An Adaptive Agent-Specific Sub-Optimal Bounding Approach for Multi-Agent Path Finding

MUSTAFIZUR RAHMAN1, MD. AHASANUL ALAM1, MD. MOFIJUL ISLAM2, (Graduate Student Member, IEEE), ISMAT RAHMAN1, MD. MOSADDEK KHAN1, AND TARIQ IQBAL2

1Department of Computer Science and Engineering, University of Dhaka, Dhaka 1000, Bangladesh
2School of Engineering and Applied Science, University of Virginia, Charlottesville, VA 22904, USA

Corresponding author: Mustafizur Rahman (mustafizz996@gmail.com)

This work was supported by the Information and Communication Technology (ICT) Innovation Fund (2020–2021) of Bangladesh Government.

ABSTRACT A Multi-Agent Path Finding (MAPF) problem involves multiple agents who want to reach their destinations without obstructing other agents. Although a MAPF problem needs to be solved for many real-world deployments, solving such a problem optimally is NP-hard. Many approaches have been proposed in the literature that offers sub-optimal solutions to this problem. For example, the Enhanced Conflict Based Search (ECBS) algorithm compromises the solution quality up to a constant factor to gain a notable runtime improvement. However, these algorithms use a fixed global sub-optimal bound for all agents, regardless of their preferences. In effect, with the increase in the number of agents, the runtime performance degrades. Against this backdrop, with the intent to further speed up the runtime, we propose an adaptive agent-specific sub-optimal bounding approach, called ASB-ECBS, that can be executed statically or dynamically. Specifically, ASB-ECBS can assign sub-optimal bound considering an individual agent’s requirement. Additionally, we theoretically prove that the solution cost of ASB-ECBS remains within the sub-optimal bound. Finally, our extensive empirical results depict a notable improvement in the runtime by using ASB-ECBS while reducing the search space compared to the state-of-the-art MAPF algorithms.

INDEX TERMS Multi-agent path finding, robots, collision avoidance, path planning, scalability.

I. INTRODUCTION

Robots are started to be widely employed in various environments, including automating warehouses [1]–[4], controlling traffic [5], robot-guided evacuation [6], and human-robot interaction [7]–[9]. In many of these environments, multiple robots are deployed forming a multi-agent system, and each agent interacts with other agent or humans to complete an assigned task [10]–[13]. In doing so, agents need to work cooperatively with other agents to find conflict-free paths by avoiding other agents to improve the productivity and efficiency of the multi-agent systems.

In the literature, finding the conflict-free paths for a set of cooperative agents from their initial positions to their respective goals is often termed as a Multi-Agent Path Finding (MAPF) Problem [14]. Fig. 1 shows a simple MAPF problem where three robots try to reach their goals without colliding with each other. Over the years, several algorithms have been developed to solve the MAPF problems. These can be broadly categorized as optimal and sub-optimal algorithms. The optimal algorithms aim to minimize one of the following cost functions: (i) Sum of Individual Costs (SIC) or (ii) Makespan. In this context, SIC indicates the sum of time-steps for each agent to reach its goal while makespan means the maximum of time-steps of all the agents to reach their goals (Section II for detail).

Among the optimal algorithms, Conflict Based Search (CBS) and its variants have widely been explored in the literature. These approaches try to avoid exploring higher dimensional spaces by imposing constraints to each individual agent [15]. CBS is mainly a decoupled algorithm [16]–[19] that employs two levels of searching: high-level and low-level. At the high-level search, it operates as a tree. At each node of the tree, it imposes constraints on individual agents and directs the low-level search maintaining the constraints. If the low-level search fails to
find conflict-free paths for the agents, the high-level search generates two new nodes in the tree, adding constraints to resolve a conflict between two agents. CBS repeats this process until a valid solution is found and tries to minimize the cost function by expanding the node of the tree with the least cost. However, as solving a MAPF problem optimally has proven to be NP-hard, the optimal MAPF solvers, such as CBS and its variants, are not scalable for a large number of agents [20–23].

In contrast, the sub-optimal algorithms try to address the scalability issue of the optimal solvers by compromising the solution quality. These algorithms can be broadly classified into unbounded and bounded approaches. Unbounded sub-optimal solvers, such as Diversified-path and Database-driven algorithm (DDM) [24], Cooperative A* (CA*) [25], Push and Rotate (PPR) [26], Priority Inheritance with Backtracking (PIBT) [27] and some learning based techniques [28], [29] generally aim to produce the solutions as fast as possible, often leading to large deviation from optimality.

