup \{c\})$ for all $c \in \mathcal{A}_l$. We formulate this as a bandit problem in Section 6.2 and prove a nice property: if the error in estimating $U(S_l \cup \{c\})$ is bounded for all $c \in \mathcal{A}_l$, then our computation of $V(S_l;\mathcal{A}_l)$ also has a bounded error. 2) Based on this property, we further use the obtained $V(S_l;\mathcal{A}_l)$ as an estimation of $U(S_l)$ to compute $V(S_{l-1};\mathcal{A}_{l-1})$. This recursive procedure can be naturally formed as a tree search, which is given in Section 6.3. We show that the bounds on the estimation error hold recursively from the tree leaves to the tree root, which then indicates that the value estimation error on the initial state is also bounded. Based on the above ideas, we design the MCCTS algorithm, which inherits the benefits of CBS in maintaining the tree while improving MCTS to handle the randomness of action sets.

6.2. A bandit approach to state value evaluation

We describe how to evaluate the conditioned value for a nonterminal state using Definition 5, which we restate below as

$$V(S;\mathcal{A}) = \min_{c\in\mathcal{A}} U(S \cup \{c\}).$$

Note that in this subsection we hide the subscripts of S and $\mathcal{A}$ for clarity. Our method consists of several rounds where in each round we pick up an action $c \in \mathcal{A}$ and get some estimated value $R \approx U(S \cup \{c\})$. The R values are called *rewards*. The task is to decide how to choose the actions such that the empirical mean of rewards can approach $V(S;\mathcal{A})$. We reduce this task to the classic bandit problem [34] given as below.

Definition 6 *(Bandit problem for state value evaluation)*. Given a state S and the corresponding action set $\mathcal{A} = \{c_1, \dots, c_K\}$ where K is the size of $\mathcal{A}$. Suppose we have several rounds of opportunity to select some actions from $\mathcal{A}$ and, each time when an action is selected, a random reward is revealed. Denote k_n as the index of the action selected at round n. Denote $Y_{N,k}$ as the number of times action c_k has been selected during the first N rounds:

$$Y_{N,k} = \sum_{n=1}^{N} \mathbb{I}\{k_n = k\}. \tag{11}$$

Denote $R_{n,k}$ as the random reward revealed after selecting action c_k for the n-th time. Denote $\bar{R}_{n,k}$ as the average reward of selecting action c_k n times:

$$\bar{R}_{n,k} = \frac{1}{n}\sum_{i=1}^{n} R_{i,k}. \tag{12}$$

Define the average reward of state S over the N rounds of selections as

$$\bar{R}_N = \frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{Y_{N,k}} R_{i,k} = \frac{1}{N}\sum_{k=1}^{K} Y_{N,k}\bar{R}_{Y_{N,k},k}. \tag{13}$$

The bandit problem for estimating the value of S is to find a selection strategy, parameterized by $\{k_1, \dots, k_N\}$, to minimize the quantity

$$\left|\bar{R}_N - V(S;\mathcal{A})\right|.$$

The above bandit problem has been extensively studied under the condition that the rewards are 1) unbiased and 2) independent and independently distributed [34]. Unfortunately, these conditions do not hold in the MCCTS algorithm proposed later, as the rewards are produced from simulations where the underlying distribution is non-stationary across iterations. We address this by providing a novel bandit strategy which bounds the value estimation error up to the first- and second-order moments.

Our method is to select, at the $(n+1)$-th round, the action according to the following strategy:

- Select action c_{n+1} if $n < K$;
- Select the action indexed by

$$k_{n+1} \in \arg\min_{1\le k\le K}\left\{\bar{R}_{Y_{n,k},k} - \alpha\sqrt{\frac{\sqrt{n}}{Y_{n,k}}}\right\} \tag{14}$$

  if $n \ge K$, where α is a hyperparameter.

We call (14) the lower confidence bound (LCB) strategy. It is named because the second term, $\sqrt{\frac{\sqrt{n}}{Y_{n,k}}}$, is used as a confidence radius for bounding the estimation error. In the following, we present the convergence property of this strategy.

Theorem 2. *Given a state S and the corresponding action set $\mathcal{A} = \{c_1, \dots, c_K\}$, consider solving the bandit problem in Definition 6 with the LCB strategy in (14). Define*

$$\Delta_k = U(S \cup \{c_k\}) - \min_{1 \le k \le K} U(S \cup \{c_k\}) = U(S \cup \{c_k\}) - V(S; \mathcal{A})$$

as the gap, in terms of the value, between selecting action c_k and selecting the optimal one. Define

$$\Delta = \min_{k:\Delta_k > 0} \Delta_k$$

as the minimal gap for suboptimal actions. Assume the optimal action is unique, i.e., $|\{k : \Delta_k = 0\}| = 1$. Fix an integer $N' \ge 1$. Let $\beta \ge 0, \sigma \ge 0, \gamma \ge \frac{3}{2}$ be given such that the following hold for all $k \in \{1, \dots, K\}$, $N \in \{1, \dots, N'\}$, and $z \in \left(0, \frac{1}{N'}\right]$:

$$\mathbb{E}\left[\bar{R}_{N,k}\right] - U(S \cup \{c_k\}) \le \frac{\beta}{\sqrt{N}}, \tag{15}$$

$$\mathbb{E}\left[\left(\bar{R}_{N,k} - U(S \cup \{c_k\})\right)^2\right] \le \frac{\sigma^2}{N} z^{\gamma}. \tag{16}$$

Let $\rho > 0$ be given and set α as

$$\alpha = \rho z^{\frac{\gamma}{6} - \frac{1}{4}}. \tag{17}$$

If all rewards are bounded from above by some constant B, i.e.,

$$R_{n,k} \le B, \ \forall n, k,$$

then we have

$$\mathbb{E}\left[\bar{R}_{N'}\right] - V(S; \mathcal{A}) \le \frac{\beta'}{\sqrt{N'}}, \tag{18}$$

$$\mathbb{E}\left[\left(\bar{R}_{N'} - V(S; \mathcal{A})\right)^2\right] \le \frac{(\sigma')^2}{N'} z^{\gamma'} \tag{19}$$

for some constants $\beta', \sigma', \gamma' \ge 0$ satisfying

$$\begin{cases} \beta' & \ge 2B(K-1)\left(1 + 4\left(\frac{\sigma^2}{\rho^2} + \frac{\rho^2}{\Delta^2}\right)\right) + \beta \\ \sigma'^2 & \ge 96B^2(K-1)^2\left(\frac{\sigma^2}{\rho^2} + \frac{\rho^4}{\Delta^4}\right) + 3\sigma^2 \\ \gamma' & = \frac{2}{3}\gamma - 1. \end{cases} \tag{20}$$

Proof. See Appendix B.

Remark 2 *(Convergence in non-stationary settings).* Theorem 2 states that, for a given state, if the gap between the average reward and the state value has bounded first-order and second-order moments for all its immediate successor states, then the bound holds recursively for this state itself. This does not require the exponential concentration assumption on the rewards, which is the main advantage over UCB1, the default bandit strategy for MCTS.

Remark 3 *(Difference to the work [36]).* The LCB strategy is essentially identical to (5) proposed by [36], but a fundamental difference exists in the way of bounding the value estimation error. Specifically, the work [36] directly bounds the divergence probability of the state value estimations, and this requires a priori knowledge of the reward bound B. In contrast, we show the convergence by bounding the moments of value estimation errors. Albeit achieving similar convergence results, our proof is much simpler, and, more importantly, does not need to know the exact value of B. As will be shown, this admits a problem-independent setting for the parameter α that ensures the adaptive convergence of MCCTS.

6.3. Implementation of MCCTS

We now give the implementation details of MCCTS. MCCTS is a tree-based iterative algorithm for estimating the state values of a MAPF process. Each tree node in MCCTS corresponds to a state of the MAPF process, and each edge corresponds to an action. The tree node is also associated with a solution that satisfies the state (which by definition is a constraint set). The solution is the output of the constrained planner applied to the state, and the action set must be symmetric with respect to this solution.

Algorithm 1 MCCTS.

Require: A space-time constrained A* algorithm, a simulation algorithm receiving a set of constraints and producing a feasible solution

for $i \leftarrow 1, \dots, m$ **do** ▷ Initialization
  $\boldsymbol{x}_i \leftarrow$ a shortest path planned for agent i with A*
end for
$root \leftarrow CreateNode(\emptyset, \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}, \emptyset)$
for $n \leftarrow 1, \dots, N$ **do** ▷ Main loop
  $v \leftarrow root$
  while $v.flag =$ "nonterminal" **do** ▷ Tree policy
    if $\exists v' \in v.children$ and $v'.T = 0$ **then**
      $v \leftarrow v'$
    else
      $v \leftarrow \underset{v' \in v.children}{\arg\min}\ v'.R - \alpha \sqrt{\frac{\sqrt{\sum_{u \in v.children} u.T}}{v'.T}}$
    end if
  end while
  if $v.flag =$ "unknown" **then** ▷ Node expansion
    if the solution in $v.paths$ has no conflict **then**
      $\mathcal{A} \leftarrow \emptyset$
      $v.flag \leftarrow$ "feasible"
    else
      $\mathcal{A} \leftarrow$ a random constraint set that is symmetric with respect to $v.paths$
      $v.flag \leftarrow$ "nonterminal"
    end if
    $\mathcal{S} \leftarrow$ {all constraints stored in v and its predecessors}
    for $c \in \mathcal{A}$ **do**
      $i \leftarrow$ index of the agent involved in c
      $\boldsymbol{x} \leftarrow$ a shortest path planned for agent i with A* subject to constraints in $\mathcal{S} \cup \{c\}$
      if $\boldsymbol{x}$ exists **then**
        $X \leftarrow v.paths$ with the i-th one replaced by $\boldsymbol{x}$
        $w \leftarrow CreateNode(c, X, v)$
        $v.children \leftarrow v.children \cup \{w\}$
      end if
    end for
  end if
  if $v.flag =$ "feasible" **then** ▷ Node evaluation
    $X_n \leftarrow v.paths$
  else
    $X_n \leftarrow simulation(\mathcal{S})$
  end if
  $r_n \leftarrow F(X_n)$
  repeat ▷ Backup
    $v.T \leftarrow v.T + 1$
    $v.R \leftarrow r_n / v.T + v.R \cdot (1 - 1/v.T)$
    $v \leftarrow v.parent$
  until $v = root$
end for
return $\underset{1 \le n \le N}{\arg\min}\ F(X_n)$

function CREATENODE(c, X, p)
  return a node with $constraint = c$, $paths = X$, $T = R = 0$, $flag =$ "unknown", $parent = p$, and $children = \emptyset$
end function

The pseudo-code of MCCTS is given in Algorithm 1. It preserves the backbone of CBS for holding constraints and expanding nodes, but does not use its best-first search. Instead, three mechanisms from MCTS, namely the tree policy, the Monte Carlo simulation, and the backup, are introduced. The tree policy recursively applies the proposed LCB strategy to choose a node for expansion, the simulation produces a reward as a value estimation for the state corresponding to the chosen node, and the backup updates the tree based on the reward.

A tree node in MCCTS stores all necessary information for performing the LCB strategy. Consider now a node v. Let $\mathcal{S}$ be the corresponding state and $\mathcal{A}$ be the action set associated with $\mathcal{S}$. The node v contains seven fields listed below:

- $v.parent$: a pointer to the immediate predecessor node of v. This is used for constructing the state or updating the reward incrementally.
- $v.constraint$: a single constraint. This is the action taken to reach the current state. If v is not the root, then we have $\mathcal{S} = \mathcal{S}' \cup \{v.constraint\}$ where $\mathcal{S}'$ is the state corresponding to $v.parent$.
- $v.paths$: a solution produced by the constrained planner $\mathcal{P}$ which satisfies all constraints in $\mathcal{S}$. That is, $v.paths = \mathcal{P}(\mathcal{S})$.
- $v.children$: a set of K pointers to the immediate successor nodes where K is the size of $\mathcal{A}$. Each child corresponds to a distinct action in $\mathcal{A}$.

- $v.R$: a real number recording the average reward of the state $\mathcal{S}$.
- $v.T$: an integer recording the number of visits to the node v.
- $v.flag \in \{$"feasible", "nonterminal", "unknown"$\}$: a text flag. It is initially set to "unknown" when the node is created. The flag will be changed when the associated action set $\mathcal{A}$ is generated. Precisely, it will be changed to "feasible" if the solution stored in $v.paths$ has no conflict and "nonterminal" otherwise.

The function for initializing the nodes is given in Lines 47–49.

The main loop of MCCTS can be divided into four parts: **tree policy** (Lines 6–13), **node expansion** (Lines 14–32), **node evaluation** (Lines 33–38), and **backup** (Lines 39–43). They are elaborated below.

