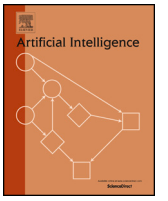
Artificial Intelligence 346 (2025) 104349

Contents lists available at ScienceDirect

Artificial Intelligence

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



CBS-Budget (CBSB): A complete and bounded suboptimal search for multi-agent path finding

Jaein Lim [a,*], Panagiotis Tsiotras [a,b]

[a] The Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, 30332, GA, USA
[b] The Institute for Robotics Intelligent Machines, Georgia Institute of Technology, Atlanta, 30332, GA, USA

ARTICLE INFO

Keywords:
Multi-agent path finding
Conflict based search
Bounded suboptimality

ABSTRACT

Multi-Agent Path Finding (MAPF) is the problem of finding a collection of conflict-free paths for a team of multiple agents while minimizing some global cost, such as the sum of the travel time of all agents, or the travel time of the last agent. Conflict Based Search (CBS) is a leading complete and optimal MAPF algorithm that lazily explores the joint agent state space, using an admissible heuristic joint plan. Such an admissible heuristic joint plan is computed by combining individual shortest paths computed without considering inter-agent conflicts, and becoming gradually more informed as constraints are added to the individual agents' path-planning problems to avoid discovered conflicts. In this paper, we seek to speed up CBS by finding a more informed heuristic joint plan that is bounded. We first propose the budgeted Class-Ordered A* (bCOA*), a novel algorithm that finds the least-cost path with the minimal number of conflicts that is upper bounded in terms of path length. Then, we propose a novel bounded-cost variant of CBS, called CBS-Budget (CBSB) by using bCOA* search at the low-level search of the CBS and by using a modified focal search at the high-level search of the CBS. We prove that CBSB is complete and bounded-suboptimal. In our numerical experiments, CBSB finds a near-optimal solution for hundreds of agents within a fraction of a second. CBSB shows state-of-the-art performance, comparable to Explicit Estimation CBS (EECBS), an enhanced recent version of CBS. On the other hand, CBSB is much easier to implement than EECBS, since only one priority queue at the low-level search is needed, as in CBS, and only two priority queues at the high-level search are needed, as in Enhanced CBS (ECBS).

1. Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding collision-free paths for a team of mobile agents from their individual start locations to their individual goal locations, while minimizing the sum of their travel times or the travel time of the last agent. Inspired from numerous real-world applications such as warehouse logistics [1], automated valet parking [2], video games [3], robotics [4] and traffic management [5,6], MAPF has drawn significant attention recently from various research communities. Solving MAPF with theoretical guarantees is imperative for operating many agents in the same cluttered environment, where uncoordinated plans or prioritized plans can result in conflicts such as collisions, interference, or deadlocks.

* Corresponding author.
E-mail addresses: jaeinlim126@gatech.edu (J. Lim), tsiotras@gatech.edu (P. Tsiotras).

https://doi.org/10.1016/j.artint.2025.104349
Received 27 April 2022; Received in revised form 27 April 2025; Accepted 29 April 2025

Available online 8 May 2025

One approach to find a conflict-free path minimizing some global cost is to search in the joint agent search space, that is, in the Cartesian product of the individual agents' search spaces [7,8]. However, the branching factor of the joint search space grows exponentially with the number of agents, prohibiting the use of a regular heuristic search algorithm like A*. For example, for a grid world with 30 agents, where each agent can either move up, down, left, right, or wait, the joint state has about 9.31×10^{20} neighbor states; even enumerating those neighbors to begin the search becomes intractable [9].

Previous complete and optimal MAPF algorithms seek to explore only the neighbors of a joint state that could potentially yield an optimal solution [10–14]. Conflict Based Search (CBS) [15] is a state-of-the-art optimal MAPF algorithm that utilizes a lazy algorithm to delay generation of irrelevant neighbors. CBS delays the generation of irrelevant neighbors by decomposing the joint search space to multiple individual path-planning problems. CBS first computes the shortest path for each individual agent without considering other agents. The combination of the shortest paths is an admissible heuristic plan of the joint space which does not overestimate the true optimal solution cost. CBS then checks if the combination of the shortest paths, or the heuristic plan, is indeed conflict-free. If the heuristic plan is found to be invalid because of inter-agent conflicts between paths, then CBS generates two new heuristic plans by replanning only for the two conflicting agents, one at a time, with the additional constraint of avoiding conflict at the low-level search.[1] CBS continues to choose the heuristic plan with the minimum cost for examination of inter-agent conflicts until a conflict-free plan is found. This minimal generation of admissible heuristic plans makes the exploration of the joint state space efficient and complete.

CBS may still become intractable, however, if there exist many infeasible plans due to conflicts whose cost is lower than the optimal solution cost. As a best-first search, CBS eliminates all those plans whose cost is lower than the optimal solution cost until the optimal solution is found. Recent improvements have made significant progress on CBS by increasing the lower bound of the optimal solution more quickly, that is, by finding a more informed admissible plan using domain-specific knowledge [16–21]. Solving the MAPF problem optimally is, however, NP-hard [22], and the problem still remains unsolvable for a large number of agents and reasonable computational resources.

If a bounded-suboptimal solution is allowed, then CBS can be modified to solve large problem instances. The main idea is to generate more informed, possibly inadmissible, and yet bounded, heuristic joint plans that could result in better progress during the search. This improvement is the main topic of the paper.

Enhanced CBS (ECBS) [23] is a recent variant of CBS that computes a bounded-suboptimal heuristic plan using a focal search. Focal search [24] maintains two priority queues, one for bounding the solution cost using an admissible heuristic and the other for finding an informed, and possibly inadmissible, path using another heuristic. ECBS uses a focal search in both the low- and high-level search of CBS to generate a more informed heuristic plan that is a bounded-suboptimal plan with a fewer number of conflicts.

Recently, a further improvement to ECBS was made by replacing the high-level focal search of ECBS with Explicit Estimation Search (EES) [25]. Focal search becomes inefficient when the two heuristics are negatively correlated. EES [26] mitigates the negative correlation between the two heuristics of focal search, using a third heuristic which may be potentially inadmissible. Explicit Estimation CBS (EECBS) [25] uses EES in the high-level search of CBS to avoid the high-level focal search being stuck locally by considering the potential cost increment upon resolving a conflict. Maintaining three different priority queues requires more memory usage and adds complexity to the algorithm. In fact, the number of frontier nodes (i.e., the number of joint plans) is exponential in the depth of the CBS search (i.e., the number of resolved conflicts) [27], which makes memory limit more of a bottleneck than the time limit in CBS implementations. In addition, incorporating modern implementations of CBS into EECBS is not trivial, as specific rules need to be carefully considered, depending on which priority queue a heuristic joint plan is chosen, adding more complexity to the algorithm.

In this paper, we propose a novel and simple bounded-cost variant of CBS, called CBS-Budget (CBSB). CBSB uses a budgeted Class-Ordered A* (bCOA*) search at the low level of CBS to produce a more informed and still bounded heuristic plan. Similarly to focal search, bCOA* finds the least-cost path for each agent with minimal inclusion of conflicts with other agents and whose cost is upper bounded either by a given budget or by the least-cost path cost. The budget for each agent is incrementally updated after each search if no lower cost path exists, making the budget-induced suboptimality bound more informative. Unlike focal search, bCOA* only uses a single priority queue by leveraging the ideas of colored planning [28–30], to reduce the runtime compared to focal search. This property of bCOA* allows the high-level search of CBSB to explore more heuristic plans, leading to a faster computation time to find a bounded suboptimal solution. We prove the theoretical properties of bCOA* and the completeness and bounded-suboptimality of CBSB. In our experiments on standard MAPF benchmarks [31], CBSB shows state-of-the-art performance compared to the modern implementations of complete and bounded-(sub)optimal MAPF algorithms such as CBS, ECBS, and EECBS.

This paper is organized as follows. We first provide some preliminaries and related work in the next section. Then, we provide detailed descriptions of the proposed algorithms with a formal analysis in the subsequent sections. Finally, we analyze the performance of the proposed algorithm using numerical experiments and we conclude with some suggestions for future work.

2. Preliminaries and relation to prior work

In this section, we formalize the definition of MAPF and provide some background on the baseline CBS algorithm and its improved variants.

[1] In case a conflict of three or more agents occurs, CBS picks any two agents to resolve the conflict at a time. The remaining conflicts are resolved in the subsequent searches.

2.1. Multi-Agent Path Finding (MAPF)

MAPF is the problem of finding a set of collision-free paths for multiple agents in the same graph. Formally, MAPF is defined by an undirected graph $G=(V,E)$ and a set of m agents $\{a_1,\dots,a_m\}$, where each agent a_i has a start vertex $s_i \in V$ and a target vertex $t_i \in V$. Time is discretized and, at each timestep, every agent can either move to an adjacent vertex or wait at the current vertex. Without loss of generality, we assume that all actions have uniform cost.[2] The agents remain at their target vertices after completion.

A path π_i for agent a_i is a sequence of adjacent vertices from s_i to t_i. The cost of a path π_i is its length. We say that two agents have a *conflict* if they occupy the same vertex at the same timestep or if they move along the same edge at the same timestep. A plan is a set of individual agent paths $\{\pi_1,\dots,\pi_m\}$, and a MAPF solution is a plan that is conflict-free. The cost of a plan is the sum of its path lengths. An optimal solution is a solution whose cost is minimal. Note that our algorithm is not limited to this *flowtime* formulation. The cost of a plan can be defined as the longest path length to solve the *makespan* problem [15].

2.2. Conflict Based Search (CBS)

Conflict Based Search (CBS) [15] is a two-level MAPF algorithm. At the low level, each agent finds the shortest path on a time-expanded graph (TEG) [32] with some variant of A*. Each vertex in a TEG is a pair of location and time, and each edge in TEG represents a move from a vertex at one timestep to another vertex at the next timestep or a wait at a vertex for one timestep. Each agent may be subject to a set of constraints that prohibit this agent from occupying a vertex or from moving through an edge at a certain timestep. The constraints are generated at the high level to avoid a collision between the two conflicting agents. Specifically, at the high level, CBS performs a best-first search by constructing a binary constraint tree (CT), where each CT node N contains a plan (possibly with conflicts) consisting of the individual paths computed at the low level, along with the corresponding sum of the costs, denoted as $cost(N)$. When CBS chooses a CT node with the minimum $cost$ to expand, CBS checks if a conflict occurs in the plan of that node. If a conflict exists, then CBS imposes constraints on the two conflicting agents, one at a time, in each of the two children of the CT node to avoid the conflict. With this additional constraint added, the low-level CBS search finds the next shortest path satisfying the constraint for the agent in the child CT node. CBS chooses the next best node prioritized with the best possible cost until no conflict is found.

In essence, CBS generates an admissible heuristic plan by finding the shortest path of individual agents. If the plan has a conflict, then CBS generates the next best plan by replanning the individual shortest paths of the conflicting agents with an additional constraint. The two-level search of CBS produces an admissible heuristic plan that becomes more informed as conflicts are discovered. Each CT node contains a heuristic plan, and a child CT node contains a more informed heuristic plan with additional constraints. The generated plans are prioritized according to their $f = cost + h$ values and the CT node with the lowest f is chosen for expansion using an A*-like search, where $cost$ is the current plan cost and h is an admissible heuristic cost-increment to resolve the conflicts in the CT node. Hence, the chosen CT node for expansion always contains an admissible heuristic plan, ensuring the optimality of CBS.

2.3. Modern improvements

In this section, we briefly introduce some of the recently developed techniques that improve the performance of CBS by computing more informed admissible heuristic plans using domain-specific knowledge. Detailed descriptions of those techniques are deferred to the provided references.

Prioritizing Conflicts [16]: The $cost$-value of a CT node monotonically increases along the depth of CT, because each individual agent finds a path with monotonically increasing length with an additional constraint at the low-level search. If one can find a conflict that would result in a large $cost$-value increase, then CBS can make good progress toward the goal CT node without sacrificing optimality. This observation has motivated the prioritization of conflicts so as to select a conflict that would maximally increase the $cost$-value of a child CT node. The potential cost increase of resolving a conflict can be determined by building a Multi-Valued Decision Diagram (MDD) [33] for each agent, an acyclic directed graph that consists of all shortest paths of the agent.

Symmetry Reasoning [17]: Imposing a more informed constraint to resolve a conflict can benefit the progress of CBS. Symmetry reasoning is a technique that identifies repeated collisions between two agents due to symmetric paths. Domain-specific knowledge can be used to generate symmetry-breaking constraints to generate a more informed and admissible heuristic plan.

Mutex Propagation [18]: Symmetry reasoning identifies three types of conflicts (rectangle, corridor, target) and generates hand-crafted constraints. Mutex propagation is a technique that automates the generation of symmetry-breaking constraints using MDDs. Each level of MDD contains all the reachable states at that time that belong to the shortest paths to the goal satisfying the constraints. Two nodes at the same level of MDDs that belong to two different agents are mutex, if no bypassing conflict-free paths exist. Propagating mutex along the MDD can generate a set of more informed constraints by identifying the unreachable set of states in the MDDs.

Weighted Dependency Graph (WDG) Heuristic [19]: WDG is an admissible heuristic for the high-level search of CBS. Vertices in WDG represent agents and edges represent two agents that are dependent. The dependency of two agents is determined by solving a two-agent MAPF instance for the two agents. If the sum-of-cost of the subproblem is larger than their current sum-of-cost, then the

[2] Our algorithm, however, remains valid also for non-uniform costs.

two agents are dependent. The edge weights are assigned the difference between the two sum-of-costs. As a result, the edge-weighted minimum vertex cover of the graph better approximates the true distance-to-go of a CT node.