To minimize this large deviation from optimality, bounded sub-optimal solvers provide a trade-off between runtime and solution quality [30]. For instance, Bounded CBS (BCBS(\(W_H, W_L\))) [31] utilizes a focal search algorithm [32] with two lists, (i) open and (ii) focal, to store nodes to guarantee \(W\)-suboptimality, where \(W\) is manually specified upper bound of the solution cost. BCBS leverages the focal list for both the high and low-level searches, with \(W_H\) and \(W_L\) sub-optimal bounds for high and low-level solution paths, respectively. Thus, the overall sub-optimal bound of BCBS(\(W_H, W_L\)) is \(W_H \times W_L\). However, there is no efficient method to distribute the weight \(W\) between \(W_H\) and \(W_L\) so that \(W_H \times W_L \leq W\). Therefore, BCBS struggles to find the optimal \(W_H\) and \(W_L\) to maintain the overall solution cost within \(W\). To address this issue, Enhanced CBS (ECBS) [31] utilizes a single fixed global \(W\) both for high and low-level searches of CBS. This global sub-optimal bound aids ECBS in reducing the complexity of the high-level search and improves runtime.

It is worth noting that, although ECBS adds more flexibility at the high-level than BCBS(\(W_H, W_L\)), a higher sub-optimal bound for finding conflict-free paths for a large number of agents can degrade the runtime performance of the ECBS algorithm [33], [34]. This is because ECBS employs a constant global sub-optimal bound \(W\) for all agents regardless of their requirement for finding the solution paths. Consequently, for a higher value of \(W\), ECBS often produces a longer solution path for an agent. Moreover, the low-level search space becomes too wide due to the large constant global sub-optimal bound, which increases the runtime of the ECBS algorithm [35], [36].

To address the aforementioned challenges, we propose an adaptive agent-specific sub-optimal bound assignment approach that we call ASB-ECBS. ASB-ECBS assigns a sub-optimal bound to each agent based on its requirement of finding a conflict-free path to reach the corresponding goal. These agent-specific sub-optimal bounds reduce the low-level search space, especially for large \(W\), which improves the MAPF solver’s runtime. Depending on the bound assignment process, we develop two versions of ASB-ECBS: Static ASB-ECBS (SASB-ECBS) and Dynamic ASB-ECBS (DASB-ECBS).

The key contributions of this paper are listed below:
1) We design a novel adaptive agent-specific sub-optimal bounding approach, ASB-ECBS to improve the runtime and low-level search space of the bounded sub-optimal MAPF solver. Based on the agent’s conflict resolution requirement this approach has two versions: i) SASB-ECBS and ii) DASB-ECBS.
2) We theoretically show that ASB-ECBS retains the \(W\)-suboptimality of the solution cost for the MAPF problems.
3) Our empirical results suggest that ASB-ECBS improves runtime and low-level search space for an increasing number of agents. This improvement is up to a factor of 1.5 compared to the state-of-the-art bounded MAPF algorithms for large sub-optimal bounds.
4) We also report comparative results of the two versions of the ASB-ECBS algorithms. It indicates that SASB-ECBS has improved runtime and low-level search space performances than DASB-ECBS in large and simple graphs whereas DASB-ECBS performs better than SASB-ECBS in small and complex graphs.

The remainder of this paper is structured as follows. In Section II, we review the current state of the literature and provide the preliminaries required for our work. Section III presents our proposed approach ASB-ECBS in details for solving the MAPF problem. Section IV provides theoretical proof and complexity analysis of ASB-ECBS. Then the Section V and VI describe the experimental result and qualitative analysis of ASB-ECBS, respectively. Finally, the section VII concludes the work presented in this paper and sheds light on the future research direction.
II. BACKGROUND

In this section, we define the MAPF problem and discuss the ECBS algorithm in detail. The notations and their description related to the formulation is listed in the Table 1.

A. MULTI-AGENT PATH FINDING

In MAPF, a graph $G(V, E)$ along with a set of $k$ agents $\{a_1, \ldots, a_k\}$ are given. Here, $V$ denotes the set of nodes and $E$ denotes the set of edges in $G$. Graph $G$ can be directed or undirected based on the underlying environment. Each agent $a_i$ has a unique initial $s_i \in V$ and goal $g_i \in V$ positions. At each time-step $t$, an agent can either visit any of its adjacent nodes or wait at its current position. Both the actions, move and wait, incur a cost of 1. An agent $a_i$ starts from its initial position $s_i(0) = s_i$ and try to reach its destination following a path. A path to the goal for an agent $a_i$ is a set of vertices $\{s_i(0), \ldots, s_i(k)\}$, where $t_i$ is the number of time-steps required for an agent $a_i$ to reach its goal position and remain there. Thus, there exists a minimum $t_i$ such that $s_i(t_i) = g_i$ for each $t \geq t_i$ [37]. A solution of the problem is defined by a set of conflict-free paths for all agents $\{a_1, \ldots, a_k\}$ while aiming at minimizing a solution cost function which includes either sum of individual cost (SIC) or makespan. The SIC is aiming at minimizing a solution cost function which includes

\[ W = \sum_{a_i} f_a(s_i), \]

where $f_a(s_i)$ denotes the cost of going to the goal position $g_i$ from node $s_i$. \[ W = \sum_{a_i} f_a(s_i), \]

The makespan indicates the maximum $t_i$ among all the agents, which can be defined as, $max_{1 \leq i \leq k} t_i$ [14]. In this paper, we aim to minimize the runtime of the solver while maintaining the bounded sub-optimal SIC.