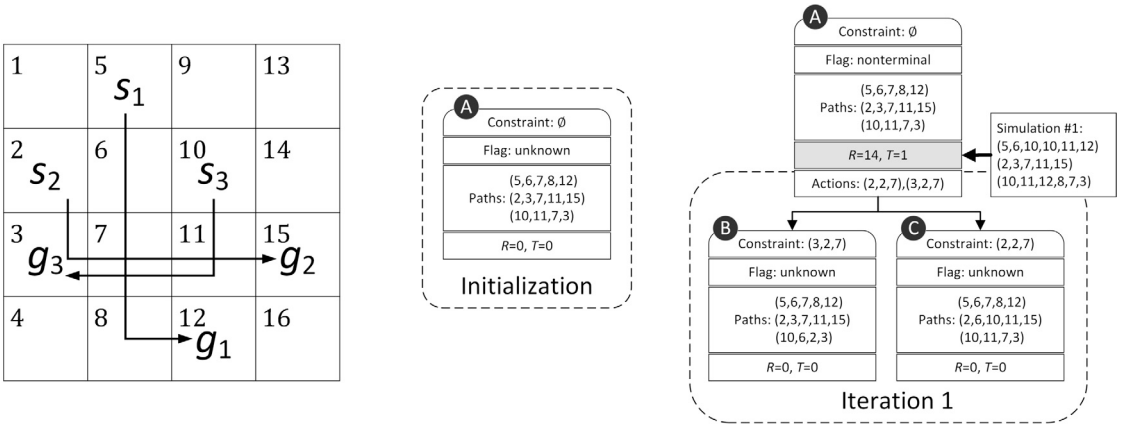
- **Node expansion**. This part creates the action set $\mathcal{A}$ in a way similar to CBS. However, in our implementation, we allow $\mathcal{A}$ to be randomly generated and its size can grow beyond 2. New solutions associated with the child nodes are obtained using the space-time constrained A* algorithm [38]. Note that, for each child node, the space-time constrained A* algorithm is only performed for the agent associated with the newly added constraint; the paths of other agents are extracted from the parent node as no new constraints are added for them (Lines 24–27).
- **Tree policy**. It performs a descent from the root to a leaf. Each step of the descent is essentially a state transition, so we apply the LCB strategy to decide which action to take. Precisely, suppose we are at a node v and find an unvisited child v', i.e., $v'.T = 0$, then this child is selected (Lines 8–9). By the initialization rule of the node, the newly selected node has a flag "unknown", and thus, the tree policy terminates. If, on the contrary, all the children have been visited before, then a child minimizing $v'.R - \alpha\sqrt{\frac{\sqrt{\sum_{u \in v.children} u.T}}{v'.T}}$ is selected. Note here that the quantity $v'.T$ is the number of visits to the child v', $v'.R$ is the average of the revealed rewards, and $\sum_{u \in v.children} u.T$ is the total number of visits to all children of v, so this selection rule implements the LCB strategy given in (14).
- **Node evaluation**. In this part we aim to produce a reward, denoted by r_n, for the selected node v. If the stored solution has no conflict (i.e., $v.flag =$ "feasible"), we compute the reward using the total cost of the paths (Lines 33–34). Then, by definition (6), the reward equals to the state value of v. On the other hand, if the solution has conflicts (i.e., $v.flag =$ "nonterminal"), we call a simulation to get the reward (Line 36). How to implement the simulation will be discussed in Section 6.6. Ideally, this simulation should produce an estimation of $U(\mathcal{S})$. But our analysis shows that its exact value is not critical; the convergence is guaranteed as long as the output of the simulation is bounded. Also note that we will never encounter invalid solutions, as they have been discarded in the node expansion procedure (for failing to pass the test in Line 26).
- **Backup**. It propagates the reward obtained at a leaf to all nodes along the path back to the root, making sure the R-field in each node is equal to the average reward of the corresponding state. This is a standard procedure from the MCTS, designed for accumulating historical information over iterations.

In each iteration, the algorithm first samples a trajectory $(\mathcal{S}_0, \mathcal{A}_0), \ldots, (\mathcal{S}_L, \mathcal{A}_L)$ from the MAPF process, where the state $\mathcal{S}_l$ corresponds to the tree node at depth l and the action set $\mathcal{A}_l$ determines the children of that node. The length of the trajectory, i.e., $L+1$, may vary over iterations. The state transitions are controlled by the LCB strategy and are implemented in **tree policy**. The last action set $\mathcal{A}_L$ is created in **node expansion** which may introduce randomness. Then, a reward is obtained in **node evaluation** as an estimation of $U(\mathcal{S}_L)$. The algorithm is grounded on the following idea: if the reward is close to $U(\mathcal{S}_L)$, then, by Theorem 2, we can reuse it to estimate $V(\mathcal{S}_{L-1}; \mathcal{A}_{L-1})$. In addition, if our estimation of $V(\mathcal{S}_{L-1}; \mathcal{A}_{L-1})$ is sufficiently accurate, and since $\mathbb{E}\left[V(\mathcal{S}_{L-1}; \mathcal{A}_{L-1})\right] = U(\mathcal{S}_{L-1})$, we can further estimate $\mathcal{V}(\mathcal{S}_{L-2}; \mathcal{A}_{L-2})$ by using Theorem 2 again. This suggests that we can apply Theorem 2 recursively to reuse the reward obtained for the last state $\mathcal{S}_L$ in estimating all previous states $\mathcal{S}_{L-1}, \ldots, \mathcal{S}_0$. The **backup** is intended to achieve this goal, by using the reward to update the estimated state values of all predecessor nodes. Such a reward-reuse mechanism comes with no price, since we do not require the rewards in different iterations to be stationary.

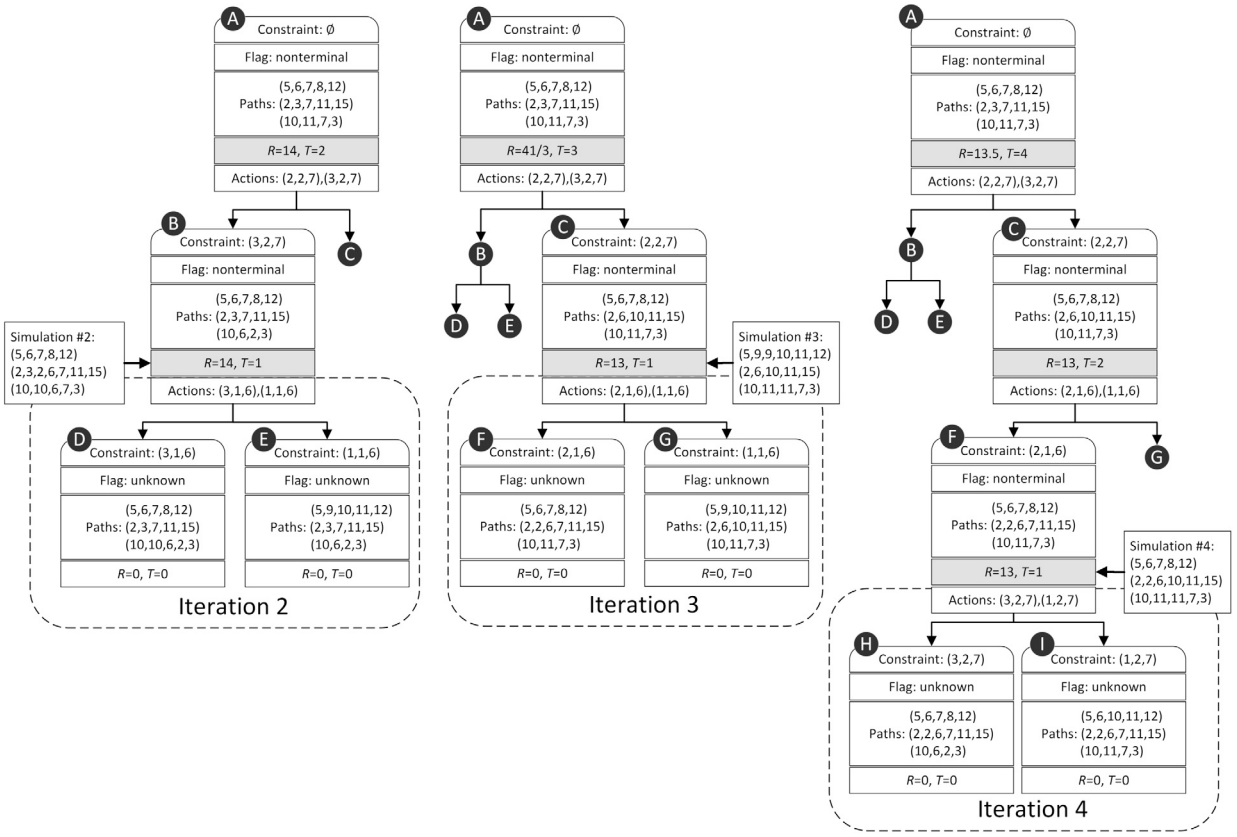
6.4. An example

Fig. 1 illustrates the working procedure of MCCTS on an example map of size 4×4 and with 3 agents. The map is empty, grid-based, and 4-connected. The reward is obtained using random simulation which will be detailed later. The number of constraints used for forming the symmetric action set, K, is fixed to 2. The map and agent configurations are shown in the left of Fig. 1a. The root node (indexed by A and shown in the middle of Fig. 1a) is initialized with the shortest paths planned for each agent individually. It is found that all three agents occupy location 7 at time step 2, so we have three different vertex conflicts to be resolved (i.e., the one between agents 1 and 2, between agents 2 and 3, and between agents 1 and 3).

In the first iteration, the algorithm selects the root node, and changes its flag to "nonterminal" since it involves conflicts. Then, the conflict between agents 2 and 3 is chosen randomly and a constraint set $\{(2,2,7),(3,2,7)\}$ that is symmetric with respect to the root node is generated. Two child nodes, indexed by B and C, are created then, and they record the constraints $(2,2,7)$ and $(3,2,7)$ respectively. Next, agents 2 and 3 are re-planned individually to satisfy these constraints, and the solutions in B and C are updated correspondingly. In the end, a simulation is run from the root node. Suppose for simplicity that it returns a solution with reward 14, and the R and T fields are updated accordingly. The new tree is shown in the right of Fig. 1a.



(a) Left: the grid map. s_i and g_i denote respectively the source and goal locations of agent i. The number in the top left of a grid cell is the index of the corresponding location. Arrows depict the shortest paths planned by A*, which are used for initializing the root node. Middle: the root node after initialization. Right: the tree after the first iteration.



(b) Left, middle, right: trees after the 2nd, 3rd, and 4th iterations, respectively.

Fig. 1. An example on a 4×4 empty grid map with 3 agents. Nodes are indexed by capital letters. Nodes and fields in dashed boxes are created in the corresponding iterations. Fields with gray background store statistics that are updated in the backup procedure.

In iterations 2 and 3, nodes B and C are selected and expanded in order, as they have not been visited before. The trees built after iteration 2 and after iteration 3 are shown in the left and middle of Fig. 1b respectively. Note that node B involves two conflicts, namely the vertex conflict between agents 1 and 3 at time step 1 and the one between agents 1 and 2 at time step 2. So we have two possible action sets that are symmetric with respect to node B: $\{(1,1,6),(3,1,6)\}$ and $\{(1,2,7),(2,2,7)\}$.[2] In this example, the first set is chosen in iteration 2, and two child nodes D and E are created. The simulation performed on node B gives a reward 14, so we have $\{B.R = 14, B.T = 1\}$ and $\{A.R = (14+14)/2 = 14, A.T = 2\}$. The operation on node C in iteration 3 is similar. The reward obtained on node C is 13, and the statistics of nodes C and A are updated to $\{C.R = 14, C.T = 1\}$ and $\{A.R = (14\times 2+13)/3 = 41/3, A.T = 3\}$. It means the average reward recorded at the root node is decreased.

In iteration 4, the **tree policy** performs a descent from the root to its right child C. It is since both B and C have been visited only once but C has a smaller average reward, yielding a lower LCB value. The left child of C, node F, is then chosen to be expanded, as it has not been visited before, which then results to two new children indexed by H and I. The simulation is performed on F and a reward of 13 is obtained. The statistics of F and all its predecessor nodes are then updated to $\{F.R = 13, F.T = 1\}, \{C.R = (13+13)/2 = 13, C.T = 2\}$, and $\{A.R = (41/3\times 3+13)/4 = 13.5, A.T = 4\}$. Therefore, the average reward recored in the root is further reduced. The tree after iteration 4 is shown in the right of Fig. 1b. The algorithm will continue until reaching a pre-defined computation budget.

6.5. Main theoretical results

In the following we show that the value estimation error for the initial state diminishes over iterations and the R-field of the root node converges to the optimal objective value of the MAPF problem.

Theorem 3. *Consider solving the MAPF problem (4) using Algorithm 1. Assume each reward r_n is upper bounded by B and the depth of any tree built in the algorithm is upper bounded by L. Suppose that when performing the LCB strategy at any node of depth l, we set the parameter α to*

$$\alpha_l = \rho N^{\frac{3}{4}-\frac{1.5^l}{2}} \tag{21}$$

for some $\rho > 0$. Then the iterations generated by Algorithm 1 satisfy

$$\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} F(X_n)\right] - F_* = \mathbb{E}[root.R] - F_* \le \frac{2BK}{\sqrt{N}}\left((1+4\rho^2)L + 2\rho^4\left(\frac{96B^2K^2}{\rho^2}+3\right)^L\right), \tag{22}$$

where $root.R$ denotes the average reward stored in the R-field of the root node.

Proof. See Appendix C.

Remark 4 *(The adaptive convergence).* Theorem 3 states that the average reward stored in the root node converges to the optimal objective value of the MAPF problem. As it equals to the averaged objective values over the solutions $\{X_n\}$, we know there exists a convergent subsequence of solutions to the optimum. The convergence here is adaptive, in the sense that it holds for any α satisfying (21). This is the main advantage over the previous work [36], which enjoys a similar theoretical guarantee but requires the problem-related knowledge such as B.

Remark 5 *(The $\mathcal{O}(1/\sqrt{N})$ convergence rate).* For a given problem and a specific algorithmic implementation, B, K, and L can be seen as constants,[3] so the convergence rate of MCCTS is on the order of

$$\mathcal{O}\left(\frac{1}{\sqrt{N}}\right).$$

This result is worse than the one given in [28] (which is $\mathcal{O}(1/N)$), but it does not need the exponential concentration of rewards revealed when performing actions. On the other hand, properly setting the parameter α can further tighten the bound. For example, if B is known, we can choose $\rho = 10BK$ and get a more concrete rate of

[2] As a counterexample, the set $\{(1,1,6),(1,2,7)\}$ is not symmetric with respect to B, as it only restricts agent 1 but there may exist the case where a feasible solution can only be reached by re-planning other agents. The set $\{(1,1,6),(2,2,7)\}$ is not symmetric as well, because it may lead to the situation where the two child nodes are both infeasible.

[3] The upper bound of the tree depth, L, exists once B is fixed. This is because B bounds the reward values from above, and it implies the maximal number of different constraints that are generated in MCCTS is also bounded. Take the vertex constraint as an example. A vertex constraint, by the definition given in Section 5.1, takes the form of (i,t,l), where i is the index of an agent, and t and l denote when and where the constraint should be activated, respectively. Therefore, we have $i \in \{1,\dots,m\}$, $t < B$, and $l \in \mathcal{V}$, which suggests that the number of different vertex constraints is at most $mB|\mathcal{V}|$. Similarly, the number of different edge constraints is at most $mB|\mathcal{E}|$. Since each node in the tree built in MCCTS stores a constraint, and by construction the constraints stored in the path from the root to any leaf are distinct, the tree depth is at most $mB(|\mathcal{V}|+|\mathcal{E}|)$.

$$\mathcal{O}\left(\frac{B^5K^5 4^L}{\sqrt{N}}\right). \tag{23}$$

Here the 4^L term shows the NP-hardness of the MAPF problem. In the case that B is unknown, the convergence is tolerant of overestimation of ρ. For example, one can simply choose any $\rho > 10BK$ and achieve convergence with a rate of

$$\mathcal{O}\left(\frac{\rho^4 BK 4^L}{\sqrt{N}}\right).$$

The importance of Theorem 3 is that it shows how quickly a MAPF problem can be solved:

Corollary 1. *Under the same assumptions in Theorem 3, MCCTS finds an ϵ-suboptimal solution in the first N iterations with probability*

$$\mathbb{P}\left\{F(X_{N,*}) - F_* \le \epsilon\right\} \ge 1 - \frac{2BK}{\epsilon\sqrt{N}}\left((1+4\rho^2)L + 2\rho^4\left(\frac{96B^2K^2}{\rho^2}+3\right)^L\right), \tag{24}$$

where $X_{N,*} \in \arg\min_{1\le n\le N} F(X_n)$ is the best solution found so far.