Bypassing Conflicts [34]: Bypassing conflict (BP) is a conflict-resolution technique that avoids splitting a CT node into two children CT nodes by simply replacing the plan of the parent CT node with a new plan found in the child CT node if the new plan is better. Specifically, when expanding a CT node N and generating a child CT node N', BP checks if the child CT node N' has the same cost as the parent CT node N and the number of conflicts in N' is less than the number of conflicts in N. If so, then BP replaces N with N' and discards all children CT nodes. Otherwise, N is split as before. Bypassing conflicts without splitting to children nodes produces smaller CTs and results in a reduced runtime.

2.4. Bounded-suboptimal variants

Finding an optimal solution for MAPF remains difficult for instances with a large number of agents, even with these modern improvements. CBS still cannot find a solution for hundreds or thousands of agents within reasonable time or memory limits. If a bounded-suboptimal solution is allowed, then large-scale MAPF instances can be solved. Some of the recent variants of CBS that can find a bounded-suboptimal solution much faster than CBS are discussed below.

2.4.1. Enhanced CBS (ECBS)

ECBS [23] replaces the A* search of CBS with focal search, both at the high-level and at the low-level CBS searches in order to compute a more informed and bounded-suboptimal plan. Focal search, or A^*_ϵ [24], is a bounded-suboptimal search algorithm that maintains two priority queues: OPEN and FOCAL. OPEN is a regular A* priority queue that sorts nodes according to their admissible cost estimate f. FOCAL contains a subset of OPEN nodes whose f values are within a user-specified bound from the current best cost estimate, i.e., FOCAL= $\{n \in \text{OPEN} : f(n) \leq w \cdot f(best_f)\}$, where $best_f$ is the top node in OPEN with minimum f-value and $w > 1$ is a user-specified suboptimality factor. FOCAL sorts the nodes with a different heuristic d, which may be inadmissible, but potentially more informative.

Focal search always expands a node from FOCAL with the minimum d value in FOCAL, which removes this node from both FOCAL and OPEN. The neighbors of the expanded node are inserted to OPEN. They are also inserted to FOCAL if their f values are within the suboptimality bound, i.e., if $f \leq w \cdot f(best_f)$. This search propagation continues until a goal node is expanded. Since the minimum f value in OPEN is always a lower bound on the optimal solution cost, any node in FOCAL does not overestimate the optimal solution cost f^* by more than w. Hence, when focal search returns a solution, the cost is at most $w \cdot f^*$.

ECBS utilizes focal search with the same suboptimality factor w at both the low and the high levels of CBS. At the low level, each agent uses a focal search with heuristic d that counts the number of collisions with other paths in the current CT node N. Hence, the low-level ECBS finds a path satisfying the constraints in N with the minimum number of collisions with other agents, whose cost is no higher than the optimal solution cost by w. The low-level ECBS also returns the minimum $f^i_{\min}$ value in OPEN for agent a_i, which is a lower bound on the cost of the shortest path for a_i. The $f^i_{\min}$ value is used to obtain a lower bound at the high level ECBS search. At the high level, ECBS prioritizes the CT nodes in the OPEN list according to their $lb(N) = \sum_{i=1}^{m} f^i_{\min}(N)$ values. The high-level FOCAL list contains a subset of the nodes in OPEN, whose $cost(N) \leq w \cdot lb(N)$, sorted according to the number of conflicts in N, or $h_c(N)$. Since the minimum lb value in OPEN is a lower bound on the optimal solution cost $cost^*$, FOCAL results in a solution whose cost is at most $w \cdot cost^*$.

2.4.2. Explicit Estimation CBS (EECBS)

EECBS [25] improves upon ECBS by addressing the drawbacks of the focal search in the high-level search of ECBS [25]. The number of conflicts in a CT node N, $h_c(N)$ is negatively correlated to the cost of the node $cost(N)$, because when a conflict is resolved in the child node, the cost of the child CT node will likely increase. Hence, when the focal search chooses a CT node to expand, then the children nodes will not likely be put in the FOCAL list because of their high costs, despite the smaller number of conflicts. Instead, a neighboring CT node from the FOCAL list with approximately the same number of conflicts will be chosen. Consequently, the lower bound rarely increases before all nodes in FOCAL are exhausted. Hence, if the solution is not within the initial suboptimal bound, ECBS may experience difficulty finding a bounded solution [25].

Explicit Estimation Search (EES) [26] addresses this negative correlation issue of the focal search by introducing a third priority queue. The additional priority queue considers the potential cost increase to remove the negative correlation between OPEN and FOCAL. Formally, EES maintains three lists: CLEANUP, OPEN, and FOCAL. CLEANUP is a regular A* priority queue that sorts its elements according to an admissible heuristic cost estimate f. OPEN is another priority queue that sorts its elements according to a more informed and, possibly inadmissible, heuristic cost estimate $\hat{f}$. FOCAL contains a subset of nodes in OPEN that does not overestimate the minimum $\hat{f}$ value in OPEN by more than a suboptimality factor w and sorts them according to another potentially inadmissible heuristic d. EES expands a node $best_d$ from FOCAL only if $f(best_d) \leq w \cdot f(best_f)$. If not, then EES selects $best_{\hat{f}}$ from OPEN and expands it only if $f(best_{\hat{f}}) \leq w \cdot f(best_f)$; otherwise, EES expands $best_f$ from CLEANUP.

EECBS replaces the high-level focal search with EES. EECBS uses an admissible heuristic $lb = \sum_{i=1}^{m} f^i_{\min}$ to sort the CT nodes in CLEANUP, and uses an inadmissible heuristic $\hat{f}(N) = cost(N) + \hat{h}(N)$ to sort a CT node N in OPEN, where $\hat{h}(N)$ is some approximated cost increment to resolve the conflicts in N. EECBS uses an additional inadmissible heuristic h_c (the number of conflicts) to sort CT nodes in FOCAL, which contains a subset of OPEN whose $\hat{f}(N) \leq w \cdot \hat{f}(best_{\hat{f}})$. EECBS first tries to expand a node $best_{h_c}$ from FOCAL with minimum h_c value if $cost(best_{h_c}) \leq w \cdot lb(best_{lb})$. If not, then EECBS tries to expand a node $best_{\hat{f}}$ from OPEN with minimum $\hat{f}$ value if $cost(best_{\hat{f}}) \leq w \cdot lb(best_{lb})$. Otherwise, EECBS expands $best_{lb}$ from CLEANUP, increasing the lower bound.

Table 1
Comparison with existing related work.

	Low-level search (individual path search)	High-level search (joint plan search)	Total Number of Queues (low-level + high-level)
CBS [15]	A*	A*	1+1 = 2
ECBS [23]	Focal Search	Focal Search	2+2 = 4
EECBS [25]	Focal Search	EES	2+3 = 5
CBSB (Our Contribution)	**bCOA***	**Modified Focal Search**	**1+2 = 3**

The bounded-suboptimality of ECBS/EECBS comes from the fact that a chosen CT node is always bounded above by $w \cdot lb(best_{lb})$, where $lb(best_{lb})$ is the lower bound of the optimal solution cost. However, the $lb(best_{lb})$ value is not always a tight lower bound of the optimal solution cost, since the value is computed based on the sum $\sum_{i=1}^{m} f^i_{\min}$, where $f^i_{\min}$ is the lowest f-value in OPEN of the low-level search of agent a_i. When the low-level focal search expands a goal node from FOCAL and returns a path, $f^i_{\min}$ may not necessarily be the shortest path length of a_i. Such a gap between $lb(best_{lb})$ and the true optimal solution cost prohibits the high-level search of ECBS/EECBS from exploring heuristic plans having a lower number of conflicts, i.e., h_c, even though those paths may, in fact, be near optimal. At the same time, tracking $f^i_{\min}$ during the search has a notable computational overhead as two priority queues are needed. To this end, this paper investigates a novel and simple way to find a bounded solution faster using a bounded-cost search framework. The prior related work and our contributions are summarized in Table 1.

3. CBS-Budget (CBSB)

In this section, we present CBS-Budget (CBSB), a novel variant of CBS, which utilizes a bounded-cost search (BCS) algorithm for the low-level search and a modified focal search for the high-level search. We will first provide a detailed description of the low-level search of CBSB and then explain how the CBSB algorithm utilizes the results of the low-level search to facilitate its high-level search.

At the low-level search, we fix a cost upper bound B to find a conflict-minimal path faster, instead of spending computational resources approximating lower bounds incrementally. Adapting to such bounded-cost search (BCS) framework [35,36] from the previously discussed bounded-suboptimal search (BSS) framework (i.e., Focal Search, EES) has two notable advantages. First, a BCS algorithm can find a solution much faster using an inadmissible heuristic without estimating the true lower bounds incrementally, as the task is to find a solution within a given cost bound as fast as possible. Second, a BCS algorithm is much simpler to implement than BSS-based algorithms, since BCS does not need additional priority queues to approximate the lower bound. BCS has one fundamental drawback, however; that is, it cannot find a solution for an arbitrarily small cost bound if no solution with a cost smaller than the bound exists [35]. A bounded-cost search problem is the problem of finding a path with cost less than or equal to the bound, and the cost of the optimal solution or the amount of suboptimality is of no interest. Consequently, bounded-cost algorithms return failure if no solution cost is less than or equal to the bound. Thus, if the bound B is less than the optimal solution cost, the algorithm will fail to find a solution.

In this work, we leverage the colored planning framework [28–30] to mitigate this drawback of BCS. To this end, we propose the budgeted COA* (bCOA*) algorithm that finds, for each individual agent, the least-cost path with the minimal number of conflicts whose cost is still bounded from above either by a given budget, or else, it finds the least-cost path if no such path exists.

Note that bCOA* shares similarities with Focal-A* search [37], which switches from focal search to standard A* (starting from scratch) to find the least-cost path—rather than a conflict-minimal path—once a user-defined number of nodes have been expanded. In contrast, bCOA* leverages the colored planning framework [28] to gracefully recover the least-cost path when no valid path exists within the specified budget. It builds upon our previous Class-Ordered A* (COA*) algorithm [29].

A key distinction between bCOA* and standard A* is that bCOA* employs an edge queue instead of a vertex queue. This prioritizes candidate paths constrained to heuristically evaluated edges, improving the computational efficiency by deferring edge evaluation—a common bottleneck in motion planning tasks such as collision checking, solving two-point boundary value problems, and detecting inter-agent conflicts.

We now present the formal definition and analysis of bCOA* below.

3.1. Weighted colored graph

Let $G = (V, E)$ be an undirected graph with vertices V and edges E. Let $\phi_V : V \to \mathcal{K}$ be a function that classifies each vertex $v \in V$ according to a color $k \in \mathcal{K} = \{1, ..., K\}$, such that ϕ_V partitions V, that is, $V = \bigcup_{k\in\mathcal{K}} V_k$ where each $V_k = \{v \in V : \phi_V(v) = k\}$ and $V_i \cap V_j = \varnothing$ for $i \neq j$. Similarly, define $\phi_E : E \to \mathcal{K}$ to be a function that classifies each edge $e \in E$ according to a color $k \in \mathcal{K}$, and $E = \bigcup E_k$ where each $E_k = \{e \in E : \phi_E(e) = k\}$ and $E_i \cap E_j = \varnothing$ for $i \neq j$. Also, for each edge $e \in E$, a weight function $c : E \to \mathbb{R}_+$ assigns a non-negative real number, e.g., distance. We will assume that the functions ϕ_E and c are given. For simplicity, we will only focus on a specific $\phi_E : E \to \{1, 2\}$, where $\phi_E(e) = 1$ if an edge $e = (u, v)$ is free of conflict (i.e., no edge conflict and no vertex conflict at both u and v) and $\phi_E(e) = 2$, otherwise.

3.2. Path

Define a path $\pi = (v_1, v_2, \dots, v_m)$ on the graph $G = (V, E)$ as a sequence of adjacent vertices $v_i \in V$, $i = 1, \dots, m$ such that, for any two consecutive vertices v_i, v_{i+1}, there exists an edge $e = (v_i, v_{i+1}) \in E$. Throughout this paper, we will interchangeably denote a path as the sequence of such edges.[3] Let $v_s, v_g \in V$ be the start and goal vertices, respectively. Denote by $\Pi(v_s, v)$, or $\Pi(v)$ for short, if there is no danger for ambiguity, the set of all acyclic paths connecting v_s and some vertex v in G. Given a value $B > 0$, we define a function $\phi_B : \Pi(v) \to \{1, 2\}$ that classifies a path $\pi \in \Pi(v)$ to a class $k \in \{1, 2\}$ as follows. If the sum of edge weights in π does not exceed B and π does not contain any color-2 edges, then $\phi_B(\pi) = 1$, otherwise $\phi_B(\pi) = 2$. That is,

$$\phi_B(\pi) = \begin{cases} 1 & \text{if } c(\pi) \le B \text{ and } c(\pi, 2) = 0, \\ 2 & \text{otherwise,} \end{cases} \tag{1}$$

where, with a slight abuse of notation, $c(\pi) = \sum_{e \in \pi} c(e)$ is the sum of the edge weights in π and $c(\pi, k) = \sum_{e \in \pi : \phi_E(e) = k} c(e)$ is the sum of the edge weights that are of color-k in path π. We define a binary relation between two paths according to their class, breaking ties, first by their $c(\pi)$-value, and then by their $c(\pi, 2)$-value. Formally,

$$\pi_1 \prec \pi_2 \iff \begin{cases} \phi_B(\pi_1) < \phi_B(\pi_2) & \\ c(\pi_1) < c(\pi_2) & \phi_B(\pi_1) = \phi_B(\pi_2) \wedge c(\pi_1) \neq c(\pi_2), \\ c(\pi_1, 2) \le c(\pi_2, 2) & \text{otherwise.} \end{cases} \tag{2}$$

3.3. Search tree

In the standard A* algorithm, each node in the search tree is associated with a vertex of the graph, along with the cost-to-come, the estimated cost-to-go, and the back-pointer which together represent the optimal path constraint to the vertex found so far. This node representation is used to prioritize the search for the best candidate solution. bCOA* generalizes this node-path association by linking a node to an edge instead of a vertex, where the node represents the optimal path constraint to the edge in the graph. Similarly, the priority queue in bCOA* prioritizes nodes (representing paths constrained to edges) as the standard A* prioritizes nodes constrained to vertices, but with a generalized ordering.