While moving towards their goals, agents need to avoid two types of conflicts: vertex and edge conflict. The vertex conflict is defined by the tuples $(a_i, u, t-1) \rightarrow (a_i, v, t)$, that is, agent $a_i$ is going to node $v \in V$ from node $u \in V$ and $(a_j, w, t-1) \rightarrow (a_j, v, t)$ where agent $a_j$ is also going to the node $v \in V$ from node $w \in V$. Consequently, $a_i$ and $a_j$ try to occupy the same vertex $v$ at the same time-step $t$ experiencing a vertex conflict. While the edge conflict can be defined by $(a_i, u, t) \rightarrow (a_j, v, t)$, where agent $a_i$ tries to go to vertex $v \in V$ from $u \in V$ between time-steps $t$ and $t+1$. At the same time-step $t$, another agent $a_j$ tries to go to vertex $u \in V$ from $v \in V$ incurring head-to-head conflict in $t+1$; that is, $(a_j, v, t) \rightarrow (a_i, u, t+1)$ [39]–[41].

B. ENHANCED CBS

Enhanced CBS (ECBS) is a sub-optimal variant of the Conflict Based Search (CBS) algorithm. It utilizes the focal-search algorithm [32] to ensure the W-suboptimality of the MAPF solution. Here, $W$ is a manually specified global upper bound of the solution cost, also known as weight. Algo. 1 contains the steps of ECBS (except [Algo. 1: lines 4-6] and [Algo. 1: lines 14-16]). ECBS uses a two-level focal-search to find a solution. At the high-level, a constraint tree is formed to find the set of constraints that produce a valid solution for a deployed MAPF problem. A high-level node $N$ consists of three information: i) a set of constraints to avoid conflicts, ii) tentative solution paths for $k$ agents that satisfy all the assigned constraints and iii) solution cost [Algo. 1: lines 1-3]. At low-level, ECBS employs focal search to find the sub-optimal paths for agents while maintaining the corresponding high-level constraints. Both high and low-level focal-searches simultaneously maintain two types of the priority lists, OPEN and FOCAL. The high-level OPEN contains all the high-level nodes where nodes are sorted according to an admissible cost function $f(N)$. Here, $f(N) = \sum_{i=1}^{k} f_{\min}(n)$, indicates a lower bound of the solution cost below the constraint tree node $N$. Moreover, the low-level OPEN for an agent $a_i$ is sorted according to a function $f_{i}(n)$ where $f_i(n) = g_i(n) + h^i(n)$. Here, $g_i(n)$ is the distance of node $n$ from the initial position $s_i(0)$ and $h^i(n)$ is the admissible cost of going to the goal position $g_i$ of agent $a_i$ from the node $n$. The high-level FOCAL contains all the nodes $N$, where $N \in OPEN$ and $NCost \leq W \times LB$ [Algo. 1: line 9]. Here, $LB$ (Lower Bound) is the minimum $f(N)$ among all the nodes in high-level OPEN. The low-level FOCAL is similar to the high-level FOCAL except the LB value. The low-level LB value for an agent $a_i$ is the minimum $f_{i}(n)$, i.e., $f_{\min}(n)$. Additionally, the low-level FOCAL utilizes the low-level OPEN.

At each iteration [Algo. 1: lines 8-19], ECBS chooses the node $N$ from FOCAL that has the minimum number

| Notations | Descriptions |
|-----------|--------------|
| $G(V, E)$ | Underlying Graph |
| $V$ | Set of nodes in $G$ |
| $E$ | Set of edges in $G$ |
| $K$ | Number of agents |
| $a_i$ | i$^{th}$ agent, where $i \in K$ |
| $s_i$ | Starting position of agent $a_i$ |
| $g_i$ | Goal position of agent $a_i$ |
| $t$ | Timestep |
| $s_i^t$ | Position of agent $a_i$ at timestep $t$ |
| $f(N)$ | Admissible cost function of high-level node $N$ |
| $g_i^0(n)$ | Distance of node $n$ from the initial position $s_i(0)$ |
| $h_i^0(n)$ | Admissible cost of going to $g_i$ from node $n$ |
| $f_i(n)$ | Admissible cost function of low-level node $n$ for agent $a_i$ |
| $W$ | The global sub-optimal bound |
| $w_i$ | Sub-optimal bound for agent $a_i$ |
| $W_H$ | High-level global sub-optimal bound |
| $W_T$ | Low-level global sub-optimal bound |
| $LB$ | Level wise (High or Low) minimum cost |
| $off set$ | The amount of weight requires for each conflict |
| $C_{opt}$ | Optimal solution cost |
| $d_i$ | Depth of high-level constraint tree |
| $d_g$ | Depth of solution in the Low-level search |
Algorithm 1 ASB-ECBS($W, A, \text{Method}$)