Proof. See Appendix D.

Remark 6 *(Almost sure optimality).* CBS is known as an optimal MAPF algorithm. Similar to CBS, MCCTS asymptotically converges to the optimal solution, since, for any ϵ, the right-hand side of (24) converges to 1 when $N \to \infty$. MCCTS differs from CBS in that the convergence is not deterministic, but instead, has probability 1. Therefore, MCCTS is almost surely optimal. This means the optimal guarantee of MCCTS is weaker than that of CBS. Nevertheless, these two types of optimality do not differ too much in practice, since we usually only care about the $N < \infty$ case.

6.6. Implementation of simulation

The proposed method differs from existing CBS-like algorithms in that we do not try to evaluate conflicting solutions. When a state S is found to be nonterminal, i.e., the associated solution contains conflicts, we use a simulation to get a reward to approximate $U(S)$. The simulation is a sub-procedure which produces a feasible solution based on the constraints in the current state. According to the above analysis, the simulation does not need to be very accurate; the only condition to guarantee convergence is that the objective value of this solution should be bounded (which usually holds in practice). In the following, we discuss two possible ways to construct a simulation.

6.6.1. Option I: using existing MAPF algorithms

Any MAPF algorithm can be used as a simulation as long as it can accept as input a set of constraints and then output a feasible solution satisfying the accepted constraints. Here we propose that a practical choice is to use the ECBS family of suboptimal algorithms with a large suboptimal bound parameter. Technically, using ECBS or its variants may weaken our theoretical results, as these algorithms do not guarantee to terminate after solving a polynomial number of single-agent subproblems. But in practice, we can set the suboptimal bound parameter ϵ to a large value, say $\epsilon = 2$, such that these algorithms can quickly return a feasible solution. On the other hand, this shows that the proposed MCCTS can serve as a MAPF framework in which existing state-of-the-art algorithms can be incorporated as building blocks.

6.6.2. Option II: random simulation

Here we provide another implementation called random simulation. Its pseudo-code is given in Algorithm 2. This method starts with a conflicting solution X and updates it iteratively. The initial solution is extracted from the node where the simulation is performed. At each iteration, a symmetric constraint set consisting of two constraints c_1 and c_2 is generated randomly (Line 3). We attempt to resolve the conflict in X with c_1 and c_2 individually. This can be done via re-planning the corresponding agent with an A* algorithm. The constraint yielding a solution with fewer conflicts is then chosen to update X and merged into the constraint set $\mathcal{S}$. Since the accuracy of the solution is not a necessary condition for convergence, we use a bounded suboptimal variant of A* to accelerate the simulation. Note that, as no backtracking mechanism is involved, there is no guarantee that a feasible solution can always be found.[4] However, once the random simulation produces a feasible solution, the number of iterations performed can be shown to be polynomial in the objective value of this solution. We formulate this in the following proposition.

Proposition 1. *When Algorithm 2 returns a feasible solution with the objective value upper bounded by B, the number of calls to the (bounded suboptimal) A* algorithm is upper bounded by $2mB(|\mathcal{V}| + |\mathcal{E}|)$.*

[4] Practically, the random simulation produces a feasible solution in many cases. In our experiments presented in Section 7, it always does so.

Algorithm 2 Random simulation.

Require: A set of constraints S
1: $X \leftarrow$ the solution associated with the node on which the simulation is performed
2: **while** X has conflicts **do**
3: $\{c_1, c_2\} \leftarrow$ a symmetric constraint set of size 2, generated randomly for resolving a conflict in X
4: **for** $j \in \{1,2\}$ **do**
5: $i_j \leftarrow$ index of the agent involved in c_j
6: $\boldsymbol{x}_j \leftarrow$ a path planned for agent i_j with (bounded suboptimal) A* subject to constraints in $S \cup \{c_j\}$
7: **if** $\boldsymbol{x}_j$ exists **then**
8: $y_j \leftarrow$ the number of conflicts in X if replacing its i_j-th path with $\boldsymbol{x}_j$
9: **else**
10: $y_j \leftarrow \infty$
11: **end if**
12: **end for**
13: $j' \leftarrow \arg\min_{j \in \{1,2\}} y_j$
14: $S \leftarrow S \cup \{c_{j'}\}$
15: Replace the $i_{j'}$-th path in X with $\boldsymbol{x}_{j'}$
16: **end while**
17: **return** X

Proof. See Appendix F.

The random simulation has an important theoretical implication: when coupling random simulation with MCCTS, the non-asymptotic convergence can be completely measured in terms of the number of calls to SAPF solvers. This is because in each iteration of MCCTS, we only need one call to an optimal SAPF solver (Line 25 of Algorithm 1) and a polynomial number of calls to a suboptimal SAPF solver (Line 6 of Algorithm 2). We formulate this observation in the following corollary. To our knowledge, this is the first time that the hardness of solving a MAPF problem can be quantified using the number of SAPF subproblems required to be solved.

Corollary 2. *Consider solving the MAPF problem (4) using Algorithm 1 with the random simulation in Algorithm 2. Let N be the number of iterations performed in Algorithm 1 and $\tilde{N}$ be the total number of SAPF subproblems solved. Then, with the same assumptions as in Theorem 3, we have*

$$\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} F(X_n)\right] - F_* \le 2BK\sqrt{\frac{1+2mB(|\mathcal{V}|+|\mathcal{E}|)}{\tilde{N}}}\left((1+4\rho^2)L + 2\rho^4\left(\frac{96B^2K^2}{\rho^2}+3\right)^L\right).$$

Proof. See Appendix G.

7. Numerical studies

To verify the theoretical results, we use the proposed MCCTS method to solve a set of benchmark problems as well as several real-world problems.

7.1. Experimental settings

7.1.1. Test problems

We consider two sets of problems, all of which are defined over 4-neighbor grids. The first one is a benchmark set[5] constructed for a 32×32 grid map with 204 uniformly distributed obstacles. We vary the number of agents m in $\{20, \ldots, 100\}$. For each setting of m, 50 different problems are generated via randomly placing the sources/goals of the agents and the obstacles in the map. In some problems, the source location and the goal location may be the same for a single agent; we remove those problems and obtain a total of 420 valid test problems.

The second test set includes three widely used maps from video games, namely "den520d", "brc202d", and "lak303d", collected in the Moving AI benchmark suites [7]. We consider two settings for the number of agents: $m = 50$ and $m = 100$. For each setting, we choose 25 different scenes,[6] where different scenes have different sources/goals for the agents. In this way, we have a total of $3 \times 2 \times 25 = 150$ different test problems in this test set.

7.1.2. Algorithms for comparison

Two MCCTS implementations and three existing MAPF algorithms are chosen for comparison:

[5] Publicly available at github.com/whoenig/libMultiRobotPlanning.
[6] Publicly available at movingai.com.

- **CBS**. This is the optimal version of the CBS algorithm [9].
- **ECBS**. This is the suboptimal variant of CBS proposed in [27]. It receives the suboptimal bound parameter ϵ and produces a solution with cost no larger than ϵF_*. In the experiment, we choose ϵ from $\{1.05, 1.1, 1.2\}$. An ECBS instantiation with parameter value ϵ is denoted as ECBS-ϵ.
- **AFS**. This is an anytime variant of ECBS proposed in [29]. It iteratively performs ECBS-ϵ with a decreasing ϵ. Therefore, it is similar to MCCTS with ECBS as the simulation in that every iteration can generate a bounded suboptimal solution. To save computation, the constraint tree in ECBS is retained over iterations rather than rebuilt from scratch. In the experiment, we choose $\epsilon = 2$ for the initial iteration and then decrease it gradually in a way that every iteration can monotonically decrease the cost of its produced solution.
- **MCCTS-ECBS**. This is the MCCTS implementation using ECBS as the simulation. The ECBS involved has been modified such that it can receive a set of initial constraints. We fix $\epsilon = 2$ in the experiment. We note that this ϵ value is generally too large for using ECBS as a standalone MAPF solver, as in most cases the produced results would have a quite long path. The intention of this choice is threefold: 1) showing the potential of using existing state-of-the-art MAPF solvers as a building block in the proposed method; 2) trying to accelerate the simulation, as a large ϵ usually leads to short computation time; 3) demonstrating that a less accurate simulation can lead to highly accurate final solutions.
- **MCCTS-RANDOM**. This is the MCCTS algorithm where random simulation (Algorithm 2) is used. We use the ϵA* algorithm proposed in [29] for re-planning the agents (Line 6 of Algorithm 2), fixing $\epsilon = 2$ in the experiment.

For CBS, ECBS, AFS, and our simulation procedures that rely on CBS or ECBS, the following improvements are applied: 1) conflict bypassing [16], 2) constraint prioritization [16], and 3) high-level heuristics [17].

7.1.3. Generic settings

For MCCTS, our theoretical analysis suggests that the bandit parameter α should shrink with the depth of the tree nodes. However, since the optimal setting for this parameter is problem-dependent, we simply set it to $\alpha = 0.9^i$ where i is the depth of the node on which the bandit strategy is executed. For example, the value of α is 1, 0.9, and 0.81 at the first, the second, and the third levels of the tree, respectively. We find this setting is simple and is effective on a wide range of test problems. For each node expansion in MCCTS, we consider generating a single pair of constraints to construct the action set, i.e., $K = 2$, which coincides with the CBS algorithm. However, we will also investigate other settings.

All algorithms have a computation time budget of 3 minutes. Since MCCTS is a randomized algorithm, we run MCCTS independently for 5 times on each test problem.

7.1.4. Performance metrics

We consider two visualization methods for evaluating the performance of the algorithms. The first is the usual trajectory plot showing the convergence behavior of a certain algorithm on an individual test problem. It is used for verifying the theoretical results, via showing the objective values versus the computation time or iterations.

The second is the performance profile plot [39], a widely used tool in benchmarking optimization softwares. It compares multiple algorithms on multiple test problems simultaneously, providing a way of reflecting the overall performance of algorithms under consideration. We briefly introduce the performance profile plot below.

Suppose we have a set of algorithms $\mathfrak{A} = \{a_i\}$ and a set of problems $\mathfrak{P} = \{p_j\}$. Let $c_{i,j}$ denote the best objective value found by algorithm a_i on problem p_j. Then, for each problem p_j, its numerical optimal solution is defined as

$$c_j^* = \min_{i:a_i \in \mathfrak{A}} \{c_{i,j}\}.$$

Let $\tau > 1$ be a small number greater than 1. Define $t_{i,j}(\tau)$ as the shortest computation time required for algorithm a_i on problem p_j to find a solution with the objective value lower than τc_j^*. Define the relative performance ratio of a_i on p_j as

$$r_{i,j}(\tau) = \frac{t_{i,j}(\tau)}{\min_{i:a_i \in \mathfrak{A}} \{t_{i,j}(\tau)\}}. \tag{25}$$

The performance profile of algorithm a_i is a cumulative distribution function of the relative performance ratio over all problems:

$$\rho_i(\eta; \tau) = \frac{1}{|\mathfrak{P}|} \sum_{j:p_j \in \mathfrak{P}} \mathbb{I}\{r_{i,j}(\tau) \le \eta\}. \tag{26}$$

It is then clear that τ defines the target suboptimality. So the slope of a profile measures how fast, in expectation, an algorithm can solve the problems with suboptimality bounded by τ. The area under the curve of the profile is an overall performance indicator of the algorithm; the larger the better. For example, if the profile curve of an algorithm a lies to the left of another algorithm a', then we can conclude that a has better performance than a'.

Table 1
Models fitted to the curves of MCCTS-RANDOM.

m	Fitted model: $y(N) = a/\sqrt{N} + b$	Fitting error: $\lvert b - F_* \rvert / F_*$
20	$y(N) = 28.527604/\sqrt{N} + 458.021989$	$= 0.002236$
30	$y(N) = 45.743587/\sqrt{N} + 691.822917$	$= 0.000256$
40	$y(N) = 39.953093/\sqrt{N} + 938.831336$	$= 0.006250$
50	$y(N) = 156.530393/\sqrt{N} + 1167.435158$	≤ 0.013991
60	$y(N) = 80.226322/\sqrt{N} + 1479.893186$	≤ 0.010767
70	$y(N) = 124.264197/\sqrt{N} + 1724.192285$	≤ 0.001273
80	$y(N) = 87.249687/\sqrt{N} + 1976.989896$	≤ 0.013321
90	$y(N) = 142.306807/\sqrt{N} + 2319.718204$	≤ 0.021003
100	$y(N) = 87.277631/\sqrt{N} + 2565.267591$	≤ 0.024059

7.2. Verification of theoretical analysis

First, we verify the $\mathcal{O}(1/\sqrt{N})$ convergence rate derived in Theorem 3. We perform MCCTS-RANDOM and MCCTS-ECBS on the 32×32 grid maps with different numbers of agents, m. Its convergence behavior is shown in Fig. 2. For each setting of m, the curve of an algorithm is plotted using data from its median run.[7] We have the following observations:

- The sublinear convergence of the average reward is obvious especially in the cases of $m \leq 50$. The algorithms are found to achieve a rapid improvement in the initial phase, and then followed by a significant slowdown after several hundreds of iterations. This behavior is common for stochastic optimization algorithms and is consistent with the obtained $\mathcal{O}(1/\sqrt{N})$ rate.
- MCCTS-ECBS suffers from divergence in certain cases. This is probably because the ECBS based simulation generally takes longer computation time and hence the total number of iterations performed is quite limited (since the computation time budget is fixed). These plots therefore do not capture the long-term trend of the convergence. However, we can still observe a decreasing trend in the best reward.
- In certain cases for $m \geq 50$, MCCTS-ECBS converges faster than MCCTS-RANDOM in terms of the R-field of the root node, but the latter is more likely to identify better solutions. This is probably due to that the random simulation is purely randomized and thus has better exploration ability than the ECBS based simulation.