To prioritize nodes using the generalized ordering, we store the cost of a path as a vector in $\mathbb{R}^2$, denoted by $\theta(\pi) = (c(\pi, k))_{k \in \{1,2\}} \in \mathbb{R}^2$, where the k-th index stores the sum of edge weights of color k in the path π. Notably, the total order defined over two paths π_1 and π_2 from the same start vertex to the same end vertex induces a total order on the vector values, that is, $\theta(\pi_1) \prec \theta(\pi_2)$ if $\pi_1 \prec \pi_2$. The other direction is also true. That is, $\pi_1 \prec \pi_2$ if $\theta(\pi_1) \prec \theta(\pi_2)$ for any π_1, π_2 with the same start and end vertices. More precisely, given $\theta(\pi)$, we first compute $c(\pi) = \theta_1(\pi) + \theta_2(\pi)$ and $c(\pi, 2) = \theta_2(\pi)$. Using these, we determine $\phi_B(\pi)$ according to Equation (1). Subsequently, we define $\theta(\pi_1) \prec \theta(\pi_2)$ as specified in Equation (2). We also define an admissible estimate of $\theta(\pi)$ with $\widehat{\theta}(\pi)$, such that $\widehat{\theta} \preceq \theta$. A trivial heuristic estimate of $\widehat{\theta}$ is the zero vector.

At the start of the search, none of the edges in G are evaluated. As the search progresses, an explicit search tree T rooted at v_s is built by adding evaluated edges to it. Each node in the search tree corresponds to a path from v_s to a specific edge (u, v), and it stores the cost-to-come value accumulated along the current best path in the tree, denoted by $g(u)$, and a back-pointer to its parent node, denoted by $bp(u)$. Note that $g(u) = \theta(\pi(u))$, where $\pi(u)$ is the path from v_s to u found so far in the current search tree.

In the standard A* algorithm, each node in the search tree T stores a back-pointer to its parent node and the cost-to-come value obtained by traversing the back-pointed path. While the cost-to-come value can be retrieved by following the back-pointed path in the forward direction, the numerical value is explicitly stored at each node to speed up the ordering of expanding leaf nodes, preserving the completeness and optimality of the algorithm with respect to the cost-to-come. This approach can be extended to different total ordering of paths without sacrificing the correctness of the algorithm. In fact, the completeness and optimality of such generalization with respect to any ordered set of paths can be formally demonstrated, as we will show in Section 4.

3.4. Priority queue

We will use an edge queue Q to prioritize the expansion (i.e., evaluation) of edges based on an admissible estimate of the total cost constrained to an edge, breaking ties using an admissible cost-to-come constrained to this edge. The key $k(e)$ of an edge $e = (u, v)$ contains two components: $k(e) = [k_1(e); k_2(e)]$, where $k_1(e) = g(u) + \widehat{\theta}(u, v) + \widehat{\theta}(v, v_g)$, $k_2(e) = g(u) + \widehat{\theta}(u, v)$, where

$$\theta(u, v) = (c(e, k))_{k \in \{1,2\}}, \qquad \widehat{\theta}(u, v) = (\widehat{c}(e, k))_{k \in \{1,2\}}. \tag{3}$$

Here, $c(e, k)$ equals $c(e)$ if $\phi_E(e) = k$ and is 0 otherwise. Similarly, $\widehat{c}(e, k)$ equals $\widehat{c}(e)$ if $\widehat{\phi}_E(e) = k$ and is 0 otherwise, where $\widehat{c}(e)$ represents an admissible heuristic weight that does not overestimate the actual weight $c(e)$, while $\widehat{\phi}_E(e)$ represents an admissible heuristic color that does not overestimate the actual color $\phi_E(e)$, i.e., $\widehat{c} \le c$ and $\widehat{\phi}_E \le \phi_E$ for all $e \in E$. A trivial $\widehat{c}$ is 0 and a trivial $\widehat{\phi}_E$ is 1. Note that $\widehat{\theta} \preceq \theta$ is an admissible estimate of path cost, and we can compute $\widehat{\theta}(u, v)$ using an admissible heuristic weight $\widehat{c}$

[3] Note that for a time expanded graph (TEG), each vertex is associated with a unique pair of location and time, and each edge is associated with a unique pair of vertices.

and an admissible heuristic color $\widehat{\phi}_E$ of an edge $e=(u,v)$. Likewise, we can also compute $\widehat{\theta}(v,v_g)$, the cost of an admissible heuristic path from v to the goal, using an admissible heuristic weight (e.g., distance to the goal) and an admissible heuristic color. Edges are sorted by their key values in lexicographical order, that is, $k(e) \prec k(e')$ if and only if either $k_1(e) \prec k_1(e')$ or $(k_1(e)=k_1(e')$ and $k_2(e) \prec k_2(e'))$. These heuristic estimates of the path cost delay the actual evaluation of the edge, as the priority queue prioritizes edges based on their heuristic estimates. Note that in many motion planning applications, evaluating the weight and color of edges can be often more expensive than searching (i.e., expanding the node), since the edge evaluation often involves collision checking, solving two-point boundary value problems, detecting inter-agent conflicts, etc. Hence, using a heuristic weight and color of an edge can defer the expensive evaluation until it becomes necessary to find the optimal solution [38–40].

3.5. Algorithm

Algorithm 1 presents the pseudo code for bCOA*. The algorithm initiates the search by creating a search tree T with the start vertex as the root and inserting promising outgoing edges into the priority queue Q (Line 16). An edge is considered promising if it has the potential to improve the current search tree T with its heuristic estimates. Among the chosen edges, only those that can potentially improve the current path to the child vertex with the heuristic estimates are selected for insertion into the queue.

At each iteration, bCOA* removes the edge with the highest priority from the queue (Line 18) and evaluates it. Evaluating an edge e returns the actual $\theta(e)$. If the evaluation of the edge improves the child vertex's g-value, the edge is added to the tree T. The outgoing edges of the newly added leaf vertex in T are then inserted into the queue according to their priority. The algorithm continues to expand the search tree until the goal vertex is reached. The resulting search tree consists of connected classified edges rooted at v_s, and it contains an optimal path from v_s to v_g. The solution path is traced back from v_g to its back-pointer up to v_s using the bp function.

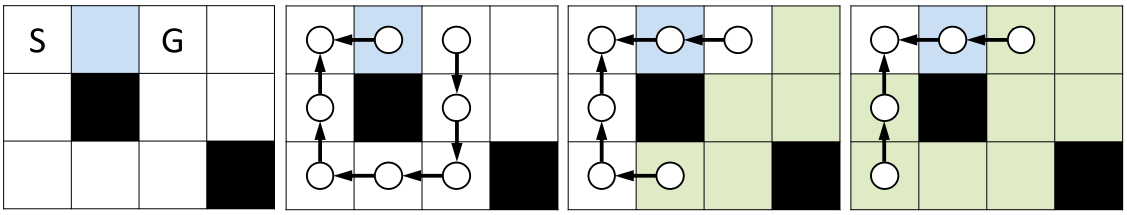
Algorithm 1 Budgeted Class Ordered A*(v_s, v_g, G).

```
1:  procedure EnqueueOutgoingEdges(v)
2:      for all w ∈ succ(v) do
3:          if g(v) + θ̂(v,w) + θ̂(w,v_g) ⪯ g(v_g) then
4:              if g(v) + θ̂(v,w) ≺ g(w) then
5:                  Q.PushEdge((v,w))
6:  procedure RewireTree(u,v)
7:      bp(v) = u
8:      g(v) = g(u) + θ(u,v)
9:      T.Insert(u,v)
10: procedure Main()
11:     T.Insert(v_s, v_s)
12:     Q ← ∅
13:     bp(v_s) = null
14:     g(v_s) = (0,0)
15:     for all v ∈ G do g(v) = (∞,∞)
16:     EnqueueOutgoingEdges(v_s)
17:     while true do
18:         (u,v) ← Q.PopFrontEdge()
19:         if u = v_g then
20:             π ← (u)
21:             while u ≠ null do
22:                 π ← (bp(u), π)
23:                 u ← bp(u)
                return π;
24:         if g(u) + θ(u,v) ≺ g(v) then
25:             RewireTree((u,v))
26:             EnqueueOutgoingEdges(v)
```

3.6. Example of budgeted COA* (bCOA*)

To illustrate the basic idea behind bCOA*, consider, for example, the 2D grid world in Fig. 1a, where an agent tries to find a path from S to G. The black cells are obstacles where no passing paths exist, and the blue cell indicates another agent sitting at that location, so a conflict will occur when crossing that cell. Any path crossing the blue cell will be considered as being *class*-2. For simplicity of exposition, assume $h=0$ for all cells, and each move takes a unit cost. bCOA* with an infinite budget (Fig. 1b) builds a search tree from S to G and finds the least-cost path that has no conflict. Now suppose that bCOA* is given a budget $B=2$ (Fig. 1c). In this case, bCOA* considers the paths that are more than 2 steps away from S as *class*-2, that is, any path passing through the green cells will be considered as *class*-2. As a result, bCOA* with $B=2$ builds a different search tree, and the resulting path goes through the blue cell, because that path is the least-cost path with minimal inclusion of *class*-2 paths. Likewise, when bCOA* is given zero budget, then bCOA* finds the least-cost path from S to G as no *class*-1 paths exist.

Given a budget B, bCOA* returns a collision-free solution with a cost of at most B, if such a path exists. Otherwise, it provides the least-cost path, breaking ties by the number of conflicts. Hence the solution returned by bCOA* with a budget B is bounded above



(a) A grid world (b) $B = \infty$ (c) B=2 (d) B=0

Fig. 1. A 2D grid environment with three bCOA* search trees with different budget values. A path can be retrieved by following the arrows (parent nodes). The blue cell indicates a static agent sitting at that location resulting in a conflict, and the green cells indicate the cells that are farther than a given budget steps away from the start. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

by $\max(B, f^*)$, where f^* is the least-cost path length in the graph. Before we prove the properties of bCOA* in Section 4, we first investigate how these properties of bCOA* benefit CBS.

3.7. Details of the CBSB algorithm

The proposed algorithm, CBS-Budget (CBSB), uses the budgeted COA* (bCOA*) on a time-expanded graph (TEG) at the low-level search of CBS and uses a modified focal search at the high-level search of CBS, in order to expand a bounded-suboptimal heuristic plan with the minimum number of conflicts. The modified high-level focal search of CBSB is different than the high-level focal search of ECBS. ECBS uses a lower bound, namely, the lb-value to approximate the suboptimal upper bound, i.e., $w \cdot lb(best_{lb}) \leq w \cdot cost^*$, where $best_{lb}$ is the top CT node of OPEN of the ECBS high-level focal search. On the other hand, CBSB uses a different suboptimal upper bound, called b-value, which never over-approximates the true optimal solution cost by more than w, i.e., $b(best_b) \leq w \cdot cost^*$, where $best_b$ is the top node CT of OPEN of the CBSB high-level focal search.

To achieve the more informative suboptimal upper bound $b(best_b)$, each CT node N of CBSB maintains a list $\{b_i(N)\}_{i=1}^m$, where $b_i(N)$ is a budget given for agent a_i to find a path using bCOA*. The b-value of N is simply the sum of these individual budgets, i.e., $b(N) = \sum_{i=1}^m b_i(N)$. The root CT node of CBSB is initially assigned $b_i = w \cdot \hat{f}_i$, where $\hat{f}_i$ is an admissible heuristic cost estimate (e.g., the least-cost path cost) and w is a user-specified suboptimality factor. Therefore, b_i does not over-approximate the least-cost path of agent a_i with cost f_i^* by more than w, i.e., $b_i = w \cdot \hat{f}_i \leq w \cdot f_i^*$. These b_i values are incrementally updated based on the result of the low-level search of CBSB, which are then used for the low-level search at the children CT nodes.

At the low level, using the value $b_i(N)$, bCOA* computes the least-cost path with minimal inclusion of conflicts that is at most b_i-long for agent a_i, while satisfying the constraints in N, if one exists; otherwise, bCOA* finds the least-cost path that satisfies the constraints in N. Suppose bCOA* finds a path for agent a_i with length f_i. If $b_i < f_i$, then $f_i = f_i^*$, which we will prove in Section 4.2. Hence, if $b_i < f_i$ after the low-level search of agent a_i, CBSB updates b_i for the child CT node with $w \cdot f_i$, so that b_i becomes more informed and yet still never over-approximates the least-cost path satisfying the current constraints by more than w, i.e., $b_i \leq w \cdot f_i^*$. Then $b = \sum_i^m b_i$ for the current CT node with updated b_i's is computed, which never overestimates the optimal cost subject to the constraints in that CT node.