Input: $W$: sub-optimal bound, $A$: set of agents, $\text{method}$: the version of ASB-ECBS
Output: high-level node, or no solution
1 $R.\text{Constraints} \leftarrow \phi$ \hspace{0.5cm} ($R =$ root node)
2 $R.\text{Solution} \leftarrow \text{low_level}(W)$ \hspace{0.5cm} $R.\text{Cost} \leftarrow \text{solution_cost()}$
   \hspace{0.5cm} \{\text{SIC or Makespan}\}
3 \hspace{0.5cm} \textbf{if} \hspace{0.5cm} \text{method} = \text{SASB} - \text{ECBS} \hspace{0.5cm} \textbf{then}
4 \hspace{1cm} $A \leftarrow \text{weight_assignment}(W, R, A)$ \hspace{0.5cm} \{Algo.2\}
5 \hspace{0.5cm} \textbf{end}
6 $\text{OPEN} \leftarrow \text{append}(\text{OPEN}, R)$ \hspace{0.5cm} \textbf{while} \hspace{0.5cm} $\text{OPEN} \neq \emptyset$ \textbf{do}
7 \hspace{1cm} $\text{FOCAL} \leftarrow \text{append}(\text{FOCAL}, N)$;
8 \hspace{2cm} $N \in \text{OPEN} \& N.\text{cost} \leq W.\text{LB}$
9 \hspace{1cm} $N \leftarrow \text{node with minimum conflict in FOCAL}$
10 \hspace{2cm} \textbf{if} \hspace{0.5cm} $N$ has no conflict \textbf{then}
11 \hspace{3cm} \textbf{return} $N$
12 \hspace{2cm} \textbf{end}
13 \hspace{1cm} \textbf{if} \hspace{0.5cm} \text{method} = \text{DASB} - \text{ECBS} \hspace{0.5cm} \textbf{then}
14 \hspace{2cm} $A \leftarrow \text{weight_assignment}(W, N, A)$ \hspace{0.5cm} \{Algo.2\}
15 \hspace{1cm} \textbf{end}
16 \hspace{1cm} $C \leftarrow \text{get_conflict}(N)$
17 \hspace{1cm} $\text{OPEN} \leftarrow \text{make_child_node}(N, C, \text{OPEN})$
18 \hspace{0.5cm} \textbf{end}
19 \hspace{0.5cm} \textbf{return} No solution

of conflicts. In the case of a tie, the node with minimum solution cost is chosen. If $N$ is a goal node (conflict-free), it is returned as a sub-optimal solution [Algo. 1: lines 11-13]. Otherwise, a conflict has occurred at the tentative solution of the high-level node $N$ [Algo. 1: line 17]. To resolve the conflict, \text{make_child_node} method creates two new child nodes, $N_1$ and $N_2$, of the parent node $N$. Both of them inherit all the parent node $N$ constraints and add an extra constraint to avoid the current conflict. Then the low-level focal-search is employed for each agent involved in the conflict to produce the sub-optimal path. The low-level focal-search returns not only the solution path but also the minimum $f^*(n)$ of the low-level $\text{OPEN}$ list, $f^*_\text{min}(n)$. This indicates a lower bound on the shortest path of the agent $a_i$. Finally, the updated $\text{OPEN}$ is returned from the \text{make_child_node} method [Algo. 1: line 18]. The iteration continues until a valid solution is found or $\text{OPEN}$ becomes empty. If the $\text{OPEN}$ becomes empty, that means there is no solution for the given problem [Algo. 1: line 20].

C. LIMITATIONS OF THE ECBS ALGORITHM

Although ECBS has better runtime among all the bounded sub-optimal MAPF solvers, it has limitations when employing a large $W$. In circumstances with few agents, a higher $W$-value allows them to maneuver around other agents more freely, resulting in fewer conflicts. However, in an environment with a large number of agents, the increased $W$ tends to spread the agents apart. Consequently, ECBS experiences some additional conflicts among the agents.

In effect, it requires more conflicts to resolve for the high-level nodes. Fig. 2 visualizes this limitation of ECBS. In Fig. 2, the deep green cells are those that have been explored more by the agents than the light green ones. The red cells are the blocked cells. The right heat map in Fig. 2 is significantly greener than the left heat map, especially towards the edges of the map. This suggests that utilizing a large bound, $W = 2$, spreads the agents more than utilizing a smaller bound, $W = 1.5$. This spread of the agents adds some unnecessary conflicts and increases the total number of the explored low-level nodes by the ECBS algorithm. Consequently, the runtime of the ECBS increases [34], [42].

Additionally, ECBS utilizes a large constant global sub-optimal bound for all agents regardless of their requirement. Consequently, low-level search space of the solver increases. This often results in a longer solution path, which increases the MAPF solver’s runtime. Moreover, the large value of $W$ allows many nodes in the low-level $\text{FOCAL}$ list of ECBS, resulting in runtime overhead.