On the other hand, we note that the derived rate is just an upper bound on the convergence; the real convergence behavior could be different. It is therefore interesting to verify whether the convergence rate is tight. The tightness of the rate can be revealed via firstly fitting the curves obtained in Fig. 2 with a model on the order of $\mathcal{O}(1/\sqrt{N})$ and then observing the fitting error. Precisely, consider the following sublinear model

$$y(N) = \frac{a}{\sqrt{N}} + b \tag{27}$$

to be fitted with the curves in Fig. 2. Then according to Theorem 3, if the convergence rate is tight, the parameter b will be close to the optimal objective value F_*; therefore, b can be considered as a prediction of F_*. Here we consider fitting the curves of MCCTS-RANDOM with least squares regression. The results are given in Table 1.

The second column of Table 1 provides the fitted model obtained for different settings of m. In general, the model parameter b increases quickly with the increase of m; since this parameter is intended to capture the optimal objective value, this observation is consistent with the NP-hard nature of the MAPF problem.

The last column gives the relative fitting error measured by $\lvert b - F_* \rvert / F_*$. For the case of $m \in \{20, 30, 40\}$, F_* can be obtained from CBS, so we calculate the relative fitting error explicitly. In other cases, CBS fails to terminate within reasonable computation time, so we use ECBS instead. This admits computing an upper bound of the relative fitting error, which is sufficient to show the tightness of the convergence rate. Specifically, let $\tilde{F}_*$ denote the obtained suboptimal objective value from ECBS and ϵ be the suboptimal bound parameter. If the parameter b is greater than F_*, we can bound the relative fitting error as

$$\frac{\lvert b - F_* \rvert}{F_*} = \frac{b - F_*}{F_*} = \frac{b}{F_*} - 1 \leq \frac{\epsilon b}{\tilde{F}_*} - 1, \tag{28}$$

where the inequality is due to the fact $\tilde{F}_* \leq \epsilon F_*$. In order to meet the condition $b \geq F_*$, when fitting the model (27), we impose a boundary constraint $b \geq F(X_{N,*})$ where $X_{N,*}$ denotes the best solution found in the first N iterations. In this way, we can ensure

[7] The median run is defined as the run achieving the median value among all independent runs and all randomly generated test problems. To be more concrete, suppose for a given m there are 10 randomly generated test problems, then we will have 50 runs for MCCTS, as on each problem it is run independently 5 times. Each run results in a sequence of solutions and the best of them is identified. With the 50 obtained best solutions, compute the median of the corresponding objective values. Then find out from the 50 runs which run leads to the best objective value closest to the median. This run is what we called a "median run".

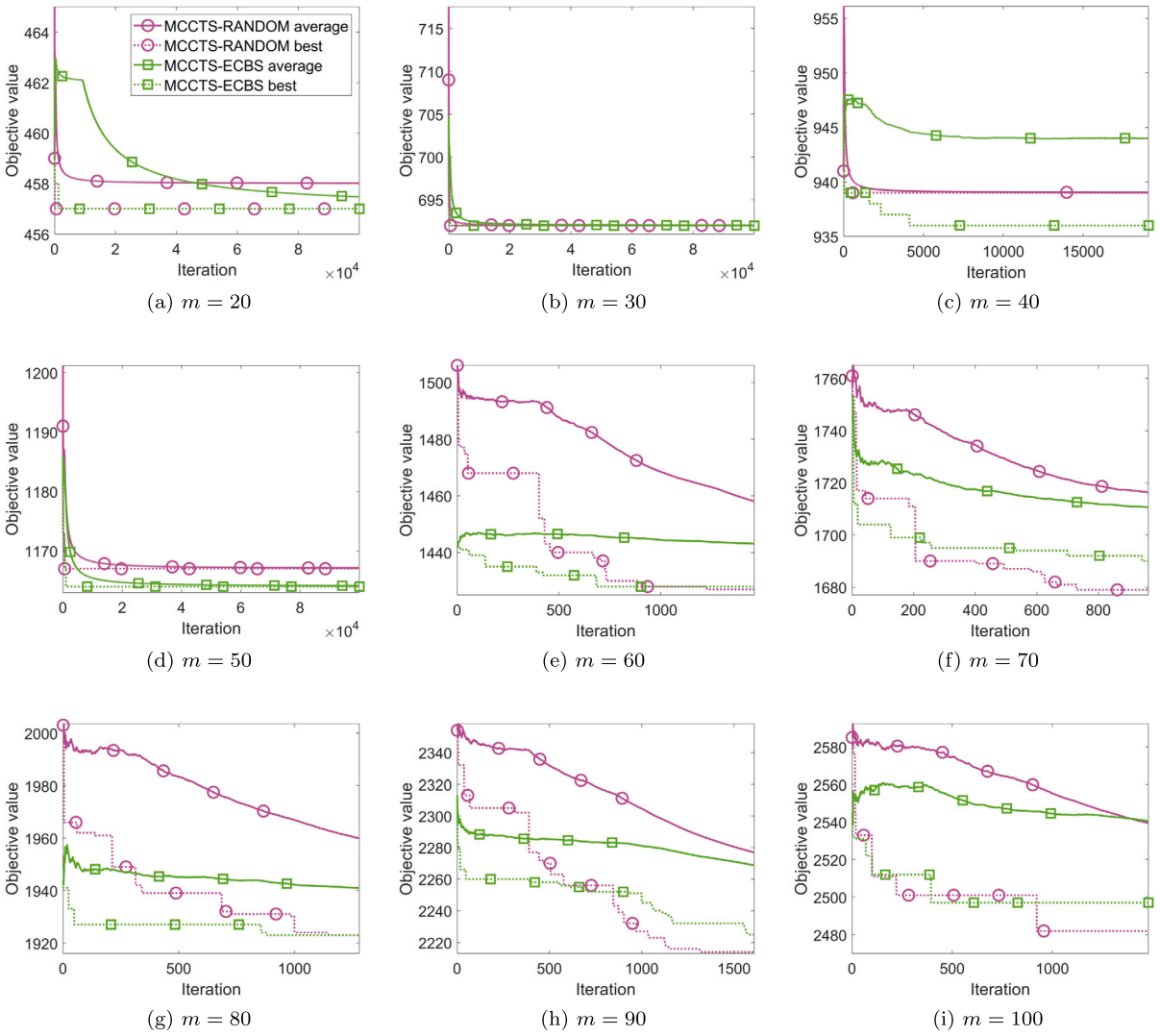


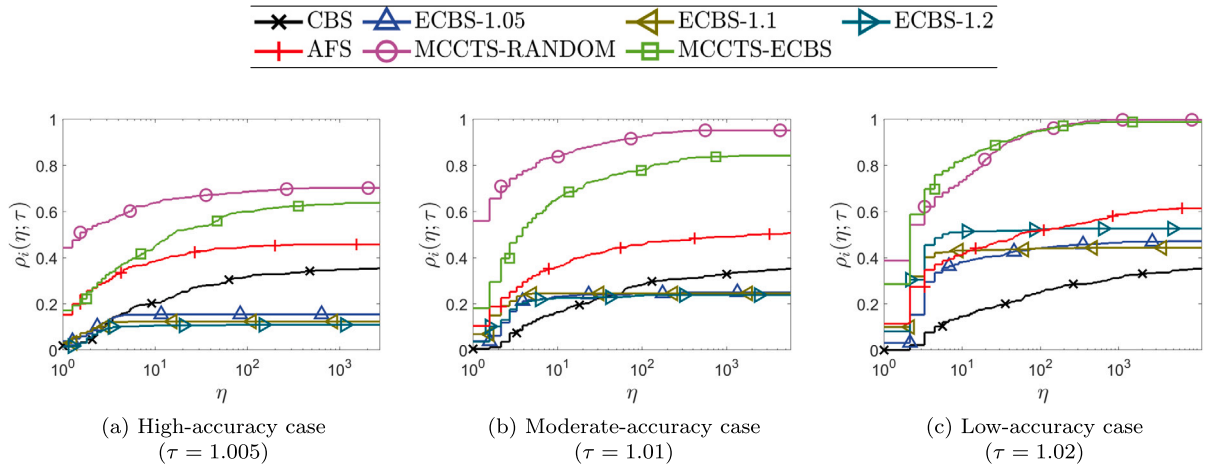
Fig. 2. Convergence behaviors of MCCTS-RANDOM and MCCTS-ECBS on the 32×32 random grid maps. The "average" curve depicts the average over all simulation results, which is stored in the R-field of the root node. The "best" curve reflects the objective value of the best feasible solution found so far.

$b \geq F(X_{N,*}) \geq F_*$ and therefore the bound given in (28) is valid. We choose $\epsilon = 1.05$ for $m \in \{50, 60, 70\}$, $\epsilon = 1.1$ for $m \in \{80, 90\}$, and $\epsilon = 1.2$ for $m = 100$. It is then found that in all cases the relative fitting error is small, being less than 0.05 in all cases. This suggests that the obtained $\mathcal{O}(1/\sqrt{N})$ rate has achieved a relatively high accuracy and does reflect the real algorithmic behavior.

7.3. Comparison with existing algorithms

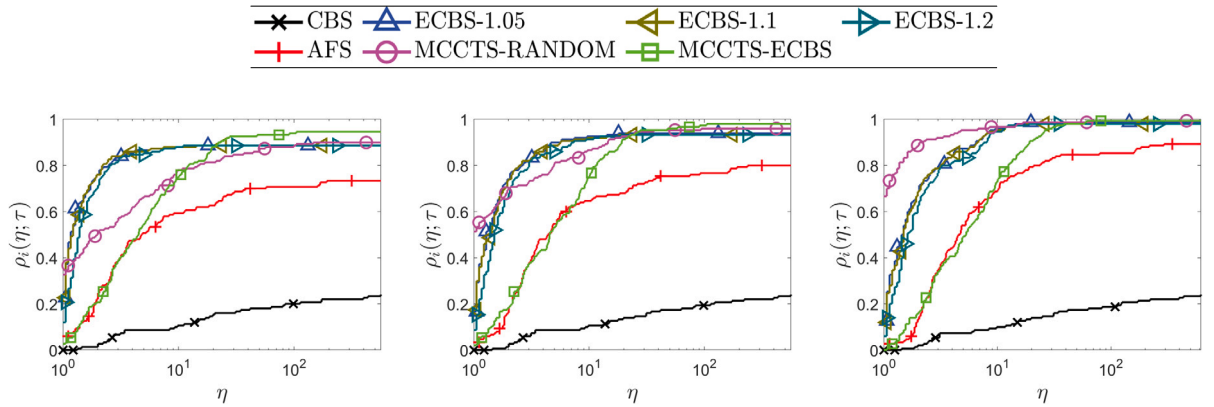
Next, we investigate the relative performance of the two MCCTS implementations compared to existing algorithms. To capture the overall performance on a wide range of different test problems, we use the profile plots introduced earlier. The accuracy parameter τ is set to 1.005, 1.01, and 1.02, which correspond respectively to the high-, moderate-, and low-accuracy cases. The results for the 32×32 random maps are shown in Fig. 3 and those for the video games are shown in Fig. 4. The following observations can be obtained:

- Compared to ECBS, the MCCTS methods offer a much smoother and more flexible trade-off between tractability and accuracy. ECBS does not show a significant difference in the overall performance when using different suboptimal bound parameters. In fact, ECBS can be even worse than the standard CBS method in certain cases. The two MCCTS implementations do not suffer from this issue and they are superior to CBS in all considered cases.
- MCCTS-RANDOM and MCCTS-ECBS behave similarly to AFS in that they all exhibit the anytime characteristic: the longer computation time is spent, the more problems these algorithms can solve with the required accuracy. The two MCCTS implementations have clearly better overall performance, as their profile curves in most cases lie to the left of AFS.



(a) High-accuracy case ($\tau = 1.005$) (b) Moderate-accuracy case ($\tau = 1.01$) (c) Low-accuracy case ($\tau = 1.02$)

Fig. 3. Performance profiles on 32×32 random grid maps. The x-axis denotes the relative performance ratio defined in (25), which is a unified and normalized measure for the computation time budget. The y-axis denotes the percentage of test problems solved with the required accuracy, as defined in (26). They have the same meaning in other profile plots in the rest of this paper.



(a) High-accuracy case ($\tau = 1.005$) (b) Moderate-accuracy case ($\tau = 1.01$) (c) Low-accuracy case ($\tau = 1.02$)

Fig. 4. Performance profiles on the three real maps from video games.

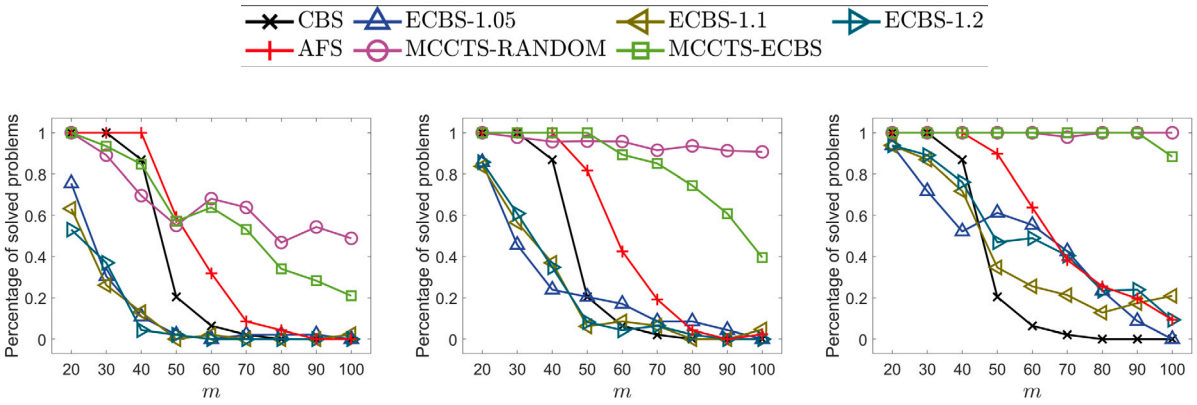
- On the 32×32 random maps, MCCTS-RANDOM and MCCTS-ECBS are the best and the second best performers, having a clear advantage over all other competitors in all cases. This can be seen from that their profile curves lie to the left of all other algorithms.
- The MCCTS methods demonstrate competitive performance on the real maps. Once a large computation time budget is allowed, they can solve the most test problems. MCCTS-RANDOM generally surpasses MCCTS-ECBS when the time budget is limited. This suggests that the effectiveness of the simulation methods is problem-dependent.