At the high level, CBSB maintains two priority queues: OPEN and FOCAL. OPEN is a regular A* open list that sorts CT nodes according to their b values. The minimum b value of OPEN, or $b(best_b)$ never overestimates the true optimal solution cost by more than w factor. FOCAL contains a subset of OPEN whose cost estimate does not exceed the minimum b of OPEN and sorts according to the number of conflicts, i.e., FOCAL= $\{n \in$ OPEN $: cost(N) \leq b(best_b)\}$. In our implementation, we used zero heuristic, i.e., $h(N) = 0$. CBSB always chooses a node from FOCAL with the minimum number of conflicts. Hence, when CBSB expands a goal node N, then the cost of the node does not exceed the optimal solution cost by more than a w factor, i.e., $cost(N) \leq b(best_b) \leq w \cdot cost^*$. Note that in case $w = 1$, the b-value of each CT node is equal to the $cost$-value since $b_i = f_i^*$ for all agents, so the FOCAL queue of CBSB always expands the same node as head (OPEN), and CBSB reduces to CBS.

The pseudo-code of the high-level CBSB is provided in Algorithm 2. CBSB is similar to CBS in the way it chooses a conflict and imposes constraints on the two children CT nodes (Algorithm 2 Line 41-42). Line 44 GENERATECHILD finds a path for a single agent with newly imposed constraint. CBSB is different from CBS in the way a CT node is prioritized for expansion. When a new CT node is generated, CBSB pushes it to OPEN and also to FOCAL if the cost estimate is lower than the minimum b-value in OPEN. When choosing a CT node for expansion, CBSB first updates FOCAL according to the current minimum b-value in OPEN, and then selects a node from FOCAL with the minimum number of conflicts. When generating a new CT node, CBSB uses the current b_i to compute a path using bCOA*, and then updates b_i and the b-value, accordingly.

3.8. Example

Let us momentarily skip the lines in teal color for bypassing conflicts (Algorithm 2 Line 45-48) and let us describe the basics of CBSB using the example shown in Fig. 2. Fig. 2 shows an MAPF instance where the bears need to swap their positions, that is, agent 1 at A needs to reach C, and agent 2 at C needs to reach A.

We will first consider the CBSB search with a suboptimality factor of 2 (i.e., w=2). The corresponding CT is shown in Fig. 3a. The OPEN and FOCAL lists of CBSB of each iteration are shown in Fig. 3b. The search begins by generating the root node N_0, where

Algorithm 2 High-level search of CBSB-BP.

```
1: procedure PushNode(N)
2:     OPEN ← OPEN ∪ {N}
3:     if cost(N) + h(N) ≤ b(head(OPEN)) then
4:         FOCAL ← FOCAL ∪ {N}
5: procedure SelectNode()
6:     if b_min < b(head(OPEN)) then
7:         b_min ← b(head(OPEN))
8:         for all N ∈ OPEN do
9:             if cost(N) + h(N) ≤ b_min then
10:                FOCAL ← FOCAL ∪ {N}
11:    N ← head(FOCAL)
12:    FOCAL ← FOCAL \ {N}
13:    OPEN ← OPEN \ {N}
14:    return N
15: procedure GenerateRoot()
16:    R ← create a new CT node
17:    for all agent i do
18:        R.b_i ← w · f̂_i
19:        R.p_i ← FindPath(R.b_i)
20:        if R.b_i < |R.p_i| then R.b_i ← w · |R.p_i|
21:    R.b ← Σ_i^m R.b_i
22:    return R
23: procedure GenerateChild(N, constraint)
24:    N′ ← a copy of N
25:    N′.constraints ← N′.constraints ∪ {constraint}
26:    k ← constraint.agent
27:    N′.p_k ← FindPath(N′.b_k, N′.constraints)
28:    if N′.b_k < |N′.p_k| then N′.b_k ← w · |N′.p_k|
29:    N′.b ← Σ_i^m N′.b_i
30:    return N′
31: procedure Main()
32:    OPEN ← ∅
33:    FOCAL ← ∅
34:    b_min ← 0
35:    R ← GenerateRoot()
36:    PushNode(R)
37:    while OPEN is not empty do
38:        N ← SelectNode()
39:        if N.conflicts is empty then
40:            return N.paths
41:        conflicts ← ChooseConflict(N)
42:        constraints ← ResolveConflict(conflict)
43:        for each constraint in constraints do
44:            N′ ← GenerateChild(N, constraint)
45:            if cost(N′) ≤ b_min and b_i(N′) ≤ b_i(N) and h_c(N′) < h_c(N) then
46:                N.paths ← N′.paths
47:                N.conflicts ← N′.conflicts
48:                Go to Line 39
49:            PushNode(N′)
```

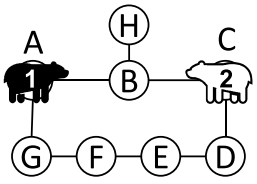
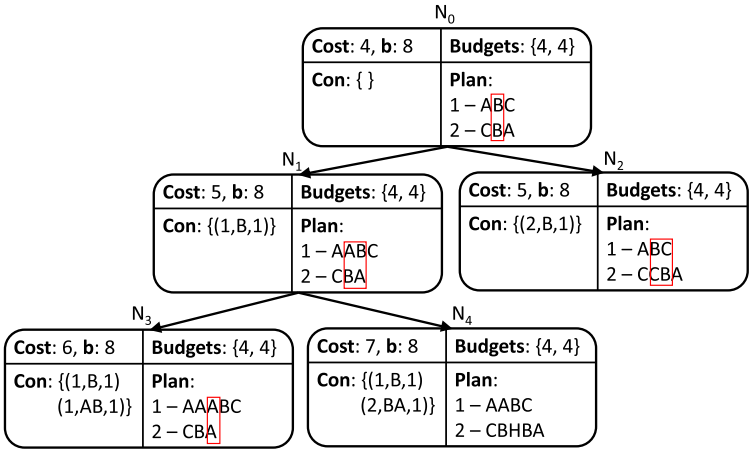


Fig. 2. An example of MAPF instance with 2 agents. The two bears must swap their positions.

each agent is given an initial budget b_i of 4 according to its admissible heuristic path length and the given suboptimality factor, i.e., $b_i = w \cdot \hat{f}_i$. bCOA* then finds the path with the given budget. Since the path found for each agent does not exceed the given budget, all the budgets stay the same after the low-level searches. The root node is inserted into OPEN and FOCAL. OPEN sorts the CT nodes according to their b-values, which is the sum of the individual budgets. FOCAL sorts the subset of the OPEN list whose *cost* does not exceed the minimum b, according to the number of conflicts h_c.

During the first iteration, CBSB chooses the top node from FOCAL, namely N_0. Since there exists a conflict at vertex B at timestep 1 in N_0, CBSB generates two children nodes to resolve the conflict. The children CT nodes N_1 and N_2 are generated by replanning for the conflicting agents with additional constraints to prevent the conflict. When an agent replans, the existing paths of other agents



(a) The constraint tree (CT) of CBSB with w=2.

Iteration 0	OPEN	(N_0, b=8, h_c=1)
	FOCAL	(N_0, b=8, h_c=1)
Iteration 1	OPEN	(N_1, b=8, h_c=1), (N_2, b=8, h_c=1)
	FOCAL	(N_1, b=8, h_c=1), (N_2, b=8, h_c=1)
Iteration 2	OPEN	(N_2, b=8, h_c=1), (N_3, b=8, h_c=1), (N_4, b=8, h_c=0)
	FOCAL	(N_4, b=8, h_c=0), (N_2, b=8, h_c=1), (N_3, b=8, h_c=1)

(b) The high-level CBSB search iterations with $w = 2$.

Fig. 3. The CBSB search with $w = 2$. The constraint set of each CT node is denoted with **Con**.

are taken into consideration such that bCOA* finds the least-cost path with minimal inclusion of conflicts or paths exceeding the given budget in length. At the CT node N_1, the agent 1 finds a new path using the budget of agent 1 at N_0, and at the CT node of N_2, the agent 2 finds a new path using the budget of agent 2 at N_0. Since these new path lengths do not exceed the initial budgets, the budgets stay the same after the searches. The generated CT nodes N_1 and N_2 are put in OPEN and FOCAL, accordingly.

At the next iteration, the CT node N_1 is chosen from FOCAL for expansion. The two children nodes N_3 and N_4 are generated, where each node contains an additional constraint to resolve the chosen conflict in N_1. The budgets stay the same after the low-level searches as before. The generated CT nodes are put in OPEN and FOCAL, accordingly. Since the cost of N_4 is less than the minimum b-value in OPEN, N_4 is put in FOCAL. Also, since N_4 has 0 conflict, the node becomes the head of FOCAL. Hence, at the next iteration N_4 is chosen and the search halts, returning the paths in N_4 as a solution. The paths in N_4 are guaranteed to be bounded suboptimal, that is, the cost does not exceed twice that of the optimal solution length. In fact, the paths in N_4 is the optimal solution to this MAPF instance.

Now let us consider solving the same problem but with $w = 1$, that is, we wish to find the optimal solution. The purpose of this example is to show that CBSB with $w = 1$ is equivalent to CBS. The corresponding CT is shown in Fig. 4. The OPEN and FOCAL lists of CBSB of each iteration are shown in Fig. 5. The search begins by generating the root node N_0, where each agent is given an initial budget $b_i = 2$ according to its admissible heuristic path length and the given suboptimality factor, i.e., $b_i = w \cdot \hat{f}_i$ and finds the path with bCOA* with the given budget. Since the path found for each agent does not exceed the given budget, all the budgets stay the same after the low-level searches at the root. The root node is inserted into OPEN and FOCAL.

During the first iteration, the CT node N_0 is chosen from FOCAL and expanded to resolve the conflict found at vertex B at timestep 1. At the CT node N_1, agent 1 finds a new path with the budget of agent 1 at N_0, and since the path returned (AABC) of length 3 is longer than the given budget of 2, the budget of agent 1 is updated to 3. Likewise, agent 2 at the CT node N_2 returns a path longer than the given budget, increasing the budget accordingly. The CT nodes N_1 and N_2 are put in OPEN and FOCAL. The CT node N_1 is chosen at the next iteration (Fig. 5 at Iteration 2). Two children nodes N_3 and N_4 are generated and put in OPEN with their updated b values. Although N_4 is conflict-free it is not put in FOCAL, since its *cost*-value exceeds the minimum b-value in OPEN. Consequently, the CT node N_2 is chosen from FOCAL for expansion at iteration 3. Likewise, when N_2 is chosen for expansion, N_5 and N_6 are generated and these children CT nodes are put in OPEN. Similar to N_4, N_5 is not put in FOCAL as the cost exceeds the minimum b-value in OPEN. The goal nodes N_4 or N_5 are not chosen until the minimum b-value increases later in the search. This is a well-studied pathological case where CBS is inefficient [15,41], that is, when there exist many infeasible plans due to conflicts whose cost is yet lower than the optimal solution. Eventually, N_4 is chosen and the search halts, confirming that the paths in N_4 is

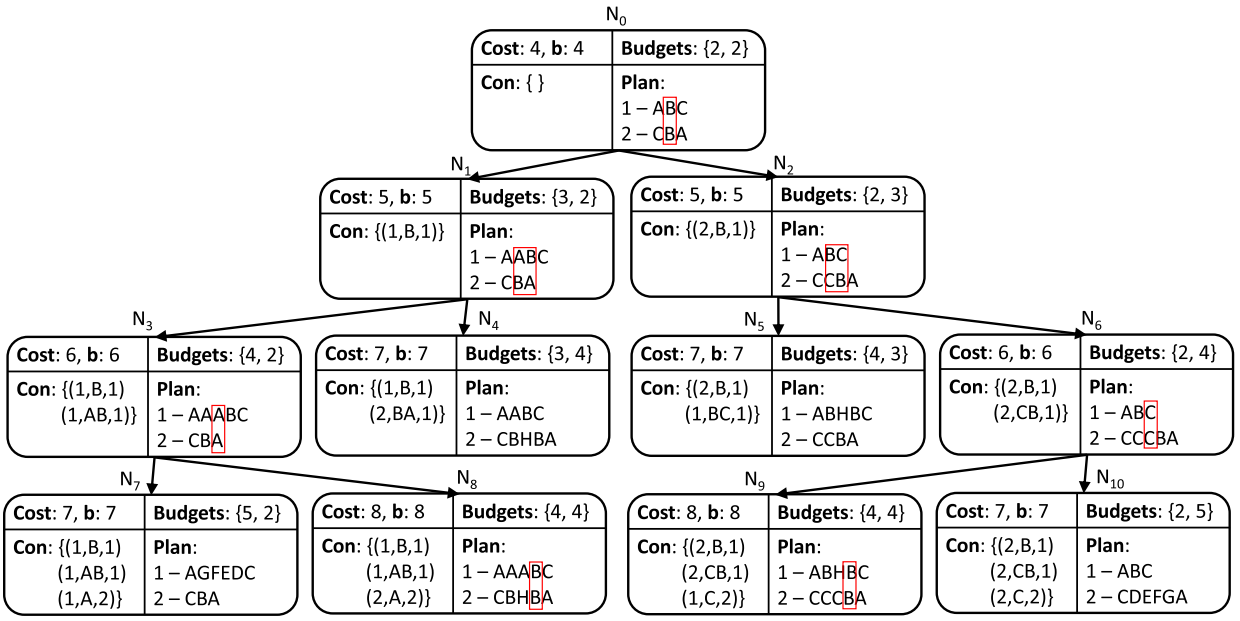


Fig. 4. The constraint tree (CT) of CBSB with $w = 1$.