III. AGENT-SPECIFIC SUB-OPTIMAL BOUNDING

We propose a novel bounded sub-optimal MAPF algorithm, ASB-ECBS, to solve the aforementioned shortcomings. ASB-ECBS is an adaptive agent-specific weight assignment approach to solve the MAPF problem that assigns sub-optimal bound (i.e., weight) to each agent according to its requirement. The idea underlying ASB-ECBS is that all the agents may not require the same global sub-optimal bound ($W$). For instance, if an agent finds a conflict-free path to its assigned goal with a sub-optimal bound less than the global sub-optimal bound $W$ then increasing its sub-optimal bound may not improve the runtime of ECBS. Therefore, the runtime of the solver can be improved by determining the optimum sub-optimal bound for each agent.

Moreover, the sub-optimal bound of an agent is related to the number of conflicts it encounters during the time of finding its goal (i.e., path finding phase). Consequently, an agent that encounters a higher number of conflicts requires a larger sub-optimal bound to resolve or bypass the conflicts.
by taking a longer path. On the other hand, an agent with a relatively lower number of conflicts requires a lower sub-optimal bound. Because extending its sub-optimal bound may result in an extra computational overhead at the low-level search of the solver (Section II-C for detail).

In light of the above reasoning, we develop two adaptive sub-optimal bounds assignment algorithms, called Static-ASB-ECBS (SASB-ECBS) and Dynamic-ASB-ECBS (DASB-ECBS). SASB-ECBS assigns agent-specific sub-optimal bounds once to the agents based on their requirement to find conflict-free paths. On the other hand, DASB-ECBS dynamically assigns agent-specific sub-optimal bounds to the agents, which adapt throughout the runtime according to the agents’ requirements. In a small and complex (high percentage of obstacles and narrow passages) environment with an increasing number of agents, more conflicts are likely to occur. Furthermore, the number of conflicts fluctuates during each iteration. As a result, the requirements of the agents vary over time. In that case, DASB-ECBS is preferable as it adjusts the sub-optimal bound in each iteration based on the agent’s requirements. Contrarily, in a large and simple environment, agents get sufficient free space to move. Thus, they tend to have fewer conflicts, and most of the time, the number of conflicts remains consistent during the iterations. Moreover, the calculation of the sub-optimal bounds in each iteration requires some extra runtime. In that scenario, SASB-ECBS is more suitable as it calculates the sub-optimal bounds only at the beginning and maintains the bounds throughout the execution process.

ASB-ECBS algorithm is complete because it is a more relaxed version of the CBS algorithm, which is optimal and complete [43]. Additionally, ASB-ECBS maintains a systematic search, and all valid solutions are consistent with at least one of the constraint tree nodes in the OPEN list. So, ASB-ECBS will eventually find a solution if there exists any. Therefore, ASB-ECBS guarantees completeness.

A high-level overview of the ASB-ECBS can be obtained by the flowchart of Fig. 3. The algorithm first initializes individual shortest path of each agent without considering the presence of other agents. Then the algorithm enters into a high-level component where the agent-specific sub-optimal bound is calculated. Some constraints are also added for a chosen conflict in this component. Then the low-level component is utilized based on the constraints added in high-level. If there is no conflict among the agents then the algorithm terminates and returns the obtained solution. Otherwise, based on the variant of the ASB-ECBS the algorithm goes to the different part of the high-level component. If the SASB-ECBS variant of the ASB-ECBS runs then no weight calculation happens. However, in the case of DASB-ECBS, the agent-specific sub-optimal bound is calculated again. Thus, the high-level to low-level loop continues until a conflict free bounded solution is obtained. In the following sections, we present SASB-ECBS and DASB-ECBS, in detail.

![Flowchart of ASB-ECBS algorithm.](image)

**Algorithm 2 Weight_Assignment(W, N, A)**

**Input:** W: sub-optimal bound, N: high-level node, A: set of agents

**Output:** A with updated sub-optimal bound

1. \(\text{count} \leftarrow \max_{a \in A} \text{conflict}_\text{count}(a, N)\)
2. \(\text{offset} \leftarrow \frac{(W-1)}{\text{count}}\)
3. \(\text{for } a_i \in A \text{ do}\)
4. \(n = (a_i).\text{conf}\)
5. \((a_i).w \leftarrow 1 + (n \times \text{offset})\)
6. \(\text{end}\)
7. \(\text{return } A\)

**A. STATIC ASB-ECBS (SASB-ECBS)**

SASB-ECBS extends the ECBS algorithm while utilizing a different weight assignment mechanism. Rather than a fixed sub-optimal bound \(W\) for all agents, SASB-ECBS assigns different sub-optimal bounds to the agents. Once the bounds are assigned to the agents, these bounds remain constant throughout the path finding phase. Pseudocode for ASB-ECBS is shown in Algo. 1 where lines 4-6 are for SASB-ECBS and lines 14-16 are for DASB-ECBS. Primarily, the high-level root node \(R\) chooses an optimal path for each agent without considering the presence of other agents [Algo. 1: line-2]. Then a \textit{weight_assignment} function [Algo. 2] is called to generate weights for the agents based
on their number of conflicts in the current path [Algo. 1: lines 4-6]. The function takes sub-optimal bound \( W \), a high-level node \( N \) (\( R \) for high-level root node), and the set of agents \( A \) as the inputs. Then it counts the number of conflicts for each agent against the path of other agents. After that, the maximum number of conflicts [Algo. 2: line 1] is utilized to calculate the amount of weight requires for each conflict, called offset [Algo. 2: line 2]. Then, for each agent, \( a_i \), the conflict count of that agent, \((a_i)_{\text{conf}}\), is multiplied with the offset [Algo. 2: lines 3-6] value. Consequently, each agent gets a different sub-optimal bound, which is proportional to the number of conflicts of its path with others. Finally, the set of agents, \( A \), with updated weights is returned to line 5 of the Algo. 1.