Fig. 5 compares the algorithms' long-term performance via showing the percentage of problems that are finally solved within the time budget. The results are obtained on the 32×32 grid maps. MCCTS-RANDOM performs the best as its curves in most cases lie above the others. MCCTS-ECBS is worse than MCCTS-RANDOM, and is more likely to be affected by the increasing number of agents.

Fig. 6 compares the anytime performance between MCCTS and AFS. The results are obtained on the 32×32 grid maps and presented individually for each setting of m. In general, AFS is effective if the number of agents is small (e.g., $m \leq 30$) while MCCTS has better performance when dealing with a large number of agents. MCCTS-ECBS tends to take longer time than MCCTS-RANDOM in producing the first feasible solution, but MCCTS-ECBS in the majority of the cases can produce better solutions than MCCTS-RANDOM when the computation time budget is limited, e.g., within one second.

7.4. Effectiveness of the LCB strategy

We have shown that, in non-stationary settings, MCCTS enjoys better theoretical guarantees than the standard MCTS. Here we verify that MCCTS is also competitive with the latter in practice. To this end, we implement a variant of MCCTS-RANDOM by replacing the LCB strategy with UCB1, the standard bandit strategy of UCT. It is done by changing Line 11 of Algorithm 1 to



(a) High-accuracy case ($\tau = 1.005$) (b) Moderate-accuracy case ($\tau = 1.01$) (c) Low-accuracy case ($\tau = 1.02$)

Fig. 5. Percentage of solved problems versus the number of agents, measured on 32×32 random grid maps and within the computation budget of 3 minutes. The criterion for checking whether a problem can be successfully solved by a given algorithm is the same as that in defining the profile plots.

$$v \leftarrow \underset{v' \in v.children}{\arg\min} \; v'.R - \alpha \sqrt{\frac{\log\left(\sum_{u \in v.children} u.T\right)}{v'.T}}$$

while keeping all other parts and parameter settings unchanged. So MCCTS-RANDOM with UCB1 can be viewed as an adaptation of UCT to the MAPF problem.

The MCCTS-RANDOM with the two different strategies are compared on the 32×32 random grid maps. Experimental settings are the same as those used in Section 7.3. Fig. 7 visualizes the results in profile plots. The proposed LCB strategy is slightly worse than UCB1 in the high-accuracy case, but is better in the moderate- and low-accuracy cases. But in general, the difference in performance is insignificant. This suggests that the proposed LCB strategy is able to improve the theoretical performance of MCTS, and is supported by empirical results that are at least comparable to the classical UCB1 strategy.

7.5. Influence of number of paired constraints K

According to the analysis, the convergence rate in its optimal case (i.e., in (23)) involves only one hyperparameter: K, the number of constraints in each action set. Therefore we are interested in its influence on the algorithm performance. The MCCTS methods in the above experiments use the setting $K = 2$. That is, only a single pair of constraints is generated when expanding a tree node, being consistent with existing CBS-like algorithms. In the following we investigate the case for $K > 2$.

We compare two settings of K, namely $K = 2$ and $K = 4$, when used in MCCTS-RANDOM on the 32×32 random grid maps. Note that $K = 4$ means generating two pairs of constraints per node expansion.[8] The results are visualized using profile plots in Fig. 8. The impact of K generally depends on the setting: a large K is preferred in the high-accuracy case, whereas in the low-accuracy case a small K leads to better short-term performance. So this does not coincide with the optimal rate derived in (23), which suggests using a small K in all situations. It implies the K-dependence of the obtained convergence rate is improvable.

8. Concluding remark

In this paper we bring theoretical improvements to existing MAPF algorithms. Our method is to convert a MAPF problem into a stochastic process where the state is a set of constraints and the state transition is performed by including a new constraint for addressing an unresolved conflict. We show that the original MAPF problem can be solved via evaluating the initial state value of the corresponding stochastic process. The key step is to find the optimal state transition, which is equivalent to selecting the best constraint in traditional conflict tree based MAPF algorithms. We propose a novel bandit strategy to achieve this goal, aiming to bound the number of visits to non-optimal actions. Based on these techniques, the MCCTS method is proposed and two possible methods for implementing the simulation are discussed. We show that the MCCTS method achieves the $\mathcal{O}(1/\sqrt{N})$ convergence rate, which to our knowledge is the first non-asymptotic performance guarantee for MAPF algorithms. We carry out numerical simulations on several grid maps and show that MCCTS is superior to or at least competitive with existing algorithms such as CBS, ECBS, and AFS.

[8] In the case when only one conflict exists, we generate a single pair of constraints.

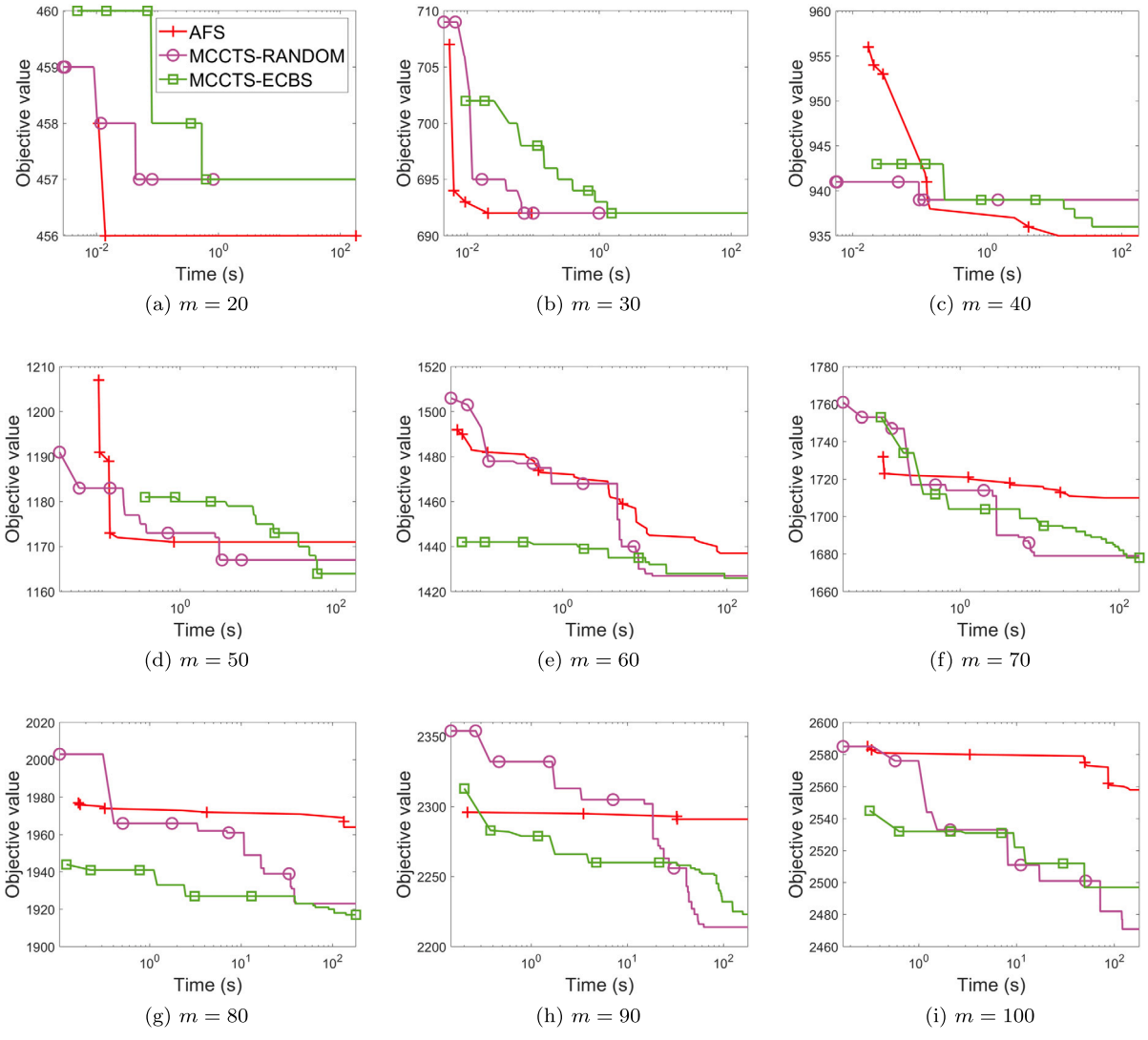


Fig. 6. Anytime performance comparison of MCCTS-RANDOM, MCCTS-ECBS, and AFS on the 32×32 random grid maps. The curves show the best objective value found versus the computation time. The experimental data are from the run that achieves the median final result among all independent runs and all randomly generated test problems.

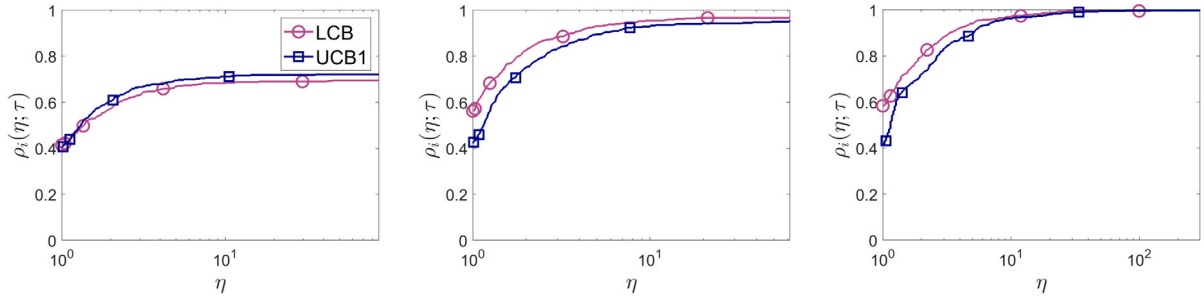
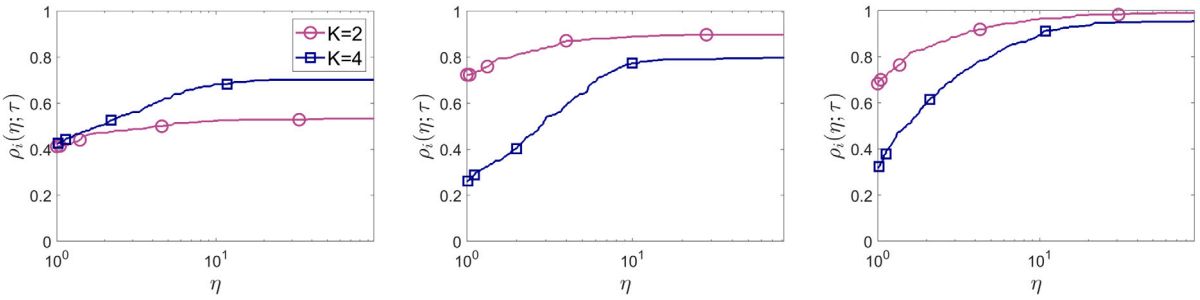


Fig. 7. Performance profiles of MCCTS-RANDOM with LCB and with UCB1, obtained on 32×32 random grid maps.



(a) High-accuracy case ($\tau = 1.005$) (b) Moderate-accuracy case ($\tau = 1.01$) (c) Low-accuracy case ($\tau = 1.02$)

Fig. 8. Performance profiles of MCCTS-RANDOM with different settings of K, obtained on 32×32 random grid maps.

CRediT authorship contribution statement

Xiaoyu He: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Xueyan Tang:** Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Writing – review & editing. **Wentong Cai:** Funding acquisition, Project administration, Supervision. **Jingning Li:** Project administration, Resources, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This study is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from Singapore Telecommunications Limited (Singtel), through Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU). Xiaoyu He is also partially supported by the National Natural Science Foundation of China (62006252). Xueyan Tang is also partially supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award MOE-T2EP20122-0007).

Appendix A. Proof of Theorem 1

Let X be a feasible MAPF solution. We say a state S permits a solution X if X is consistent with all constraints in S. It is clear that S_0 permits X.

Now assume a nonterminal state S permits X. Let $\mathcal{A}$ be a symmetric random action set with respect to $\mathcal{P}(S)$. If X is not consistent with a constraint $c \in \mathcal{A}$, then by the feasibility of X and the symmetric property of the action set, we have another constraint $c' \in \mathcal{A}$ such that X must be consistent with c'. This leads to an induction rule such that S always has an immediate successor state permitting X. We then conclude that there exists a terminal state permitting X.

Let X_* be an optimal solution and S_* be a terminal state permitting X_*. By (6) and (7), we have

$$V(S_*;\emptyset) = U(S_*) = F(X_*) = F_*. \tag{A.1}$$

Let S' be any immediate predecessor state of S_* and $\mathcal{A}'$ the corresponding random action set. We then have

$$V(S';\mathcal{A}') \overset{(7)}{=} \min_{c\in\mathcal{A}'} U(S' \cup \{c\}) = U(S_*) \overset{(A.1)}{=} F_*, \tag{A.2}$$

where the second equality is because S' is the immediate predecessor state of S_*.

Now we prove $U(S') = F_*$ via showing $U(S') > F_*$ is false.

First, we assume $U(S') > F_*$ holds. With this assumption, we know that there exists another random instance of the action set, denoted as $\tilde{\mathcal{A}}'$, such that $V(S';\tilde{\mathcal{A}}') > F_*$; otherwise, (8) would not hold. We then obtain for any immediate successor state $S'' = S' \cup \{\tilde{c}\}$ for $\tilde{c} \in \tilde{\mathcal{A}}'$ such that

$$U(S'') \overset{(7)}{\geq} V(S';\tilde{\mathcal{A}}') > F_*.$$

It then forms an induction rule stating that, conditioned on the random set $\tilde{\mathcal{A}}'$, all successor states of S' have a U value greater than F_*. On the other hand, since S' permits X_*, we know that there exists a terminal state, conditioned on $\tilde{\mathcal{A}}'$, permitting X_* and hence having a value equal to F_*. This yields a contradiction; so $U(S') > F_*$ cannot hold. It then follows from (A.2) that

$$U(S') = V(S'; \mathcal{A}') = F_* \quad \forall \mathcal{A}'. \tag{A.3}$$

Combining (A.1) and (A.3) we conclude that all predecessor states of S_* have U and V values equal to F_*, independent of the concrete instances of the random action set $\mathcal{A}$. Since S_0 is a predecessor state of S_*, we then reach (10).