Iteration 0	OPEN	(N_0, b=4, h_c=1)
	FOCAL	(N_0, b=4, h_c=1)
Iteration 1	OPEN	(N_1, b=5, h_c=1), (N_2, b=5, h_c=1)
	FOCAL	(N_1, b=5, h_c=1), (N_2, b=5, h_c=1)
Iteration 2	OPEN	(N_2, b=5, h_c=1), (N_3, b=6, h_c=1), (N_4, b=7, h_c=0)
	FOCAL	(N_2, b=5, h_c=1)
Iteration 3	OPEN	(N_3, b=6, h_c=1), (N_6, b=6, h_c=1), (N_4, b=7, h_c=0), (N_5, b=7, h_c=0)
	FOCAL	(N_3, b=6, h_c=1), (N_6, b=6, h_c=1)
Iteration 4	OPEN	(N_6, b=6, h_c=1), (N_4, b=7, h_c=0), (N_5, b=7, h_c=0), (N_7, b=7, h_c=0), (N_8, b=8, h_c=1)
	FOCAL	(N_6, b=6, h_c=1)
Iteration 5	OPEN	(N_4, b=7, h_c=0), (N_5, b=7, h_c=0), (N_7, b=7, h_c=0), (N_{10}, b=7, h_c=0), (N_8, b=8, h_c=1) , (N_9, b=8, h_c=1)
	FOCAL	(N_4, b=7, h_c=0), (N_5, b=7, h_c=0), (N_7, b=7, h_c=0), (N_{10}, b=7, h_c=0)

Fig. 5. The high level CBSB search iterations with $w = 1$.

the optimal solution. Note that CBSB with a suboptimality factor of 1 ($w = 1$) is equivalent to CBS in the way a CT node is chosen for expansion because the *cost*-value and the *b*-value of a CT node is always the same when $w = 1$.

3.9. CBSB with bypassing conflicts

The Bypassing Conflict (BP) [34] technique can be used on top of CBSB (Algorithm 2 Line 45-48) to reduce the size of the CT by avoiding the splitting of CT nodes. This improvement is denoted as CBSB-BP and is shown in teal color in Algorithm 2. Since CBSB finds a bounded suboptimal path, we can relax the conditions of accepting bypasses. Specifically, when expanding a CT node N and generating a child CT node N', CBSB-BP checks if $cost(N') < b_{\min}$ and $b_i(N') \leq b_i(N)$ and $h_c(N') < h_c(N)$, where $b_{\min}$ is the b-value of the top node of OPEN, b_i is the budget of the replanned agent i at N', and h_c is the number of conflicts. Note that $cost(N') < b_{\min}$ ensures that the plan in N' is still bounded suboptimal. The condition $b_i(N') \leq b_i(N)$ ensures that the replaced path of agent i in N is still bounded above by $b_i(N)$ and the condition $h_c(N') < h_c(N)$ avoids cycles. If these conditions hold, then CBSB-BP adopts the paths of N' and the conflicts of N' for N. Otherwise, CBSB-BP splits N as before.

4. Analysis of CBSB

In this section, we analyze the properties of CBSB. First, we examine the properties of the bCOA* algorithm, and then we analyze the completeness and bounded suboptimality of CBSB.

4.1. Properties of bCOA*

Let us first prove the completeness and optimality of bCOA* by extending the results of the original A* [42]. The key observation here is that the original A* algorithm uses an ordinary ordering of real numbers for expanding paths to find a path minimizing the cost-to-come, but the vertex ordering of the algorithm can be generalized to a total ordering of paths. In our analysis, we will assume that a total path ordering exists such that for any pair of paths having the same start and goal vertices, the order gives a binary relation between the two. Then, we will prove that the resulting path of such an algorithm is optimal with respect to the defined ordering.

We will present an edge queue version of the proof instead of the vertex queue proof of the original A*. But since the goal is to find a path from a start vertex to a goal vertex, we need to first define the trivial edges involving the start and goal vertices. To this end, we define a start edge $s = (v_s, v_s)$ for a start vertex v_s, and a goal edge $t = (v_g, v_g)$ for a goal vertex v_g, where the evaluations of these trivial edges are naturally defined as $c(s) = 0$, $c(t) = 0$, $\phi_E(s) = \phi_V(v_s)$ and $\phi_E(t) = \phi_V(v_g)$.

With some abuse of notation, we will denote the optimal path $(e, \dots, e')$ from a start edge e to an end edge e' in the graph with $\pi^*(e, e')$. Also, we will denote the path from an edge e to e' found by the bCOA* algorithm so far in the current search tree with $\pi_T(e, e')$. We will assume that there exists an ordering for any pair of paths from the same start and end edges. For example, we have $\pi^*(e, e') \preceq \pi_T(e, e')$ since any search tree built by the algorithm cannot contain a strictly better path than the optimal path. Also, we denote an underestimating heuristic path from e to e' with $\widehat{\pi}(e, e')$, such that $\widehat{\pi}(e, e') \preceq \pi^*(e, e')$ for any edges e, e'. Concatenation of paths is denoted with $\pi(e, e') \cup \pi(e', e'') = \pi(e, e'')$ for two adjacent paths. If e' and e'' are adjacent edges on the optimal path, then, $\pi^*(e, e') \cup e'' = \pi^*(e, e'')$. We will use $\pi(e, e) = \varnothing$ for any $e \in E$, and $\varnothing \preceq \pi(e, e')$ for any edges e, e'.

We are now ready to present Lemma 1, which states that the path ordering is invariant under the concatenation with the same subpath.

Lemma 1. *Let $\widehat{\pi}(e, t) \preceq \pi(e, t)$. Then $\pi(s, e) \cup \widehat{\pi}(e, t) \preceq \pi(s, e) \cup \pi(e, t)$, for any e.*

Proof. Fix an edge $e = (u, v)$, and assume that $\widehat{\pi}(e, t) \preceq \pi(e, t)$. Then, $\widehat{\theta}(u, v) + \widehat{\theta}(v, t) \preceq \theta(u, v) + \theta(v, t)$ where $\theta(u, v)$ and $\widehat{\theta}(u, v)$ are as defined in (3). Hence, $\theta(s, u) + \widehat{\theta}(u, v) + \widehat{\theta}(v, t) \prec \theta(s, u) + \theta(u, v) + \theta(v, t)$, since the ordering is invariant under the vector addition. This implies that $\pi(s, e) \cup \widehat{\pi}(e, t) \preceq \pi(s, e) \cup \pi(e, t)$, thus completing the proof. □

Next, Lemma 2 states that if an edge on an optimal path is not evaluated, then the edge queue Q must contain an edge such that an optimal subpath from the start edge to this edge is in the search tree T.

Lemma 2. *For any unevaluated edge e, and for any optimal path P from s to e, there exists an edge $e' \in Q$ on P such that $\pi^*(s, e') = \pi_T(s, e')$.*

Proof. Let $P = (s = e_0, e_1, \dots, e_n = e)$. Let $s \in Q$, that is, bCOA* has not completed the first iteration. Let $e' = s$, then $\pi^*(s, s) = \pi_T(s, s) = \varnothing$, and hence the lemma is trivially true. Now, let s be evaluated. Let Δ be the set of all evaluated edges e_i in P such that $\pi^*(s, e_i) = \pi_T(s, e_i)$ for all $e_i \in \Delta$. Then, Δ is not empty since, by assumption, $s \in \Delta$. Let e^* be the element of Δ with the highest index. Clearly, $e^* \neq e$, since e has not yet been evaluated. Let e' be the successor of e^* on P. Then, $\pi_T(s, e') \preceq \pi_T(s, e^*) \cup e'$. However, $\pi_T(s, e^*) = \pi^*(s, e^*)$ since $e^* \in \Delta$. Moreover, $\pi^*(s, e') = \pi^*(s, e^*) \cup e'$, since e^* and e' are on π^*. Therefore, $\pi_T(s, e') \preceq \pi^*(s, e')$. In general, $\pi_T(s, e') \succeq \pi^*(s, e')$, and hence $\pi_T(s, e') = \pi^*(s, e')$. Also, e' must be inserted in Q since e' is adjacent to e^*, the highest index edge in Δ. □

Corollary 3. *Suppose $\widehat{\pi}(e, t) \preceq \pi^*(e, t)$ for all e, and suppose bCOA* has not terminated. Then, for any optimal path P from s to the goal t, there exists an edge $e' \in Q$ on P with $\pi_T(s, e') \cup \widehat{\pi}(e', t) \preceq \pi^*(s, t)$.*

Proof. By the Lemma 2, there exists an edge $e' \in Q$ in P with $\pi^*(s, e') = \pi_T(s, e')$. Then, by applying Lemma 1, it follows that

$$\begin{aligned} \pi_T(s, e') \cup \widehat{\pi}(e', t) &= \pi^*(s, e') \cup \widehat{\pi}(e', t) \\ &\preceq \pi^*(s, e') \cup \pi^*(e', t) \\ &= \pi^*(s, t), \end{aligned}$$

where the last equality holds since $e' \in P$. □

We are now ready to prove the completeness and optimality properties of bCOA*.

Theorem 4. *Suppose $\widehat{\pi}(e,t) \preceq \pi^*(e,t)$ for all e. Then, bCOA* terminates in a finite number of iterations and finds the optimal path from s to the goal t, if one exists.*

Proof. We prove this theorem by contradiction. Suppose the algorithm does not terminate by finding an optimal solution. There are three cases to consider:

1. *The bCOA* algorithm terminates at a non-goal.* This contradicts the termination condition (Line 19, Algorithm 1).
2. *The bCOA* algorithm fails to terminate.* Let $\pi(s,t)$ be an optimal path from s to the goal t. Clearly, no edges with $\pi_T(s,e) \cup \widehat{\pi}(e,t) \succ \pi^*(s,t)$ will ever be evaluated, since by Corollary 3 there is some e' with $\pi_T(s,e') \cup \widehat{\pi}(e',t) \preceq \pi^*(s,t)$, and hence, bCOA* will select e' instead of e. Since there is only a finite number of acyclic paths from the start edge to any edge, all edges with $\pi^*(s,e) \cup \widehat{\pi}(e,t) \preceq \pi^*(s,t)$ can be re-evaluated at most a finite number of times. Hence, the only possibility left for bCOA* that fails to terminate is when the queue Q becomes empty before a goal is reached. Suppose, ad absurdum, that Q becomes empty before the goal is reached. If there exists a path to the goal, and the goal is not already in the tree T, then there exists at least one edge that could improve the current solution path. Then, this edge would have been inserted in the queue Q by Algorithm 1, contradicting the assumption that Q becomes empty. Hence, bCOA* must terminate.
3. *The bCOA* algorithm terminates at the goal without finding an optimal solution.* Suppose bCOA* terminates at some goal edge t with $\pi_T(s,t) \succ \pi^*(s,t)$. By Corollary 3, just before termination, there must exist an edge such that $e' \in Q$ with $\pi_T(s,e') \cup \widehat{\pi}(e',t) \preceq \pi^*(s,t)$. Hence, at this stage, e' would have been selected for evaluation rather than t, contradicting the assumption that bCOA* terminated.

Therefore, the bCOA* algorithm must terminate by finding the optimal solution, if one exists. □

4.2. Analysis of bCOA*

Given the properties of bCOA*, let us now further analyze some useful properties of bCOA* for the MAPF problem. bCOA* finds a least-cost path with minimal inclusion of *class*-2 paths and then *class*-1 paths by Theorem 4. We first examine the properties of a general class of such algorithms, denoted by Y(B), which find the least-cost path with minimal inclusion of paths either in conflict or longer than B-timesteps. Note that bCOA* is an instance of Y(B). We prove the properties for Y(B).

Lemma 5. *Y(B) finds a path having length at most B, if such a path exists.*

Proof. Let Π be the set of all paths for an agent. Let $\Pi_B \subseteq \Pi$ be the set of all paths whose length does not exceed B. By assertion, $\Pi_B \neq \emptyset$. Let $P_B \subseteq \Pi_B$ be the subset of all conflict-free paths in Π_B. Denote by $P_C = \Pi_B \cap P_B^c$, the subset of all conflicting paths in Π_B. Then, $\Pi = P_B \cup P_C \cup \Pi_B^c$, where Π_B^c is the complement of Π_B, and P_B, P_C, and Π_B^c are disjoint. If $P_B \neq \emptyset$, then Y(B) finds a path in P_B. Otherwise, Y(B) finds the least-cost path in $P_C \cup \Pi_B^c$. But P_C is not empty since $\Pi_B \neq \emptyset$ and $P_B = \emptyset$. Hence, Y(B) will find a path in P_C since any path in Π_B^c is longer in length than all paths in P_C. □

The next lemma states that if the given budget is equal to the least-cost path length, then Y(B) finds the least-cost path. Also, Y(B) finds the conflict-free least-cost path, if one exists.

Lemma 6. *If $B = f^*$, the least-cost path length in the graph, then Y(B) finds the least-cost path with minimal inclusion of conflicts.*

Proof. Let π^* be the least-cost path in the graph. If π^* is conflict-free, then Y(B) finds π^* since it is the least-cost path without conflicts. Otherwise, Y(B) finds π^* since it is the least-cost path among the set of paths either in conflict or longer than B. □

The next lemma shows that if the given budget is less than the least-cost path length, then Y(B) finds a least-cost path.

Lemma 7. *If $B < f^*$, the least-cost path length in the graph, then Y(B) finds a least-cost path with length f^*.*

Proof. Any path in the graph will be longer than B. Hence Y(B) finds the least-cost path among all paths in the graph. □

Given these lemmas, we have the following corollaries.