Afterwards, \( R \) is pushed in the OPEN (Section II-B). Then ASB-ECBS utilizes the high-level search of ECBS [Algo. 1: lines 8-20]. That is, the top node of FOCAL, with the minimum number of conflicts is chosen for expansion [Algo. 1: line-10]. If it is a goal node (paths with no conflicts), then it is returned as the solution [Algo. 1: lines 11-13]. Otherwise, to resolve the conflict, two child nodes of the current node are created [Algo. 1: lines 17-18] with added constraints for the conflicting agents. Finally, the high-level iteration continues until a valid solution is found or OPEN becomes empty; that is, no solution exists [Algo. 1: line-20].

B. DYNAMIC ASB-ECBS (DASB-ECBS)

DASB-ECBS dynamically adjusts the sub-optimal bound of each agent based on its requirement to consider the variation of conflicts that occur during the search process. In addition to that, since the number of conflicts decreases over time as the search progresses, the offset value in DASB-ECBS tends to increase gradually over time as well. Therefore, DASB-ECBS steadily increases the agent-specific weight during the runtime. As a result, the low-level search space gradually expands and avoids wasteful computations. In Algo. 1, lines 14-16 present the DASB-ECBS version of ASB-ECBS. Similar to SASB-ECBS, DASB-ECBS also utilizes weight_assignment function to assign sub-optimal bound to the agents [Algo. 1: line 15]. Moreover, the execution of Algo. 1 for DASB-ECBS are similar to the SASB-ECBS (Section III-A). Algo. 1 (lines 14-16) demonstrates that in the case of DASB-ECBS, the weight_assignment function is repeatedly executed until a valid solution is found. Consequently, each agent gets sub-optimal bound according to its requirement in each iteration.

C. ILLUSTRATIVE EXAMPLE

In this section, we demonstrate the procedure of solving the MAPF problem described in Fig. 1 with the help of ASB-ECBS approach for the sub-optimal bound \( W = 2 \). In Fig. 4, the 5 \times 5 map is given where \( a_i \) and \( g_i \) denote the start and the goal positions of the \( i^{th} \) agent, respectively. The high level constraint tree of ASB-ECBS for the given environment is also shown in Fig. 4. Each node of the constraint tree consists of the following five information: i) a set of constraints (Con), ii) tentative path for each agent satisfying the constraints of the current node (Sol), iii) conflict count for each agent (Conf), iv) agent-specific sub-optimal bounds (\( W_i \)), v) sum of individual cost (Cost).

Since there is no constraint in the root high level node, each agent finds a solution path without considering other agents. But, this causes a vertex conflict at node \((5,3)\) in time-step 2 between \( a_1 \) and \( a_3 \). Furthermore, an edge conflict occurs between \( a_1 \) and \( a_2 \) while \( a_1 \) tries to visit \((3,1)\) from \((4,1)\) and \( a_2 \) tries to visit \((4,1)\) from \((3,1)\) between time-steps 5 and 6. Thus, \( a_1 \) has 2 conflicts and both \( a_2 \) and \( a_3 \) have 1 conflict. So, the maximum conflict count is 2. As agent \( a_1 \) has the maximum number of conflicts, it gets the maximum sub-optimal bound 2. Since each agents \( a_2 \) and \( a_3 \) has a single conflict, the sub-optimal bound for them is \( 1 + 1 \times (2 - 1)/2 = 1.5 \). To resolve the conflict between \( a_1 \) and \( a_3 \), ASB-ECBS constructs two new child nodes. The conflict also incurs two new constraints, \{\{c_1\}a_1, (5,3), 2\} and \{\{c_2\}a_3, (5,3), 2\} to the left and right child, respectively. With the updated constraint, each node will invoke a low-level focal search for the conflicted agents. Then the path of conflicted agents is updated with the result of the low-level focal search. Finally, a conflict-free node (goal node) is found and returned as the solution by the ASB-ECBS algorithm.

IV. THEORETICAL ANALYSIS

A. THE PROOF OF BOUNDED SUB-OPTIMALITY

Theorem: The range of the weights assigned by SASB-ECBS and DASB-ECBS is \([1, W]\).