Appendix B. Proof of Theorem 2

Let k_* be the index of the unique optimal action, i.e.,

$$k_* = \arg\min_{1 \le k \le K} U(S \cup \{c_k\}; \mathcal{A}).$$

To simplify notations, we will directly use $*$ as a subscript to denote the index k_*. For example, c_* will indicate the optimal action c_{k_*}. Define $\mu_k = U(S \cup \{c_k\})$, $\mu_* = U(S \cup \{c_*\})$. Further denote $\mu_{N,k} = \mathbb{E}\left[\bar{R}_{N,k}\right]$ as the expectation of the empirical mean reward of selecting c_k.

We first introduce a key lemma which bounds the first- and second-order moments of the number of times that a suboptimal action is selected. Its proof is given in Appendix E.

Lemma 1. *Let k be the index of a suboptimal action, i.e., $\Delta_k > 0$. Under the assumptions in Theorem 2, the number of times this action being selected in N' rounds is bounded by*

$$\mathbb{E}\left[Y_{N',k}\right] \le 1 + 4\left(\frac{\sigma^2}{\alpha^2} z^\gamma + \frac{\alpha^2}{\Delta_k^2}\right)\sqrt{N'}, \tag{B.1}$$

$$\mathbb{E}\left[Y_{N',k}^2\right] \le 16\left(\frac{\sigma^2}{\alpha^2}(N')^{2.5} z^\gamma + \frac{\alpha^4}{\Delta_k^4} N'\right). \tag{B.2}$$

Proof of (18). We begin with rewriting $\bar{R}_{N'} - \mu_*$ as

$$\begin{aligned}
\bar{R}_{N'} - \mu_* &\overset{(13)}{=} \frac{1}{N'} \sum_{k=1}^{K} Y_{N',k} \bar{R}_{Y_{N',k},k} - \mu_* \\
&= \frac{1}{N'} \sum_{k=1}^{K} Y_{N',k} \bar{R}_{Y_{N',k},k} - \mathbb{E}[\bar{R}_{N',*}] + \mu_{N',*} - \mu_* \\
&= \frac{1}{N'} \sum_{k=1}^{K} Y_{N',k} \bar{R}_{Y_{N',k},k} - \mathbb{E}\left[\frac{1}{N'} \sum_{n=1}^{N'} R_{n,*}\right] + \mu_{N',*} - \mu_*.
\end{aligned}$$

Taking expectation at both sides gives

$$\left|\mathbb{E}\left[\bar{R}_{N'} - \mu_*\right]\right| \le \frac{1}{N'} \underbrace{\left| \sum_{k=1}^{K} \mathbb{E}\left[Y_{N',k} \bar{R}_{Y_{N',k},k}\right] - \mathbb{E}\left[\sum_{n=1}^{N'} R_{n,*}\right] \right|}_{\overset{\text{def}}{=} \mathfrak{A}} + \left|\mu_{N',*} - \mu_*\right| \le \frac{1}{N'} |\mathfrak{A}| + \frac{\beta}{\sqrt{N'}}, \tag{B.3}$$

where the second inequality is due to (15).

Now we rewrite the term $\mathfrak{A}$ in (B.3) as

$$\begin{aligned}
\mathfrak{A} &= \sum_{k \ne *} \mathbb{E}\left[Y_{N',k} \bar{R}_{Y_{N',k},k}\right] + \mathbb{E}\left[Y_{N',*} \bar{R}_{Y_{N',*},*}\right] - \sum_{n=1}^{N'} \mathbb{E}\left[R_{n,*}\right] \\
&= \sum_{k \ne *} \mathbb{E}\left[Y_{N',k} \bar{R}_{Y_{N',k},k}\right] + \mathbb{E}\left[\sum_{n=1}^{Y_{N',*}} R_{n,*}\right] - \sum_{n=1}^{N'} \mathbb{E}\left[R_{n,*}\right]
\end{aligned}$$

$$= \underbrace{\sum_{k\neq *} \mathbb{E}\left[Y_{N',k}\bar{R}_{Y_{N',k},k}\right]}_{\stackrel{\text{def}}{=}\mathfrak{B}_1} - \underbrace{\mathbb{E}\left[\sum_{n=1+Y_{N',*}}^{N'} R_{n,*}\right]}_{\stackrel{\text{def}}{=}\mathfrak{B}_2},$$

where the second equality is due to (11) and (12) (applied to the optimal action).

The term $\mathfrak{A}$ can be bounded as

$$|\mathfrak{A}| \leq \left|\mathfrak{B}_1\right| + |\mathfrak{B}_2|,$$

where

$$|\mathfrak{B}_1| \leq \sum_{k\neq *} \mathbb{E}\left[Y_{N',k}\left|\bar{R}_{Y_{N',k},k}\right|\right] \leq B\sum_{k\neq *}\mathbb{E}\left[Y_{N',k}\right]$$

and

$$|\mathfrak{B}_2| \leq \mathbb{E}\left[\sum_{n=1+Y_{N',*}}^{N'} |R_{n,*}|\right] \leq B\mathbb{E}\left[N' - Y_{N',*}\right] \leq B\sum_{k\neq *}\mathbb{E}\left[Y_{N',k}\right].$$

Substituting the bounds on $\mathfrak{A}, \mathfrak{B}_1$, and $\mathfrak{B}_2$ into (B.3), we then get

$$\left|\mathbb{E}\left[\bar{R}_{N'} - \mu_*\right]\right| \leq \frac{2B}{N'}\sum_{k\neq *}\mathbb{E}\left[Y_{N',k}\right] + \frac{\beta}{\sqrt{N'}}.$$

Now using the result in Lemma 1,

$$\begin{aligned}
\left|\mathbb{E}\left[\bar{R}_{N'} - \mu_*\right]\right| &\leq \frac{2B}{N'}\sum_{k:\Delta_k>0}\left(1 + 4\left(\frac{\sigma^2}{\alpha^2}z^\gamma + \frac{\alpha^2}{\Delta_k^2}\right)\sqrt{N'}\right) + \frac{\beta}{\sqrt{N'}} \\
&\leq \frac{1}{\sqrt{N'}}\left\{2B\sum_{k:\Delta_k>0}\left(1 + 4\left(\frac{\sigma^2}{\alpha^2}z^\gamma + \frac{\alpha^2}{\Delta_k^2}\right)\right) + \beta\right\} \\
&\leq \frac{1}{\sqrt{N'}}\left\{2B(K-1)\left(1 + 4\left(\frac{\sigma^2}{\alpha^2}z^\gamma + \frac{\alpha^2}{\Delta^2}\right)\right) + \beta\right\} \\
&\leq \frac{1}{\sqrt{N'}}\underbrace{\left\{2B(K-1)\left(1 + 4\left(\frac{\sigma^2}{\rho^2} + \frac{\rho^2}{\Delta^2}\right)\right) + \beta\right\}}_{\stackrel{\text{def}}{\leq}\beta'},
\end{aligned}$$

where the last inequality is according to the settings of α, γ, and γ'. The first bound (18) is then reached.

Proof of (19). We rewrite $\bar{R}_{N'} - \mu_*$ as

$$\begin{aligned}
\bar{R}_{N'} - \mu_* &= \frac{1}{N'}\sum_{k=1}^{K} Y_{N',k}\bar{R}_{Y_{N',k},k} - \mu_* \\
&= \frac{1}{N'}\sum_{k\neq *} Y_{N',k}\bar{R}_{Y_{N',k},k} + \frac{1}{N'}Y_{N',*}\bar{R}_{Y_{N',*},*} - \frac{1}{N'}\sum_{n=1}^{N'} R_{n,*} + \bar{R}_{N',*} - \mu_*.
\end{aligned}$$

Then apply Jensen's inequality:

$$\left(\bar{R}_{N'} - \mu_*\right)^2 \leq \frac{3}{(N')^2}\underbrace{\left(\sum_{k\neq *} Y_{N',k}\bar{R}_{Y_{N',k},k}\right)^2}_{\stackrel{\text{def}}{=}\mathfrak{D}_1} + \frac{3}{(N')^2}\underbrace{\left(Y_{N',*}\bar{R}_{Y_{N',*},*} - \sum_{n=1}^{N'} R_{n,*}\right)^2}_{\stackrel{\text{def}}{=}\mathfrak{D}_2} + 3\left(\bar{R}_{N',*} - \mu_*\right)^2.$$

Take expectation at both sides, and apply (16) to the last term:

$$\mathbb{E}\left[\left(\bar{R}_{N'} - \mu_*\right)^2\right] \leq \frac{3}{(N')^2}\mathbb{E}[\mathfrak{D}_1] + \frac{3}{(N')^2}\mathbb{E}[\mathfrak{D}_2] + \frac{3\sigma^2}{N'}z^\gamma.$$

The term $\mathfrak{D}_1$ and $\mathfrak{D}_2$ can be bounded as

$$\mathfrak{D}_1 \overset{(*)}{\leq} (K-1)\sum_{k\neq *}\left(Y_{N',k}\bar{R}_{Y_{N',k},k}\right)^2 \leq (K-1)B^2\sum_{k\neq *}Y_{N',k}^2$$

and

$$\begin{aligned}\mathfrak{D}_2 &= \left(\sum_{n=1}^{Y_{N',*}} R_{n,*} - \sum_{n=1}^{N'} R_{n,*}\right)^2 = \left(\sum_{n=1+Y_{N',*}}^{N'} R_{n,*}\right)^2 \\ &\leq B^2\left(N' - Y_{N',*}\right)^2 = B^2\left(\sum_{k\neq *}Y_{N',k}\right)^2 \overset{(*)}{\leq} (K-1)B^2\sum_{k\neq *}Y_{N',k}^2\end{aligned}$$

where in $(*)$ we use Jensen's inequality.

Putting all these together yields:

$$\begin{aligned}\mathbb{E}\left[\left(\bar{R}_{N'}-\mu_*\right)^2\right] &\leq \frac{6B^2(K-1)}{(N')^2}\sum_{k\neq *}\mathbb{E}\left[Y_{N',k}^2\right] + \frac{3\sigma^2}{N'}z^\gamma \\ &\leq \frac{96B^2(K-1)}{(N')^2}\sum_{k\neq *}\left(\frac{\sigma^2}{\alpha^2}(N')^{2.5}z^\gamma + \frac{\alpha^4}{\Delta_k^4}N'\right) + \frac{3\sigma^2}{N'}z^\gamma \\ &\leq \frac{z^{\gamma'}}{N'}\left\{96B^2(K-1)^2\left(\frac{\sigma^2}{\alpha^2}(N')^{1.5}z^{\gamma-\gamma'} + \frac{\alpha^4}{\Delta^4 z^{\gamma'}}\right) + 3\sigma^2 z^{\gamma-\gamma'}\right\},\end{aligned}$$

where the second inequality is due to Lemma 1.

Substituting the settings of α and γ' into the above gives:

$$\mathbb{E}\left[\left(\bar{R}_{N'}-\mu_*\right)^2\right] \leq \frac{z^{\gamma'}}{N'}\left\{96B^2(K-1)^2\left(\frac{\sigma^2}{\rho^2}(N')^{1.5}z^{1.5} + \frac{\rho^4}{\Delta^4}\right) + 3\sigma^2 z^{\gamma-\gamma'}\right\}.$$

On the other hand, using the assumption $z \leq \frac{1}{N'}$, the above bound can be simplified to

$$\mathbb{E}\left[\left(\bar{R}_{N'}-\mu_*\right)^2\right] \leq \frac{z^{\gamma'}}{N'}\underbrace{\left\{96B^2(K-1)^2\left(\frac{\sigma^2}{\rho^2}+\frac{\rho^4}{\Delta^4}\right)+3\sigma^2\right\}}_{\substack{\text{def}\\ \leq (\sigma')^2}},$$

which is (19). The proof is then completed.

Appendix C. Proof of Theorem 3

The equality in (22) is trivial, as by construction the R-field of the root node equals to the average of the objective values of all solutions generated in the algorithm. So below we will focus on proving the inequality.

The proof consists of three parts. The first part is to prove by induction a bound for the root node of the tree built in MCCTS. The second part is to specify the constants involved in the induction, which is then used to derive a detailed convergence rate. The last part discusses the setting of α, which is a sufficient condition for the convergence.

In the following we will apply Theorem 2 recursively to the tree nodes. When referring to the constants $\alpha, \beta, \gamma, \sigma$ involved in the assumptions of Theorem 2, we include a subscript l to denote their values when we are concerning a tree node of depth l.

Part A: a bound on the root node.

In MCCTS every action set $\mathcal{A}$ satisfies Definition 1, it is then clear that MCCTS will not generate any infeasible state. Then, by assumption, every tree built in MCCTS has a maximal depth L. Without loss of generality, we assume the tree depth is exactly L. Let S_L be a terminal state at depth L. Let a state S_{L-1} at depth $L-1$ be the immediate predecessor state of S_L. Let $\mathcal{A}_{L-1}$ be the action set associated with S_{L-1}.

By the bounded depth assumption, for any $c \in \mathcal{A}_{L-1}$, $S_{L-1}\cup\{c\}$ is a terminal state and has a feasible solution to the MAPF problem. In MCCTS, each time this state is visited, the reward is produced via evaluating the objective function over its associated feasible solution. Then, by the definition of value function, we know

$$\bar{R}(S_{L-1}\cup\{c\}) = U(S_{L-1}\cup\{c\}),$$

where $\bar{R}(S_{L-1}\cup\{c\})$ is the average reward stored in the R-field of the node $S_{L-1}\cup\{c\}$. Now denote the number of visits to this node as $N(S_{L-1}\cup\{c\})$, which is exactly the value stored in the T-field. We then have two trivial bounds:

$$\begin{aligned}&\mathbb{E}\left[\bar{R}(S_{L-1}\cup\{c\})\right]-U(S_{L-1}\cup\{c\})=0\le\frac{\beta_L}{\sqrt{N(S_{L-1}\cup\{c\})}}\\&\mathbb{E}\left[\left(\bar{R}(S_{L-1}\cup\{c\})-U(S_{L-1}\cup\{c\})\right)^2\right]=0\le\frac{\sigma_L^2}{N(S_{L-1}\cup\{c\})}z^{\gamma_L}\end{aligned}\tag{C.1}$$

hold for all $\beta_L,\sigma_L,\gamma_L\ge 0$. The above inequalities coincide with the assumptions in Theorem 2. In addition, since the rewards are bounded by B, we can apply Theorem 2 to the state S_{L-1} (by specifying $N'=N(S_{L-1})$) as:

$$\mathbb{E}\left[\bar{R}(S_{L-1})\right]-V(S_{L-1};\mathcal{A}_{L-1})\le\frac{\beta_{L-1}}{\sqrt{N(S_{L-1})}}\tag{C.2}$$

$$\mathbb{E}\left[\left(\bar{R}(S_{L-1})-V(S_{L-1};\mathcal{A}_{L-1})\right)^2\right]\le\frac{\sigma_{L-1}^2}{N(S_{L-1})}z^{\gamma_{L-1}}\tag{C.3}$$

for some $\beta_{L-1},\sigma_{L-1},\gamma_{L-1}$. Here $N(S_{L-1})$ is the number of visits to the state S_{L-1}, stored in the T-field of the corresponding node.