Corollary 8. *If $B \leq f^*$, the least-cost path length in the graph, then Y(B) finds the least-cost path.*

Proof. The result follows from Lemma 6 and Lemma 7. □

Corollary 8 gives a sufficient condition so that, given that a budget is less than or equal to the least-cost path cost, then the algorithm Y finds the least-cost path. The next corollary gives a necessary condition so that if the path length found by the algorithm

Y is shorter than the given budget, then the path is necessarily the least-cost path. This corollary is useful since the f^*-values are not known a priori.

Corollary 9. *If Y(B) finds a path of length C, where $B < C$, then $C = f^*$ the least-cost path length.*

Proof. Suppose, ad absurdum, that $f^* < C$. If $B \leq f^*$, then Y(B) finds the path of length f^* by Corollary 8, contradicting that Y(B) found a path of length $C > f^*$. So it must be that $B > f^*$. Since B is sufficiently large, by Lemma 5 Y(B) finds a path that is at most B-long. But Y(B) found a path of length $C > B$, leading to a contradiction. Hence $f^* = C$. □

Next, we examine how different budget values affect path quality.

Corollary 10. *If $B_1 < B_2$, then Y(B_1) finds a path that is not longer in length than the plan found by Y(B_2).*

Proof. Let f^* be the least-cost path length in the graph. We consider three cases:

1. $B_1 < B_2 \leq f^*$: Both Y(B_1) and Y(B_2) find the least-cost path with length f^* by Corollary 8.
2. $B_1 \leq f^* < B_2$: Y(B_1) finds the least-cost path by Corollary 8.
3. $f^* < B_1 < B_2$: Let p_1 and p_2 be the paths found by Y(B_1) and Y(B_2), respectively. Suppose, ad absurdum, that p_2 is shorter than p_1. If p_2 is conflict-free, then Y(B_1) would have found p_2 instead of p_1, since p_2 is shorter. So p_2 must be in conflict. Now, if p_1 is in conflict, then Y(B_1) would have found p_2 instead of p_1, since p_2 is shorter. So p_1 must be conflict-free. But then Y(B_2) would have found p_1 instead of p_2, since p_1 is conflict-free and shorter than B_2, contradicting the fact that Y(B_2) found p_2. Hence, Y(B_2) cannot find a shorter path than Y(B_1).

In all three cases, Y(B_1) finds a path that is no longer than a path found by Y(B_2). □

The following corollary states that the algorithm Y with a lower budget finds no fewer conflicts than when using a higher budget.

Corollary 11. *If $B_1 < B_2$, then Y(B_1) finds a path that has no fewer conflicts than Y(B_2).*

Proof. Let p_1 and p_2 be the paths found by Y(B_1) and Y(B_2), respectively. Suppose p_1 has fewer conflicts than p_2. By Corollary 10, p_1 is not longer than p_2. Hence, Y(B_2) would have found p_1 instead of p_2, a contradiction. □

Note that Y(0) is equivalent to A*, as Y(0) always finds the least-cost path regardless of the path class.

4.3. Properties of CBSB

Given the previous properties of the algorithm Y, next, we examine the properties of the main algorithm CBSB, which uses Y(b_i) (equivalently, bCOA*) with a budget b_i at the low-level search for agent a_i and a focal search at the high-level search of CBS. We first prove that for a finite $b = \sum_i^m b_i$, there is a finite number of CT nodes. Then, we prove that CBSB must finish before exhausting all the CT nodes.

Theorem 12. *For a finite b, there is a finite number of CT nodes.*

Proof. Since $b = \sum_i^m b_i$ is finite, b_i is finite for all $i = 1, \dots, m$. Let f_i^* be the least-cost path of agent a_i. Then, bCOA* finds a path for agent a_i that has length at most $M_i = \max\{b_i, f_i^*\}$. Let $M = \max_i M_i$. Then, no CT node contains a path longer than M. Hence, no conflict can occur after the M-timestep, and no constraint after the M-timestep can be generated. Since there is a finite number of such constraints (at most $M|V|m$ for vertices and $M|E|m$ for edges, where $|V|$ and $|E|$ denote the cardinalities of the vertex set and the edge set in the graph, respectively), there is also a finite number of CT nodes that contain such constraints. □

Theorem 13. *CBSB returns a solution, if one exists.*

Proof. CBSB expands all nodes whose cost does not exceed the minimum b-value. The b-value of the CT nodes is monotonically increasing and finite since b is initialized and updated with a finite number. Since for each b there is a finite number of CT nodes, a solution whose cost does not exceed b must be found after expanding a finite number of CT nodes. □

Finally, we prove that a solution returned by CBSB is bounded suboptimal. First, we show that the minimum b-value in OPEN does not overestimate the optimal solution cost by more than w, a user-specified suboptimality factor. Hence, when a goal CT node is expanded whose cost does not exceed the minimum b-value in OPEN, then the goal node will contain a bounded suboptimal solution.

Theorem 14. *A CT node in OPEN with minimum b-value does not over-approximate the optimal solution cost by more than w, that is, $b(best_b) \leq w \cdot cost^*$.*

Proof. Let $\hat{b}_i(N)$ be a budget for agent a_i at a CT node N before the low-level search and $b_i(N)$ be the updated budget for agent a_i after the low-level search. By definition, $b(N) = \sum_i^m b_i(N)$. We first prove by induction that $b(N) \leq w \cdot cost^*(N)$, where $cost^*(N)$ is the optimal solution cost satisfying the constraints in N.

At the root node R, all $\hat{b}_i$'s are initialized with $\hat{b}_i(R) = w \cdot \hat{f}_i(R)$, where $\hat{f}_i(R)$ is an admissible heuristic cost estimate of agent a_i's path, that is, $\hat{f}_i(R) \leq f_i^*(R)$, where $f_i^*(R)$ is the least-cost path satisfying the constraints which is empty in R. Suppose $\mathrm{Y}(\hat{b}_i(R))$ finds a path with length $f_i(R)$. If $f_i(R) \leq \hat{b}_i(R)$, then $b_i(R)$ is set to $\hat{b}_i(R) = w \cdot \hat{f}_i(R) \leq w \cdot f_i^*(R)$. If $f_i(R) > \hat{b}_i(R)$, then $b_i(R)$ is set to $w \cdot f_i(R) = w \cdot f_i^*(R)$, where $f_i(R) = f_i^*(R)$ holds by Corollary 9. In either case, $b_i(R) \leq w \cdot f_i^*(R)$. This holds for all a_i's, and hence $b(R) = \sum_i^m b_i(R) \leq w \cdot f_i^*(R) \leq w \cdot cost^*(R)$. Now, assume that $b_i(N) \leq w \cdot f_i^*(N)$ holds for all a_i in a CT node N. We will first show that for a child CT node N', $b_i(N') \leq w \cdot f_i^*(N')$ also holds for all agents. Note that for any i, $f_i^*(N) \leq f_i^*(N')$ holds since $N.constraints \subseteq N'.constraints$. Therefore, for each i, $\hat{b}_i(N') \leq w \cdot f_i^*(N')$ holds, since $\hat{b}_i(N')$ is set to $b_i(N) \leq w \cdot f_i^*(N) \leq w \cdot f_i^*(N')$. Suppose $\mathrm{Y}(\hat{b}_i(N'))$ finds a path with length $f_i(N')$ for agent a_i in N'. If $f_i(N') > \hat{b}_i(N')$, then $b_i(N')$ is updated with $w \cdot f_i(N')$ so that $b_i(N') = w \cdot f_i^*(N')$, for $f_i(N') = f_i^*(N')$ by Corollary 9. Hence, $b_i(N') \leq w \cdot f_i^*(N')$ for all i, and $b(N') \leq w \cdot cost^*(N')$ follows as before. Therefore, $b(N) \leq w \cdot cost^*(N)$ for any CT node N. Next, we show that at least one CT node N exists in OPEN whose $cost^*(N) \leq cost^*$, which we prove by induction.

The base case is trivial when OPEN has only the root node R, since the constraints set is empty in R and $cost^*(R) \leq cost^*$. Now assume that a CT node N whose $cost^*(N) \leq cost^*$ is chosen for expansion. Assume that there is no other CT node M in OPEN whose $cost^*(M) \leq cost^*$, otherwise we are done. If no conflict exists in N, then we are done, so let F be a conflict chosen in N. Any valid solution must put a constraint on either of two conflicting agents to resolve conflict F. Hence, there exists a child CT node N' whose $cost^*(N') \leq cost^*$, which completes the induction. But $b(N) \leq w \cdot cost^*(N)$, so the minimum b in OPEN does not overestimate the true optimal solution $cost^*$ by more than a factor of w. □

Theorem 15. *CBSB returns a solution whose cost does not exceed the optimal solution cost by more than a factor w.*

Proof. CBSB expands a CT node whose *cost* does not exceed the minimum b in OPEN. Hence, when a goal CT node is expanded from CBSB, the cost does not exceed the optimal solution cost by more than w by Theorem 14. □

5. Numerical experiments

In this section, we evaluate CBSB on the standard MAPF benchmark suite [31] using six maps of different sizes with different numbers of agents and different suboptimality factors per map. In the benchmark test, CBSB shows comparable performance to the state-of-the-art algorithms despite its simplicity. For each map, we use 25 "randomly" generated scenarios of the benchmark suite for a given number of agents, where each scenario contains different start and target locations of agents. Using the benchmark suite, we compare CBSB with other CBS variants with and without modern improvements in two subsequent parts.

5.1. Baseline comparison

This experiment aims to compare four different CBS variants: CBS, ECBS, EECBS, and CBSB, using baseline implementations. We varied the number of agents from 10 to 300 in increments of 40, and suboptimality factors from 1.02 to 1.2 in increments of 0.02, except for CBS, which finds optimal solutions.

First, we found that ECBS, EECBS, and CBSB all find near-optimal solutions regardless of the given suboptimality factor. Then, we compared the success ratio and the average runtime of CBS, ECBS, EECBS, and CBSB by fixing the suboptimality factor and by varying the number of agents. Lastly, we compared the low-level search performance of CBSB with ECBS/EECBS and concluded the first part of the experiment. All experiments had a time limit of 30 seconds and were implemented in C++. We conducted the experiments on Ubuntu 20.04 LTS with an Intel Core i5-5257U CPU and 8 GB of RAM.

Fig. 6 displays the average solution cost over 25 randomly generated scenarios with 50 agents for different suboptimality factors. We only plot the average solution cost if more than 75% of the scenarios are solved. Surprisingly, we observed no correlation between the average solution cost and the given suboptimality factor, except for one random map where ECBS and EECBS produced even better solutions with higher suboptimality factors, contradicting the intuition that a lower suboptimality factor should theoretically yield better results. Our experiments show that in all cases, ECBS, EECBS, and CBSB produced near-optimal solutions with higher suboptimality factors. Also, having a sufficiently high suboptimality factor is shown to reduce the runtime, in general. This negative correlation between the suboptimality factor and the runtime, coupled with the lack of correlation between the suboptimality factor and the solution cost, suggests that a set of individual paths with fewer conflicts tends to be a more informed (near-optimal) heuristic plan for the MAPF problems than a set of shorter paths with more conflicts.

To this end, we set the suboptimality factor to 1.2 and compared the success ratio and average runtime of CBS, ECBS, EECBS, and CBSB by varying the number of agents. Fig. 7 shows the success ratio and average runtime over 25 random scenarios for different numbers of agents in six different maps. We only plot the average runtime if more than 75% of the scenarios are solved.

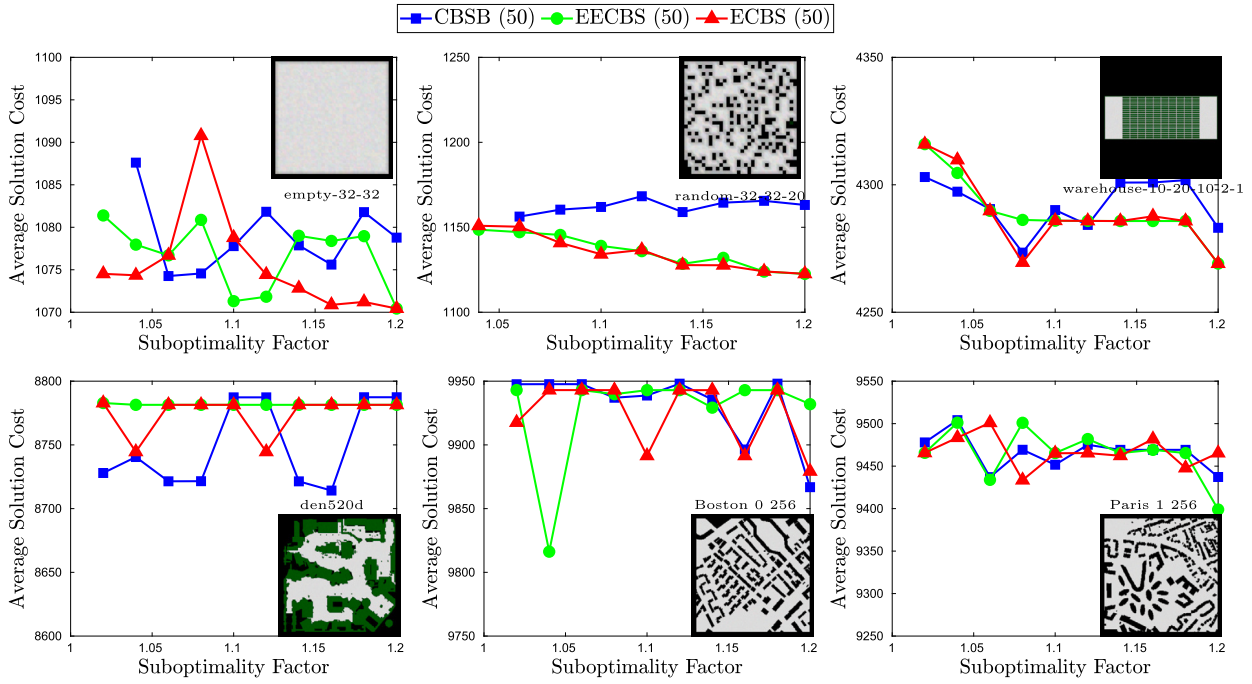


Fig. 6. Cost of solution found by ECBS, EECBS, and CBSB, averaged over 25 random scenarios with 50 agents in 6 different maps. The suboptimality factors are varied from 1.02 to 1.2 with 0.02 increment. In all the instances, ECBS, EECBS, and CBSB find a near-optimal solution regardless of its given suboptimality factor. A missing marker indicates that the corresponding algorithm could not find a solution in more than 75% of the 25 random instances given a time limit of 30 seconds.