Proof: Let, \( W \) be the maximum sub-optimal bound and \( \mathcal{W} = \{w_1, \cdots, w_k\} \) where \( w_i \) is the sub-optimal bound for \( i^{th} \) agent. Our algorithms assign maximum sub-optimality to that agent which has maximum conflicts. Hence, the maximum value of \( w \) is \( 1 + \max \{1 \times (\frac{W - 1}{\max})\} = W \). Besides, the minimum probable weight is applicable for that agent which does not have any conflict. Therefore, the minimum weight can be, \( 1 + 0 \times (\frac{W - 1}{\max}) = 1 \). So, the range of weights is \([1, W]\). Hence, we can conclude by following equation,

\[
\forall w_i \in \mathcal{W}(w_i \leq W) \tag{1}
\]
Proposition: SASB-ECBS and DASB-ECBS algorithms are W-suboptimal.

Proof: Let C is the SIC of the sub-optimal solution and \( C_{opt} \) is the SIC of the optimal solution. Let, \( C_{opti} \) is the optimal individual cost of the \( i^{th} \) agent. The following equation shows the sub-optimal bound of any W-bounded sub-optimal algorithm [31].

\[
C \leq W \times C_{opt}
\]

The following equation shows the sub-optimal bound of the SASB-ECBS and DASB-ECBS algorithms.

\[
C \leq \sum_{n=1}^{k} W_n \times C_{opta}
\]

We can derive the following equations by using Eq. 1,

\[
\sum_{n=1}^{k} W_n \times C_{opta} \leq \sum_{n=1}^{k} W \times C_{opta}
\]

\[
\sum_{n=1}^{k} W_n \times C_{opta} \leq W \times C_{opt}
\]

\[
C \leq W \times C_{opt}
\]

We can derive Eq. 2 from the sub-optimal bound equations of SASB-ECBS and DASB-ECBS algorithms. Thus, SASB-ECBS and DASB-ECBS algorithms are W-suboptimal.

B. COMPLEXITY ANALYSIS

In this section, we analyze and compare the computational complexity of ECBS and ASB-ECBS algorithms. The computational complexity of ECBS is found by analyzing the execution of the algorithm. At high-level, the algorithm searches in a binary constraint tree to find a solution. The complexity of the tree is \( O(2^{d_t}) \) where \( d_t \) is the depth of the tree [43]. For each node of the tree, a low-level focal search is also called which is a variant of the classical A* search algorithm. In the worst case, the complexity of A* search algorithm is \( b^d_e \) where \( b \) is the branching factor and \( d_e \) is the depth of the solution in the low-level search [44]. Since, in this study, the agents have unitary cost for each action in the low-level search, we can replace \( d_e \) by the solution cost of the single agent. The upper bound of the solution cost is \( W \times C^* \) where \( W \) (weight) is the fixed upper bound and \( C^* \) is the optimal solution cost. So, combining both the high-level and low-level complexity, we find the overall computational complexity i.e. \( O(b^W \times C^* \times (2^{d_t})) \).

As ASB-ECBS utilizes the ECBS algorithm, we assume that both the algorithms have the same high-level tree search complexity. Contrarily, the low-level search complexity is different as ASB-ECBS utilizes agent-specific weight \( (w) \) instead of the global sub-optimal bound \( (W) \). For calculating the agent-specific weight, ASB-ECBS requires an \( O(K^2) \) loop where \( K \) is the number of agents. The SASB-ECBS variant of ASB-ECBS finds this agent-specific weight only once at the root high-level node. Therefore, the overall complexity of SASB-ECBS is \( O(K^2 + (b^W \times C^* \times (2^{d_t}))) \). However, the DASB-ECBS variant utilizes the \( O(K^2) \) loop at each high-level node. Therefore, its complexity becomes \( O((K^2 + b^W \times C^*) \times (2^{d_t})) \). The maximum value of \( K \) for this experiment is only around hundreds and also in the case of a quadratic computational complexity being added with an exponential computational complexity the quadratic part can be ignored. Thus, we can neglect the impact of the \( O(K^2) \) loop. Consequently, the overall time complexity of ASB-ECBS is \( O(b^W \times C^* \times (2^{d_t})) \). In the section IV-A, we have proved that \( w \leq W \). So, based on the above mentioned assumptions, the computational complexity of ASB-ECBS, \( O(b^W \times C^* \times (2^{d_t})) \), the complexity of the ECBS algorithm.

V. EXPERIMENTAL RESULT

We have empirically evaluated the performance of ASB-ECBS comparing with the state-of-the-art MAPF algorithms. To ensure fairness, we considered only the state-of-the-art bounded sub-optimal solvers, ECBS and BCBS \( (W_H, W_L) \) with both SASB-ECBS and DASB-ECBS. We have experimented on the following four benchmark maps that have been used extensively to evaluate MAPF algorithms.

- 4-connected grids are a 32 x 32 map with 20% obstacles and an 8 x 8 map with 12% obstacles [27], [31], [35].
- Kiva-like map \( (22 \times 53) \) is a warehouse map in the Kiva systems. In this map, several autonomous robots picks different shelving units (initial position) and drops them at particular locations (goal location) [35], [45].
- Roundabout environment \( (24 \times 48) \) is a map with four rooms through a roundabout [36].