According to the bounded depth assumption, (C.2) and (C.3) hold for any random instance of $\mathcal{A}_{L-1}$. Therefore, we can take expectation over $\mathcal{A}_{L-1}$ at both sides of (C.2). We then obtain, with (8), the following inequality

$$\mathbb{E}\left[\bar{R}(S_{L-1})\right]-U(S_{L-1})\le\frac{\beta_{L-1}}{\sqrt{N(S_{L-1})}}.\tag{C.4}$$

Similarly, a bound on the second-order moment for the node S_{L-1} can be derived as

$$\begin{aligned}\mathbb{E}\left[\left(\bar{R}(S_{L-1})-U(S_{L-1})\right)^2\right]&\overset{(8)}{=}\mathbb{E}\left[\left(\bar{R}(S_{L-1})-\mathbb{E}\left[V(S_{L-1};\mathcal{A}_{L-1})\right]\right)^2\right]\\&\overset{(*)}{\le}\mathbb{E}\left[\left(\bar{R}(S_{L-1})-V(S_{L-1};\mathcal{A}_{L-1})\right)^2\right]\\&\overset{(C.3)}{\le}\frac{\sigma_{L-1}^2}{N(S_{L-1})}z^{\gamma_{L-1}},\end{aligned}\tag{C.5}$$

where $(*)$ is due to Jensen's inequality.

It can be observed from (C.4) and (C.5) that the two main assumptions of Theorem 2 continue to hold for the node S_{L-1}. It suggests that by induction such bounds hold for all nodes in the tree. In particular, we have

$$\mathbb{E}\left[\bar{R}(S_0)\right]-U(S_0)\le\frac{\beta_0}{\sqrt{N(S_0)}}.$$

In this case, $N(S_0)$ is the number of visits to the root, and therefore $N(S_0)=N$. By Theorem 1, $U(S_0)=F_*$. We then obtain

$$\mathbb{E}[root.R]-F_*\le\frac{\beta_0}{\sqrt{N}}.\tag{C.6}$$

Part B: the detailed convergence rate.

Now we bound the term β_0 in (C.6). This can be achieved by properly setting the constants β_l and σ_l in the above induction. First, Theorem 2 requires the following relations to hold.

$$\begin{cases}\beta_{l-1} & \ge 2B(K-1)\left(1+4\left(\frac{\sigma_l^2}{\rho^2}+\frac{\rho^2}{(\Delta^{[l]})^2}\right)\right)+\beta_l\\ \sigma_{l-1}^2 & \ge 96B^2(K-1)^2\left(\frac{\sigma_l^2}{\rho^2}+\frac{\rho^4}{(\Delta^{[l]})^4}\right)+3\sigma_l^2,\end{cases}$$

where $\Delta^{[l]}$ is the minimal suboptimal gap for the nodes at depth l. For the MAPF problem under consideration, we have $\Delta^{[l]}\ge 1$ since each motion costs a unit of time. To meet the above requisition, we can simply set β_{l-1} and σ_{l-1} to match the upper bound of the right-hand side in the above inequalities:

$$\begin{cases}\beta_{l-1} & =2B(K-1)\left(1+\frac{4\sigma_l^2}{\rho^2}+4\rho^2\right)+\beta_l\\ \sigma_{l-1}^2 & =96B^2(K-1)^2\left(\frac{\sigma_l^2}{\rho^2}+\rho^4\right)+3\sigma_l^2.\end{cases}\tag{C.7}$$

Note first that the base case in (C.1) holds for $\sigma_L=0$. This admits unrolling the term σ_l in (C.7) as

$$\sigma_l^2=\phi\rho^4\frac{\left(\frac{\phi}{\rho^2}+3\right)^{L-l}-1}{\frac{\phi}{\rho^2}+2}\le\rho^6\left(\frac{\phi}{\rho^2}+3\right)^{L-l},\tag{C.8}$$

where $\phi=96B^2(K-1)^2$.

Then, we bound the term β_0 using (C.7) and (C.8). We first note that $\beta_L = 0$ is a correct setting for the base case in (C.1). Therefore, we have

$$\begin{aligned}
\beta_0 &\overset{(C.7)}{=} L\psi(1+4\rho^2) + \frac{4\psi}{\rho^2}\sum_{l=1}^{L}\sigma_l^2 \\
&\overset{(C.8)}{\leq} L\psi(1+4\rho^2) + 4\psi\rho^4\sum_{l=1}^{L}\left(\frac{\phi}{\rho^2}+3\right)^{L-l} \\
&= L\psi(1+4\rho^2) + 4\psi\rho^4\frac{\left(\frac{\phi}{\rho^2}+3\right)^{L}-1}{\frac{\phi}{\rho^2}+2} \\
&\leq L\psi(1+4\rho^2) + 2\psi\rho^4\left(\frac{\phi}{\rho^2}+3\right)^{L},
\end{aligned}$$

where $\psi = 2B(K-1)$.

Substituting this into (C.6) and using $\psi \leq 2BK$ and $\phi \leq 96B^2K^2$ yield

$$\begin{aligned}
\mathbb{E}[root.R] - F_* &\leq \frac{\psi}{\sqrt{N}}\left(L(1+4\rho^2) + 2\rho^4\left(\frac{\phi}{\rho^2}+3\right)^{L}\right) \\
&\leq \frac{2BK}{\sqrt{N}}\left(L(1+4\rho^2) + 2\rho^4\left(\frac{96B^2K^2}{\rho^2}+3\right)^{L}\right).
\end{aligned}$$

Then the final bound is obtained.

Part C: the setting of α_l.

By Theorem 2, we need two conditions to make the above induction work:

$$\begin{cases} \gamma_l & \geq \frac{3}{2} \\ \gamma_{l-1} & = \frac{2}{3}\gamma_l - 1. \end{cases} \forall l \in \{1,\dots,L\}$$

An appropriate setting can be obtained via specifying $\gamma_0 = 0$; this leads to

$$\gamma_l = 3\left(1.5^l - 1\right). \tag{C.9}$$

On the other hand, the parameter z should satisfy $z \leq \frac{1}{N(S_l)}$ for all $l \in \{1,\dots,L\}$, where $N(S_l)$ is the number of visits to S_l at depth l. We therefore choose z to be

$$z = \frac{1}{N} = \frac{1}{N(S_0)} < \frac{1}{N(S_1)} < \dots \frac{1}{N(S_L)}.$$

Combining this with (17) and (C.9), we get the setting for α_l as

$$\alpha_l = \rho\,(1/N)^{\frac{\gamma_l}{6}-\frac{1}{4}} = \rho N^{\frac{3}{4}-\frac{1.5^l}{2}}.$$

This explains how the parameter α_l is set in MCCTS.

Appendix D. Proof of Corollary 1

This corollary is straightforward. Since the R-field in the root is averaged over the rewards obtained during the search, one must in some iteration of the search find a solution with the objective value better than $root.R$:

$$\mathbb{E}\left[F(X_{N,*}) - F_*\right] \leq \mathbb{E}\left[root.R - F_*\right].$$

The desired result follows directly from using Markov's inequality and the bound in Theorem 3.

Appendix E. Proof of Lemma 1

Proof of (B.1). Since each action must be selected at least once, we can therefore rewrite $Y_{N',k}$ as

$$Y_{N',k} = \sum_{n=1}^{N'}\mathbb{I}\{k_n = k\} = 1 + \sum_{n=K+1}^{N'}\underbrace{\mathbb{I}\left\{k_n = k, \bar{R}_{Y_{n-1,k},k} - \alpha\frac{(n-1)^{1/4}}{\sqrt{Y_{n-1,k}}} \leq \bar{R}_{Y_{n-1,*},*} - \alpha\frac{(n-1)^{1/4}}{\sqrt{Y_{n-1,*}}}\right\}}_{\overset{\text{def}}{=}\mathfrak{C}}. \tag{E.1}$$

In order to bound the event $\mathfrak{C}$, we define correspondingly three sub-events:

$$\mathfrak{C}_1 = \left\{ k_n = k, \bar{R}_{Y_{n-1,k},k} + \alpha \frac{(n-1)^{1/4}}{\sqrt{Y_{n-1,k}}} < \mu_k \right\},$$

$$\mathfrak{C}_2 = \left\{ k_n = k, \bar{R}_{Y_{n-1,*},*} - \alpha \frac{(n-1)^{1/4}}{\sqrt{Y_{n-1,*}}} > \mu_* \right\},$$

$$\mathfrak{C}_3 = \left\{ k_n = k, \mu_k - 2\alpha \frac{(n-1)^{1/4}}{\sqrt{Y_{n-1,k}}} \le \mu_* \right\} = \left\{ k_n = k, Y_{n-1,k} \le 4\alpha^2 \frac{\sqrt{n-1}}{\Delta_k^2} \right\}.$$

One can verify that $\mathfrak{C} \subset \mathfrak{C}_1 \vee \mathfrak{C}_2 \vee \mathfrak{C}_3$. We therefore have

$$\mathbb{E}\left[Y_{N',k}\right] \le 1 + \sum_{n=K+1}^{N'} \left(\mathbb{P}[\mathfrak{C}_1; n] + \mathbb{P}[\mathfrak{C}_2; n] \right) + \mathbb{E}\left[\sum_{n=K+1}^{N'} \mathbb{I}\{\mathfrak{C}_3\} \right], \tag{E.2}$$

where $\mathbb{P}[\cdot; n]$ denotes the probability conditioned on n.

The term $\mathbb{P}[\mathfrak{C}_1; n]$ can be bounded as

$$\mathbb{P}[\mathfrak{C}_1; n] \le \mathbb{P}\left[\left(\bar{R}_{Y_{n-1,k},k} - \mu_k \right)^2 \ge \alpha^2 \frac{\sqrt{n-1}}{Y_{n-1,k}}; n \right] \le \frac{\mathbb{E}\left[\left(\bar{R}_{Y_{n-1,k},k} - \mu_k \right)^2; n \right]}{\alpha^2 \sqrt{n-1}} Y_{n-1,k},$$

where the first inequality is derived from the definition of $\mathfrak{C}_1$, the second inequality is due to Markov's inequality, and $\mathbb{E}[\cdot; n]$ denotes the expectation conditioned on n. Note that the numerator in the right-most side can be bounded, for every n, using the assumption in (16). This yields:

$$\mathbb{P}[\mathfrak{C}_1; n] \le \frac{\sigma^2}{\alpha^2 \sqrt{n-1}} z^\gamma.$$

The term $\mathbb{P}[\mathfrak{C}_2; n]$ can be bounded in a similar way as

$$\mathbb{P}[\mathfrak{C}_2; n] \le \frac{\mathbb{E}\left[|\bar{R}_{Y_{n-1,*},*} - \mu_*|^2; n \right]}{\alpha^2 \sqrt{n-1}} Y_{n-1,*} \le \frac{\sigma^2}{\alpha^2 \sqrt{n-1}} z^\gamma.$$

The last term in (E.2) is bounded as:

$$\sum_{n=K+1}^{N'} \mathbb{I}\{\mathfrak{C}_3\} = \sum_{n=K+1}^{N'} \mathbb{I}\left\{ k_n = k, Y_{n-1,k} \le 4\alpha^2 \frac{\sqrt{n-1}}{\Delta_k^2} \right\} \le \frac{4\alpha^2}{\Delta_k^2} \sqrt{N'},$$

which can be proved by contradiction. Concretely, suppose that $\mathfrak{C}_3$ becomes true at more than $\psi \overset{\text{def}}{=} \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'}$ time steps, and let τ be the time step at which $\mathfrak{C}_3$ is true for the ψ-th time. Then we know the action k must have been selected at least $\psi + 1$ times until time τ (including the one initial selection). Therefore, for all $n > \tau$, $Y_{n,k} > \psi = \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'} \ge \frac{4\alpha^2}{\Delta_k^2}\sqrt{n}$. This means the event $\mathfrak{C}_3$ cannot be true after time τ, yielding a contradiction.

Putting all above bounds into (E.2) gives

$$\mathbb{E}\left[Y_{N',k}\right] \le 1 + \sum_{n=K+1}^{N'} \frac{2\sigma^2}{\alpha^2 \sqrt{n-1}} z^\gamma + \frac{4\alpha^2}{\Delta_k^2} \sqrt{N'}.$$

The bound in (B.1) then follows from the assumption $z \le 1$ and the inequality

$$\frac{1}{\sqrt{1}} + \frac{1}{\sqrt{2}} + \ldots + \frac{1}{\sqrt{N'}} \le 2\sqrt{N'}. \tag{E.3}$$

Proof of (B.2).

$$\begin{aligned} \mathbb{E}\left[Y_{N',k}^2\right] &= \mathbb{E}\left[Y_{N',k}^2 \mathbb{I}\left\{ Y_{N',k} \le \left\lfloor \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'} \right\rfloor \right\} \right] + \mathbb{E}\left[Y_{N',k}^2 \mathbb{I}\left\{ Y_{N',k} > \left\lfloor \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'} \right\rfloor \right\} \right] \\ &\le \frac{16\alpha^4}{\Delta_k^4} N' + (N')^2 \underbrace{\mathbb{P}\left\{ Y_{N',k} > \left\lfloor \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'} \right\rfloor \right\}}_{\overset{\text{def}}{=} \mathfrak{E}}. \end{aligned} \tag{E.4}$$

Consider now the event $\mathfrak{E}$. Define $u = \left\lfloor \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'} \right\rfloor$ and two sub-events $\mathfrak{E}_1$ and $\mathfrak{E}_2$ as

$$\mathfrak{E}_1 = \left\{ \forall n \in [u, N'], L_{k,u,n} > \mu_* \right\},$$
$$\mathfrak{E}_2 = \left\{ \forall s \in [1, N'-u], L_{*,s,u+s} < \mu_* \right\},$$

where

$$\begin{cases} L_{k,u,n} & = \bar{R}_{u,k} - \alpha \frac{n^{1/4}}{\sqrt{u}} \\ L_{*,s,u+s} & = \bar{R}_{s,*} - \alpha \frac{(u+s)^{1/4}}{\sqrt{s}}. \end{cases}$$

We then claim

$$\mathfrak{E}_1 \wedge \mathfrak{E}_2 \subset \neg\mathfrak{E}. \tag{E.5}$$

Equation (E.5) can be proved by contradiction. Let us assume the event $\mathfrak{E}$ happens when $\mathfrak{E}_1$ and $\mathfrak{E}_2$ both hold. Under this assumption, $\mathfrak{E} = \mathbb{I}\{Y_{N',k} > u\}$ and it suggests that the k-th action has been selected for at least $u+1$ times. Let n' be the time step that the k-th action is chosen for the u-th time, i.e., $u = Y_{n',k}$. We then have

$$N' > n' \geq u + Y_{n',*} \tag{E.6}$$

and

$$\bar{R}_{Y_{n',k},k} - \alpha \frac{(n')^{1/4}}{\sqrt{Y_{n',k}}} = \bar{R}_{u,k} - \alpha \frac{(n')^{1/4}}{\sqrt{u}} > \mu_* > \bar{R}_{Y_{n',*},*} - \alpha \frac{(u + Y'_{n',*})^{1/4}}{\sqrt{Y_{n',*}}} \geq \bar{R}_{Y_{n',*},*} - \alpha \frac{(n')^{1/4}}{\sqrt{Y_{n',*}}}.$$

The first inequality is by the definition of $\mathfrak{E}_1$ with specification of $u = Y_{n',k}$. The second inequality is by the definition of $\mathfrak{E}_2$ with specification of $s = Y_{n',*}$. The last inequality is due to (E.6). It then suggests that the k-th action would not be chosen any more after time step n', leading to the contradiction. Therefore, we have proved that the claim (E.5) is correct.