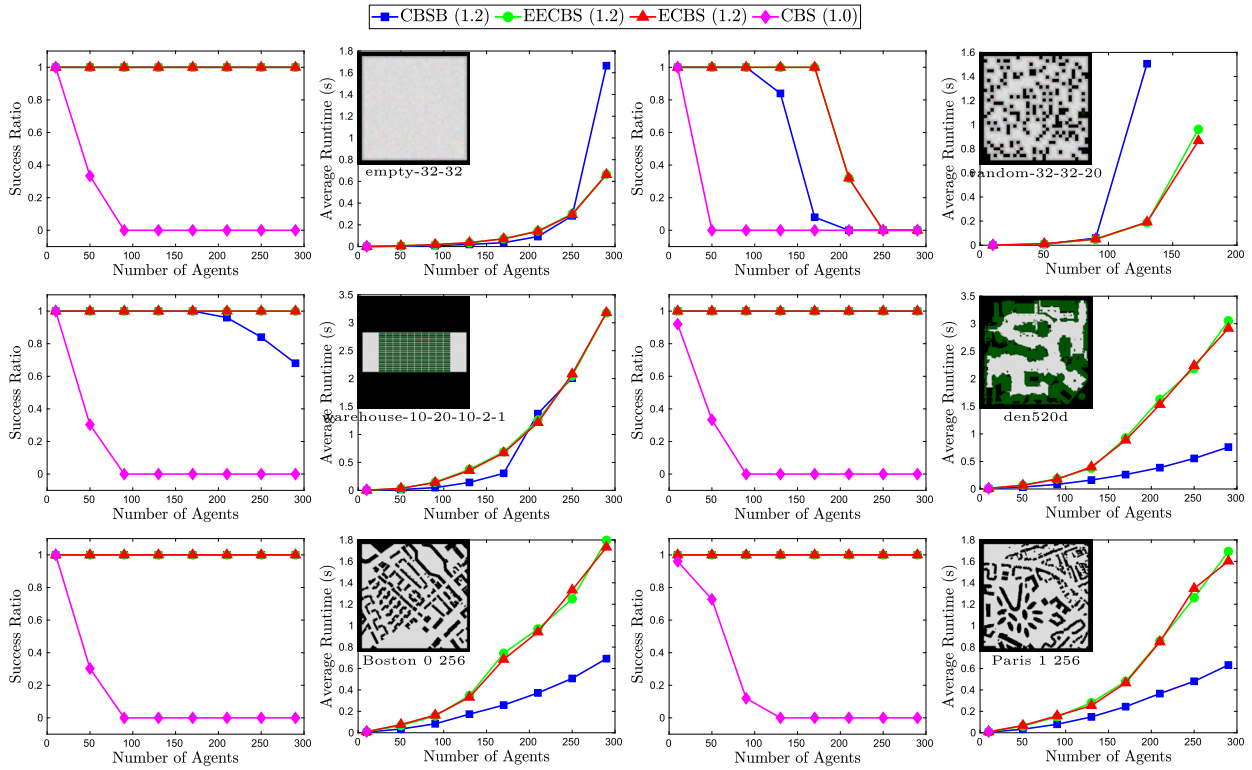


Fig. 7. Success ratio and average runtime over 25 random scenarios of CBS, ECBS, EECBS, and CBSB with different numbers of agents in 6 different maps, given a time limit of 30 seconds and a memory limit of 16 GB. All algorithms are given a suboptimality factor of 1.2 except for CBS which finds the optimal solution. Average runtime is plotted only if the corresponding algorithm found a solution in more than 75% of the 25 random scenarios.

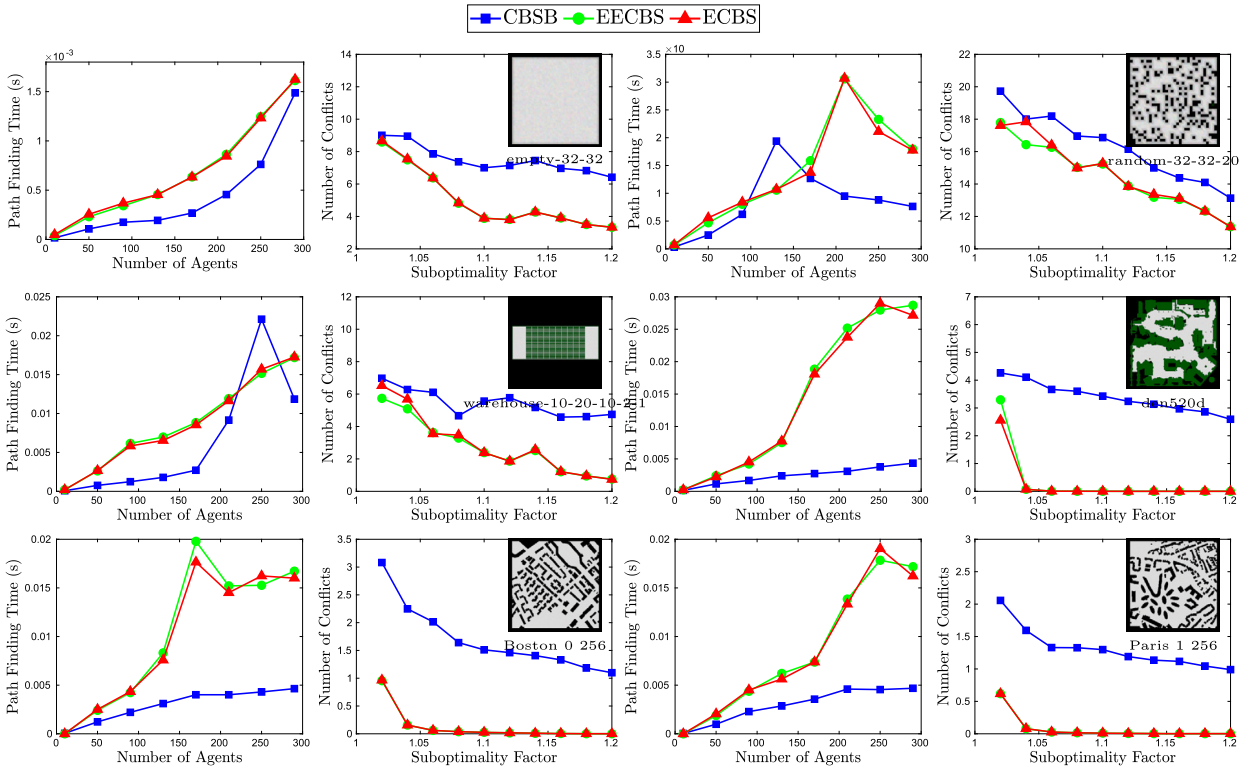


Fig. 8. Average path finding time per replanning agent and average number of conflicts in the replanned path over 25 random scenarios of CBSB, EECBS, and ECBS in 6 different maps. The number of agents is varied from 10 to 300 with 40 increments at a fixed suboptimality factor of 1.2, and the suboptimality factor is varied from 1.02 to 1.2 with 0.02 increment at a fixed 290 agents.

Our results indicate that in instances where the map is small and the number of agents is large, ECBS and EECBS outperform CBSB in terms of runtime. On the other hand, when the number of agents is relatively small or the map is large, CBSB finds a solution faster than the other algorithms. This is because CBSB's low-level search is generally faster at finding a path at the expense of more conflicts than the low-level search of ECBS/EECBS, as shown next in our comparison of the low-level search performance (see Fig. 8). Hence, ECBS/EECBS performs better when there are many conflicts to resolve in relatively small environments where the individual search space is small, while CBSB performs better when the individual search space is large with relatively fewer conflicts to resolve.

To compare the performance of the low-level search algorithms used in CBSB and ECBS/EECBS, we measured the average replanning time per agent and the average number of conflicts in replanned paths, excluding the maximum and minimum data points as shown in Fig. 8. Our results indicate that, in general, the number of conflicts per replanned path decreases with increasing suboptimality factors for all low-level searches. This is because a higher suboptimality factor allows for finding paths with fewer conflicts, at the expense of path length.

Additionally, we found that the replanning time per agent tends to increase with the number of agents, as there are more conflicts to resolve in larger scenarios. The low-level search algorithm used in CBSB, bCOA*, generally finds solutions faster than the low-level search used in ECBS/EECBS, focal search. However, focal search consistently generates paths with fewer conflicts per replanning instance than bCOA*. This is because focal search updates the focal threshold of each replanning instance during the search by expanding from OPEN, allowing for the exploration of more paths with fewer conflicts, which slows down the search. In contrast, bCOA* is given a fixed budget during the search and thus is faster. As a result, bCOA* scales better with an increasing number of agents in larger maps, whereas focal search slows down in larger maps with a large number of agents. Our implementation of CBSB can be found in https://github.com/DCSLgatech/CBSB.

5.2. Modern improvements

In the second part of the experiment, we compared the performance of CBS, ECBS, EECBS, and CBSB(-BP) using the modern implementation of CBS, ECBS, and EECBS found in https://github.com/Jiaoyang-Li/EECBS, and we denote these algorithms as CBS$^+$, ECBS$^+$, and EECBS$^+$, respectively. These algorithms are equipped with WDG heuristic [19], Prioritizing Conflicts [16], Bypassing Conflicts [34], and symmetry reasoning [17]. We fix the suboptimality factor at 1.2 and 10, except for CBS$^+$, which find the optimal solution. All the experiments in the second part were given a time limit of 10 seconds, with a memory limit of 16 GB. Note that we reduced the time limit to 10 seconds for this experiment to effectively highlight the varied success ratios of the compared

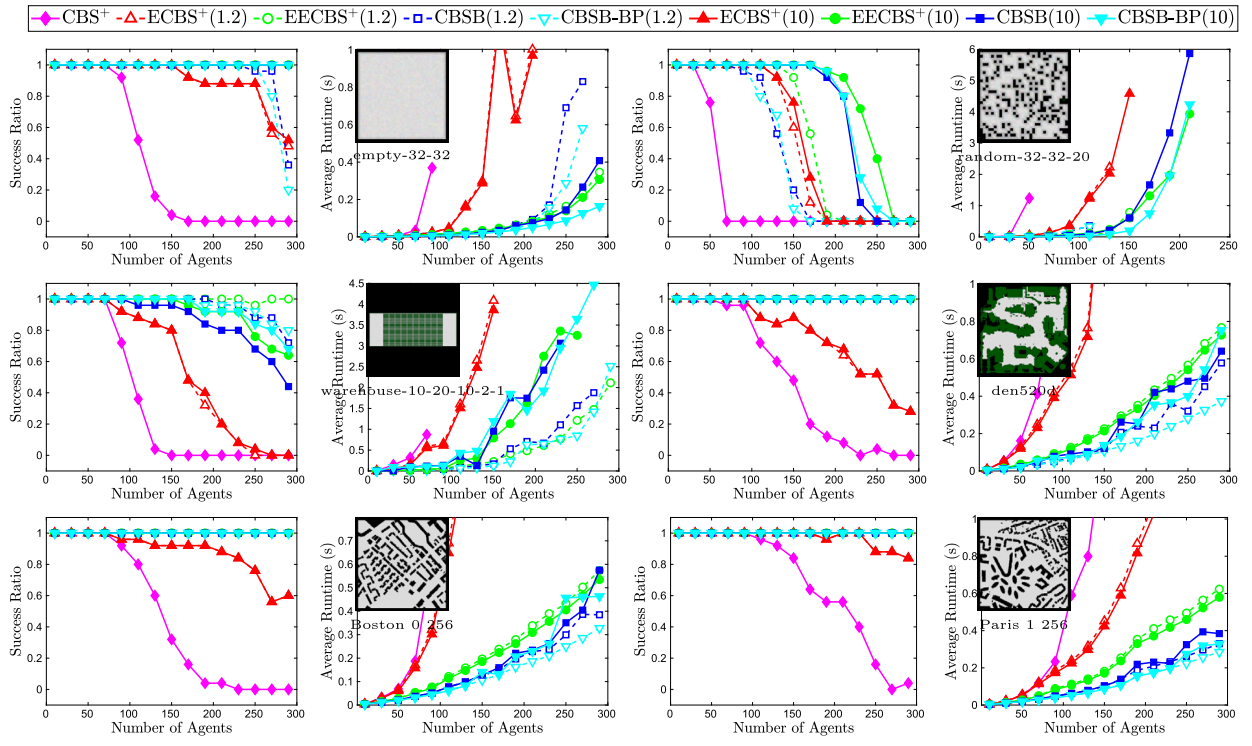


Fig. 9. Success ratio and average runtime over 25 random scenarios of CBS$^+$, ECBS$^+$, EECBS$^+$, and CBSB(-BP) with different numbers of agents in 6 different maps, given a time limit of 10 seconds and a memory limit of 16 GB. Average runtime is plotted only if the corresponding algorithm found a solution in more than 75% of the 25 random scenarios. The number inside the parenthesis is the suboptimality factor.

algorithms, especially with the additional modern improvements. The algorithms were implemented in C++ and all the experiments were conducted on Ubuntu 18.04 LTS on an Intel Core i7-8750H with 16 GB of RAM.