We then bound the probability of event $\mathfrak{E}$ using (E.5):

$$\begin{aligned} \mathbb{P}\{\mathfrak{E}\} &\leq \mathbb{P}\{\neg\mathfrak{E}_1\} + \mathbb{P}\{\neg\mathfrak{E}_2\} \\ &= \mathbb{P}\{\exists n \in [u, N'], L_{k,u,n} \leq \mu_*\} + \mathbb{P}\{\exists s \in [1, N'-u], L_{*,s,u+s} \geq \mu_*\} \\ &\leq \sum_{n=u}^{N'} \mathbb{P}\{L_{k,u,n} \leq \mu_*; n\} + \sum_{s=1}^{N'-u} \mathbb{P}\{L_{*,s,u+s} \geq \mu_*; s\} \\ &= \sum_{n=u}^{N'} \mathbb{P} \underbrace{\left\{ \bar{R}_{u,k} - \alpha \frac{n^{1/4}}{\sqrt{u}} \leq \mu_*; n \right\}}_{\stackrel{\text{def}}{=}\mathfrak{E}_3} + \sum_{s=1}^{N'-u} \mathbb{P} \underbrace{\left\{ \bar{R}_{s,*} - \alpha \frac{(u+s)^{1/4}}{\sqrt{s}} \geq \mu_*; s \right\}}_{\stackrel{\text{def}}{=}\mathfrak{E}_4}. \end{aligned} \tag{E.7}$$

The probability of $\mathfrak{E}_4$ can be bounded directly with Markov's inequality and the assumption (16):

$$\mathbb{P}\{\mathfrak{E}_4\} \leq \mathbb{E}\left[\left(\bar{R}_{s,*} - \mu_*\right)^2; s \right] \frac{s}{\alpha^2\sqrt{u+s}} \leq \frac{\sigma^2}{\alpha^2\sqrt{u+s}} z^\gamma. \tag{E.8}$$

To bound $\mathfrak{E}_3$, recalling the definition of u, which yields, for all $1 \leq n \leq N'$,

$$u + 1 \geq \frac{4\alpha^2}{\Delta_k^2}\sqrt{N'} \geq \frac{4\alpha^2}{\Delta_k^2}\sqrt{n} \Rightarrow \Delta_k \geq 2\alpha \frac{n^{1/4}}{\sqrt{u+1}} \geq \sqrt{2}\alpha \frac{n^{1/4}}{\sqrt{u}}.$$

This suggests to bound $\mathfrak{E}_3$ as

$$\mathbb{I}\{\mathfrak{E}_3\} = \mathbb{I}\left\{ \bar{R}_{u,k} - \alpha \frac{n^{1/4}}{\sqrt{u}} \leq \mu_* \right\} = \mathbb{I}\left\{ \bar{R}_{u,k} - \alpha \frac{n^{1/4}}{\sqrt{u}} \leq \mu_k - \Delta_k \right\} \leq \mathbb{I}\left\{ \bar{R}_{u,k} - \mu_k \leq (1-\sqrt{2})\alpha \frac{n^{1/4}}{\sqrt{u}} \right\}.$$

Now it's ready to apply Markov's inequality to bound $\mathbb{P}\{\mathfrak{E}_3\}$:

$$\mathbb{P}\{\mathfrak{E}_3\} \leq \mathbb{E}\left[\left(\bar{R}_{u,k} - \mu_k\right)^2; n \right] \frac{u}{\alpha^2(1-\sqrt{2})^2\sqrt{n}} \stackrel{(16)}{\leq} \frac{6\sigma^2}{\alpha^2\sqrt{n}} z^\gamma. \tag{E.9}$$

Substituting (E.8) and (E.9) into (E.7), we can reach

$$\mathbb{P}\{\mathfrak{E}\} \leq \sum_{n=u}^{N'} \frac{6\sigma^2}{\alpha^2\sqrt{n}} z^\gamma + \sum_{s=1}^{N'-u} \frac{\sigma^2}{\alpha^2\sqrt{u+s}} z^\gamma \stackrel{(E.3)}{\leq} 14\sqrt{N'}\frac{\sigma^2}{\alpha^2} z^\gamma.$$

The desired inequality (B.2) can be obtained by substituting the above into (E.4).

Appendix F. Proof of Proposition 1

We can bound the number of iterations performed by bounding the number of different constraints generated. Take the vertex constraint as an example. As defined in Section 5.1, each vertex constraint is a 3-tuple consisting of the time step when the constraint should be activated, the agent to whom the constraint is applied, and the location where we need to avoid the vertex conflict. Since we have assumed that the solution found has an objective value upper bounded by B, there are at most $mB|\mathcal{V}|$ different vertex constraints. Similarly, we know the number of different edge constraints is upper bounded by $mB|\mathcal{E}|$. It means we have at most $mB(|\mathcal{V}| + |\mathcal{E}|)$ different constraints. As in each iteration only one distinct constraint is included, the random simulation must terminate within $mB(|\mathcal{V}| + |\mathcal{E}|)$ iterations. Each iteration calls the suboptimal SAPF solver for two times, so the number of calls to the suboptimal SAPF solver is upper bounded by $2mB(|\mathcal{V}| + |\mathcal{E}|)$.

Appendix G. Proof of Corollary 2

Each iteration of MCCTS involves one call to an optimal SAPF solver in the main loop (Line 25 of Algorithm 1) and several calls to a suboptimal SAPF solver in the random simulation (Line 6 of Algorithm 2). The number of calls to the suboptimal SAPF solver is, by Proposition 1, upper bounded by $2mB(|\mathcal{V}| + |\mathcal{E}|)$. So we have

$$\tilde{N} \leq N \times (1 + 2mB(|\mathcal{V}| + |\mathcal{E}|)).$$

Plugging the above into the bound in (22) gives the desired convergence rate measured in terms of $\tilde{N}$.

References

[1] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.K.S. Kumar, R. Barták, E. Boyarski, Multi-agent pathfinding: definitions, variants, and benchmarks, in: Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16–17 July 2019, 2019, pp. 151–159.

[2] O. Salzman, R. Stern, Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems, in: Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9–13, 2020, 2020, pp. 1711–1715.

[3] I. Pohl, Heuristic search viewed as path finding in a graph, Artif. Intell. 1 (3) (1970) 193–204, https://doi.org/10.1016/0004-3702(70)90007-X.

[4] J. Yu, S.M. LaValle, Structure and intractability of optimal multi-robot path planning on graphs, in: Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013.

[5] H. Ma, J. Li, T.K.S. Kumar, S. Koenig, Lifelong multi-agent path finding for online pickup and delivery tasks, in: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8–12, 2017, 2017, pp. 837–845.

[6] A. Adler, M. de Berg, D. Halperin, K. Solovey, Efficient multi-robot motion planning for unlabeled discs in simple polygons, IEEE Trans. Autom. Sci. Eng. 12 (4) (2015) 1309–1317, https://doi.org/10.1109/TASE.2015.2470096.

[7] N.R. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Trans. Comput. Intell. AI Games 4 (2) (2012) 144–148, https://doi.org/10.1109/TCIAIG.2012.2197681.

[8] T. Standley, Finding optimal solutions to cooperative pathfinding problems, Proc. AAAI Conf. Artif. Intell. 24 (1) (2010) 173–178.

[9] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66, https://doi.org/10.1016/j.artint.2014.11.006.

[10] R. Luna, K.E. Bekris, Push and swap: fast cooperative path-finding with completeness guarantees, in: IJCAI, 2011.

[11] J. Yu, S.M. LaValle, Planning optimal paths for multiple robots on graphs, in: 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 3612–3617.

[12] E. Lam, P.L. Bodic, D.D. Harabor, P.J. Stuckey, Branch-and-cut-and-price for multi-agent pathfinding, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, Macao, China, 2019, pp. 1289–1296.

[13] G. Sharon, R. Stern, M. Goldenberg, A. Felner, The increasing cost tree search for optimal multi-agent pathfinding, Artif. Intell. 195 (2013) 470–495, https://doi.org/10.1016/j.artint.2012.11.006.

[14] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N.R. Sturtevant, J. Schaeffer, R. Holte, Partial-expansion A* with selective node generation, in: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Toronto, Ontario, Canada, July 22–26, 2012, 2012.

[15] O. Kaduri, E. Boyarski, R. Stern, Experimental evaluation of classical multi agent path finding algorithms, Proc. Int. Symp. Comb. Search 12 (1) (2021) 126–130.

[16] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, E. Shimony, ICBS: improved conflict-based search algorithm for multi-agent pathfinding, in: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, AAAI Press, Buenos Aires, Argentina, 2015, pp. 740–746.

[17] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T.K.S. Kumar, S. Koenig, Adding heuristics to conflict-based search for multi-agent path finding, in: Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, the Netherlands, June 24–29, 2018, 2018, pp. 83–87.

[18] J. Li, A. Felner, E. Boyarski, H. Ma, S. Koenig, Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search, 2019, pp. 442–449.

[19] H. Zhang, J. Li, P. Surynek, S. Koenig, T.K.S. Kumar, Multi-agent path finding with mutex propagation, Proc. Int. Conf. Autom. Plan. Sched. 30 (2020) 323–332.

[20] J. Li, W. Ruml, S. Koenig, EECBS: a bounded-suboptimal search for multi-agent path finding, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2021, pp. 12353–12362.

[21] E. Steinmetz, M. Gini, More trees or larger trees: parallelizing Monte Carlo tree search, IEEE Trans. Games 13 (3) (2021) 315–320, https://doi.org/10.1109/TG.2020.3048331.

[22] J. Dubois-Lacoste, M. López-Ibáñez, T. Stützle, Anytime Pareto local search, Eur. J. Oper. Res. 243 (2) (2015) 369–385, https://doi.org/10.1016/j.ejor.2014.10.062.

[23] J. Li, Z. Chen, D. Harabor, P.J. Stuckey, S. Koenig, Anytime multi-agent path finding via large neighborhood search, in: Twenty-Ninth International Joint Conference on Artificial Intelligence, vol. 4, 2021, pp. 4127–4135.

[24] K. Vedder, J. Biswas, X*: anytime multi-agent path finding for sparse domains using window-based iterative repairs, Artif. Intell. 291 (2021) 103417, https://doi.org/10.1016/j.artint.2020.103417.

[25] K. Okumura, Y. Tamura, X. Défago, Iterative refinement for real-time multi-robot path planning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27–Oct. 1, 2021, 2021, pp. 9690–9697.

[26] J.T. Thayer, W. Ruml, Bounded suboptimal search: a direct approach using inadmissible estimates, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16–22, 2011, IJCAI/AAAI, 2011, pp. 674–679.

[27] M. Barer, G. Sharon, R. Stern, A. Felner, Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem, in: Seventh Annual Symposium on Combinatorial Search, 2014.
[28] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 282–293.
[29] L. Cohen, M. Greco, H. Ma, C. Hernández, A. Felner, T.K.S. Kumar, S. Koenig, Anytime focal search with applications, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13–19, 2018, 2018, pp. 1434–1441.
[30] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T.K.S. Kumar, S. Koenig, H. Choset, PRIMAL: pathfinding via reinforcement and imitation multi-agent learning, IEEE Robot. Autom. Lett. 4 (3) (2019) 2378–2385, https://doi.org/10.1109/LRA.2019.2903261.
[31] S. Choudhury, J.K. Gupta, P. Morales, M.J. Kochenderfer, Scalable anytime planning for multi-agent MDPs, in: AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3–7, 2021, 2021, pp. 341–349.
[32] M.C. Fu, A tutorial introduction to Monte Carlo tree search, in: 2020 Winter Simulation Conference, WSC, 2020, pp. 1178–1193.
[33] A. Slivkins, Introduction to multi-armed bandits, Found. Trends Mach. Learn. 12 (1–2) (2019) 1–286, https://doi.org/10.1561/2200000068.
[34] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2) (2002) 235–256, https://doi.org/10.1023/A:1013689704352.
[35] J.-Y. Audibert, R. Munos, C. Szepesvári, Exploration–exploitation tradeoff using variance estimates in multi-armed bandits, Theor. Comput. Sci. 410 (19) (2009) 1876–1902, https://doi.org/10.1016/j.tcs.2009.01.016.
[36] D. Shah, Q. Xie, Z. Xu, Non-asymptotic analysis of Monte Carlo tree search, in: Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, Boston, MA, USA, June, 8–12, 2020, 2020, pp. 31–32.
[37] H.S. Chang, M.C. Fu, J. Hu, S.I. Marcus, A survey of some simulation-based algorithms for Markov decision processes, Commun. Inf. Syst. 7 (1) (2007) 59–92, https://doi.org/10.4310/CIS.2007.v7.n1.a4.
[38] D. Silver, Cooperative pathfinding, in: Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'05, AAAI Press, Marina del Rey, California, 2005, pp. 117–122.
[39] E.D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2) (2002) 201–213, https://doi.org/10.1007/s101070100263.