As shown in Fig. 9, CBSB(-BP), and EECBS$^+$ outperform the other two algorithms, both in terms of the success rate and the average runtime to find a solution. CBSB-BP always performs better than CBSB. CBSB(-BP), and EECBS$^+$ show similar performances, and there is no clear winner for all maps. CBSB(-BP) performed better in easier problem instances, where the success rates are high, whereas EECBS$^+$ performed better in harder problem instances. EECBS$^+$ performs better than CBSB(-BP) if finding a conflict minimal path is difficult due to cluttered environments. CBSB(-BP) performs better than EECBS$^+$ if conflicts are relatively easy to resolve at the expense of path length.

Note that having a very high suboptimality factor ($w = 10$) helped CBSB(-BP) to solve the problem faster in small environments (i.e., empty and random maps), whereas its performance was degraded in large environments. This is because the effective search space of bCOA* becomes larger with a larger budget. For example, suppose bCOA* is given a large budget B, and there are agents in all the neighboring locations of the start location for some time longer than B, such that all move actions will cause a conflict. Then, at the beginning of the search, bCOA* will exhaust the wait action until timestep B is reached before exploring any other move action. On the other hand, if $B = 0$, bCOA* is equivalent to A*, which explores move actions first regardless of conflicts. Hence, there is a trade-off between reducing the runtime and reducing the number of conflicts. In scenarios where avoiding conflicts without imposing constraints on the other agents is difficult, such as in the warehouse map, computing a set of paths with the minimum number of conflicts at the expense of search runtime may not be beneficial.

We omit comparisons with the many MAPF algorithms that have already been shown to perform worse than CBS [10,11] or worse than ECBS [23] or worse than EECBS [43,44]. We also omit comparisons with MAPF algorithms that have no completeness and bounded-suboptimality guarantees [45–47]. In summary, CBSB(-BP) shows state-of-the-art performance using a much simpler implementation, namely, without using the WDG heuristic [19], Prioritizing Conflicts [16], or Symmetry Reasoning [17], enhancements of CBS.

6. Conclusion and future work

We have proposed a new bounded-cost MAPF algorithm, called CBSB, which uses a budgeted COA* (bCOA*) search at the low-level and a modified focal search at the high-level. bCOA* can find an informed heuristic plan, which is still bounded above, using a single priority queue faster than focal search. We prove that, given a budget B, bCOA* returns a solution having a minimal number of conflicts, which is at most B-long, if one exists. If no solution exists that is shorter than B, then bCOA* returns the least-cost path. Based on this property of bCOA*, we also prove that CBSB is complete, and returns a bounded-suboptimal solution, if one exists.

In the benchmarking experiments, CBSB showed state-of-the-art performance despite its simple implementation. CBSB was able to compute a near-optimal solution within a fraction of a second for hundreds of agents in certain scenarios.

We conclude with several potential directions for future work. One promising avenue for future research is to investigate the impact of various budget allocation strategies in bCOA*. Our current implementation uses a uniform budget allocation strategy based on the single-agent cost suboptimality for all agents. However, it would be interesting to explore the effects of allocating more budget to certain agents or regions of the map based on their priority or likelihood of encountering conflicts, while still ensuring global suboptimality. Such strategies could potentially lead to faster convergence and improved performance in complex scenarios with many agents.

Another direction for future research is to extend the analysis of CBSB to more complex scenarios, such as environments with heterogeneous agents with different capabilities. It would be interesting to investigate how CBSB performs in these scenarios and whether modifications to the algorithm would be necessary to handle these additional complexities. Additionally, studying the behavior of CBSB on real-world robotic platforms with physical constraints, such as limited sensing and actuation capabilities, would be an important step toward practical applications of the algorithm.

CRediT authorship contribution statement

Jaein Lim: Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Panagiotis Tsiotras:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Investigation, Funding acquisition.

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.

Acknowledgement

We would like to thank Sven Koenig for introducing us to the MAPF problem and for his valuable feedback on the first version of the manuscript. We also appreciate the anonymous reviewers for their valuable insights that helped improve our theoretical results and the presentation of the numerical experiments. This work has been supported by ARL under DCIST CRA W911NF-17-2-0181 and NSF under award IIS-2008686.

Data availability

I have shared the link to my code in the manuscript.

References

[1] P.R. Wurman, R. D'Andrea, M. Mountz, Coordinating hundreds of cooperative, autonomous vehicles in warehouses, AI Mag. 29 (1) (2008) 1, https://doi.org/10.1609/aimag.v29i1.2082.
[2] A. Okoso, K. Otaki, T. Nishi, Multi-agent path finding with priority for cooperative automated valet parking, in: IEEE Intelligent Transportation Systems Conference (ITSC), 2019, pp. 2135–2140.
[3] D. Silver, Cooperative pathfinding, in: Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE'05, AAAI Press, 2005, pp. 117–122.
[4] M. Bennewitz, W. Burgard, S. Thrun, Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots, in: Ninth International Symposium on Intelligent Robotic Systems, Robot. Auton. Syst. 41 (2) (2002) 89–99, https://doi.org/10.1016/S0921-8890(02)00256-7.
[5] K. Dresner, P. Stone, A multiagent approach to autonomous intersection management, J. Artif. Intell. Res. 31 (2008) 591–656, https://doi.org/10.1613/jair.2502.
[6] F. Ho, A. Goncalves, A. Salta, M. Cavazza, R. Geraldes, H. Prendinger, Multi-Agent Path Finding for UAV Traffic Management: Robotics Track, University of Greenwich, 2019, pp. 131–139.
[7] M.R. Ryan, Exploiting subgraph structure in multi-robot path planning, J. Artif. Intell. Res. 31 (2008) 497–542, https://doi.org/10.1613/jair.2408.
[8] T. Standley, Finding optimal solutions to cooperative pathfinding problems, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI'10, AAAI Press, Atlanta, Georgia, 2010, pp. 173–178.
[9] A. Felner, R. Stern, S. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, P. Surynek, Search-based optimal solvers for the multi-agent pathfinding problem: summary and challenges, Proc. Int. Symp. Comb. Search 8 (1) (2017) 1.
[10] G. Wagner, H. Choset, M*: a complete multirobot path planning algorithm with performance bounds, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 3260–3267.
[11] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. Sturtevant, J. Schaeffer, R. Holte, Partial-expansion A* with selective node generation, Proc. AAAI Conf. Artif. Intell. 26 (1) (2012) 1.
[12] T.S. Standley, R. Korf, Complete algorithms for cooperative pathfinding problems, in: Twenty-Second International Joint Conference on Artificial Intelligence, 2011, pp. 668–673.
[13] K. Solovey, O. Salzman, D. Halperin, Finding a needle in an exponential haystack, Int. J. Robot. Res. 35 (5) (2016) 501–513, https://doi.org/10.1177/0278364915615688.
[14] R. Shome, K. Solovey, A. Dobson, D. Halperin, K.E. Bekris, dRRT*: scalable and informed asymptotically-optimal multi-robot motion planning, Auton. Robots 44 (3) (2020) 443–467, https://doi.org/10.1007/s10514-019-09832-9.
[15] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66.

[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 International Conference on Artificial Intelligence, Buenos Aires, Argentina, 2015, pp. 740–746.
[17] J. Li, G. Gange, D. Harabor, P.J. Stuckey, H. Ma, S. Koenig, New techniques for pairwise symmetry breaking in multi-agent path finding, Proc. Int. Conf. Autom. Plann. Sched. 30 (1) (2020) 193–201.
[18] H. Zhang, J. Li, P. Surynek, S. Koenig, T.K.S. Kumar, Multi-agent path finding with mutex propagation, Proc. Int. Conf. Autom. Plann. Sched. 30 (2020) 323–332.
[19] J. Li, A. Felner, E. Boyarski, H. Ma, S. Koenig, Improved heuristics for multi-agent path finding with conflict-based search, in: Proceedings of IJCAI, 2019, pp. 442–449.
[20] E. Boyarski, P. Le Bodic, D. Harabor, P. Stuckey, A. Felner, F-cardinal conflicts in conflict-based search, in: D. Harabor, M. Vallati (Eds.), Proceedings of the Twelfth International Symposium on Combinatorial Search, Association for the Advancement of Artificial Intelligence (AAAI), 2020, pp. 123–124.
[21] 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, Proc. Int. Conf. Autom. Plann. Sched. 28 (1) (2018) 83–87.
[22] J. Yu, S.M. LaValle, Structure and intractability of optimal multi-robot path planning on graphs, in: Proceedings of the AAAI Conference on Artificial Intelligence, Bellevue, WA, 2013, pp. 1443–1449.
[23] 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, pp. 19–27.
[24] J. Pearl, J.H. Kim, Studies in semi-admissible heuristics, IEEE Trans. Pattern Anal. Mach. Intell. 4 (4) (1982) 392–399, https://doi.org/10.1109/TPAMI.1982.4767270.
[25] J. Li, W. Ruml, S. Koenig, EECBS: a bounded-suboptimal search for multi-agent path finding, Proc. AAAI Conf. Artif. Intell. 35 (14) (2021) 14.
[26] J. Thayer, A. Dionne, W. Ruml, Learning inadmissible heuristics during search, Proc. Int. Conf. Autom. Plann. Sched. 21 (2011) 250–257.
[27] E. Boyarski, A. Felner, D. Harabor, P.J. Stuckey, L. Cohen, J. Li, S. Koenig, Iterative-Deepening Conflict-Based Search, vol. 4, 2020, pp. 4084–4090, ISSN: 1045-0823.
[28] D. Wooden, M. Egerstedt, On finding globally optimal paths through weighted colored graphs, in: Proceedings of the 45th IEEE Conference on Decision and Control, San Diego, CA, 2006, pp. 1948–1953.
[29] J. Lim, P. Tsiotras, A generalized A* algorithm for finding globally optimal paths in weighted colored graphs, in: IEEE International Conference on Robotics and Automation (ICRA), 2021, pp. 7503–7509.
[30] J. Lim, O. Salzman, P. Tsiotras, Class-ordered LPA*: an incremental-search algorithm for weighted colored graphs, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 6907–6913.
[31] 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: Twelfth Annual Symposium on Combinatorial Search, 2019, pp. 151–158.
[32] P. Surynek, Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems, Ann. Math. Artif. Intell. 81 (3) (2017) 329–375, https://doi.org/10.1007/s10472-017-9560-z.
[33] 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.
[34] E. Boyarski, A. Felner, G. Sharon, R. Stern, Don't split, try to work it out: bypassing conflicts in multi-agent pathfinding, Proc. Int. Conf. Autom. Plann. Sched. 25 (2015) 47–51.
[35] R. Stern, R. Puzis, A. Felner, Potential search: a bounded-cost search algorithm, Proc. Int. Conf. Autom. Plann. Sched. 21 (1) (2011) 234–241, https://doi.org/10.1609/icaps.v21i1.13455.
[36] L. Cohen, M. Greco, H. Ma, C. Hernandez, 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-18, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 1434–1441.
[37] S.-H. Chan, J. Li, G. Gange, D. Harabor, P.J. Stuckey, S. Koenig, Flex distribution for bounded-suboptimal multi-agent path finding, Proc. AAAI Conf. Artif. Intell. 36 (9) (2022) 9313–9322, https://doi.org/10.1609/aaai.v36i9.21162.
[38] A. Mandalika, O. Salzman, S.S. Srinivasa, Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges, in: Proceedings of the International Conference on Automated Planning and Scheduling, Delft, Netherlands, 2018, pp. 476–484.
[39] A. Mandalika, S. Choudhury, O. Salzman, S.S. Srinivasa, Generalized lazy search for robot motion planning: interleaving search and edge evaluation via event-based toggles, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 29, Berkeley, CA, 2019, pp. 745–753.
[40] J. Lim, M. Ghanei, R.C. Lawson, S. Srinivasa, P. Tsiotras, Lazy incremental search for efficient replanning with bounded suboptimality guarantees, Int. J. Robot. Res. 43 (8) (2024) 1175–1207, https://doi.org/10.1177/02783649241227869.
[41] J. Li, D. Harabor, P.J. Stuckey, H. Ma, G. Gange, S. Koenig, Pairwise symmetry reasoning for multi-agent path finding search, Artif. Intell. 301 (2021) 103574, https://doi.org/10.1016/j.artint.2021.103574.
[42] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107, https://doi.org/10.1109/TSSC.1968.300136.
[43] E. Lam, P.L. Bodic, New valid inequalities in branch-and-cut-and-price for multi-agent path finding, Proc. Int. Conf. Autom. Plann. Sched. 30 (2020) 184–192.
[44] P. Surynek, Lazy compilation of variants of multi-robot path planning with satisfiability modulo theory (SMT) approach, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 3282–3287.
[45] J. Li, Z. Chen, D. Harabor, P. Stuckey, S. Koenig, MAPF-LNS2: fast repairing for multi-agent path finding via large neighborhood search, Proc. AAAI Conf. Artif. Intell. (2022) 10256–10265.
[46] D. Silver, Cooperative pathfinding, Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. 1 (1) (2021) 117–122.
[47] H. Ma, D. Harabor, P.J. Stuckey, J. Li, S. Koenig, Searching with consistent prioritization for multi-agent path finding, Proc. AAAI Conf. Artif. Intell. 33 (01) (2019) 7643–7650, https://doi.org/10.1609/aaai.v33i01.33017643.