We have generated 100 different experiment instances with random start and goal positions for each map and set of agents. We have executed all the solvers on these experiment instances for comparison. As the sub-optimal algorithms may need a long runtime depending on the maps, the number of agents, and weights, we have limited the runtime to 180 seconds. An execution has been considered incomplete if a solver returns failure or does not return any solution within the time limit. For evaluation purpose, in line with prior work [43], [34], [35], we have considered the following two metrics of the successfully solved experiment instances:

- Average runtime of each agent.
- Average number of nodes (low-level search space) generated by each agent on the low-level search.

We have used C++\(^1\) to implement the algorithms and conducted all the experiments on a machine with Intel Core i5 1.5 GHz CPU and 8 GB RAM.

A. IMPACT OF WEIGHTS

We have evaluated the impact of sub-optimal bound or weight, \( W \), on the runtime and low-level search space of

\(^1\)The code is available at https://github.com/MustafizSaadi/ASB-ECBS
FIGURE 5. Impact of weight (sub-optimal bound) on average runtime and number of nodes generated by each agent in low-level search of MAPF algorithms. Average over the successfully solved experiment instances are shown. The legend is shown below the graphs. The algorithms are indicated by the legend with the same color and the same marker style. For an experiment map, black indicates blocked cells while white indicates open cells.

As of previous works [31], [34], [35], we examine the impact of different weights on the MAPF algorithms by assigning the sub-optimal bound from the range [1.1-3.5]. We have conducted this empirical analysis by employing 20, 70, 100, and 150 agents on 8 × 8, 32 × 32, Kiva-like, and Roundabout maps, respectively. The experimental results of average runtime and the low-level search space of each agent are shown in Fig. 5.

1) RESULTS
The reported results depicted in Fig. 5 suggest that ASB-ECBS achieves smaller runtime and low-level search space than ECBS and BCBS(W, 1) on all benchmark maps for the most of the weights. Between the ASB-ECBS algorithms, DASB-ECBS outperforms SASB-ECBS both in terms of runtime and low-level search space in small and complex graphs (high percentage of obstacles), such as 8 × 8 and Kiva-like maps. However, SASB-ECBS outperforms DASB-ECBS and BCBS(W, 1) both in terms of runtime and low-level search space in large and simple graphs (32 × 32 and Roundabout maps).

2) DISCUSSION
As ASB-ECBS adaptively assigns weights to each agent based on their requirements (i.e., number of conflicts), it reduces the runtime and low-level search space. This adaptive agent-specific weight assignment approach produces solution paths with a smaller sub-optimal bound for some agents and a larger sub-optimal bound for other agents. Consequently, it improves the runtime by reducing the computation of low-level searches. On the other hand, ECBS assigns a fixed global weight to all the agents, which leads to the large sub-optimal path irrespective of their requirement.

As a result, most of the agents have larger runtime and low-level search space.

Between the ASB-ECBS algorithms, the degraded runtime and low-level search space performances of SASB-ECBS than DASB-ECBS are due to the characteristics of the 8 × 8 and Kiva-like maps. In those maps, the number of free cells for movements is very few compared to the number of deployed agents. As a result, multiple agents intend to use the same nodes to reach their destinations. This incurs a large number of conflicts among the agents and it varies during the execution time. Since SASB-ECBS assigns weight considering only the initial conflicts of the agents, this static weight may not remain best throughout the execution time. Therefore, SASB-ECBS takes more runtime and low-level search space to find solutions than the DASB-ECBS and ECBS algorithms. In contrast, DASB-ECBS considers the agents’ requirements at each iteration step and assigns the weights accordingly. This adaptive weight assignment helps DASB-ECBS to find the solution faster than ECBS in 8 × 8 and Kiva-like maps.

On the other hand, there are sufficient free spaces to move in the 32 × 32 and Roundabout maps, unlike the two maps mentioned above. As a result, the number of conflicts almost remains consistent throughout the execution time. Therefore, the computation of weights at each iteration step is not required. Because this adds some extra runtime overhead to the DASB-ECBS algorithm. Since SASB-ECBS assigns weight only at the beginning of the execution time, it performs better than the DASB-ECBS algorithm both in terms of runtime and low-level search space in 32 × 32, and Roundabout maps. Nevertheless, for a smaller sub-optimal bound, in 32 × 32 map, ECBS has better runtime and low-level search space performances than ASB-ECBS. Because, in the case of a smaller sub-optimal bound, all the agents tend to find their solution path very close to the optimal one, which
increases the runtime of ASB-ECBS. However, with a large number of agents, a higher sub-optimal bound is required to increase scalability where ASB-ECBS outperforms other sub-optimal MAPF solvers.

B. IMPACT OF NUMBER OF AGENTS

In this experimental analysis, we have evaluated the impact of the number of agents on the runtime and low-level search space of SASB-ECBS, DASB-ECBS, ECBS, and BCBS($W, 1$). We have conducted this empirical analysis by varying the number of agents on the same four benchmark maps that we utilized in the previous experiment. In doing so, we have also fixed the sub-optimal bound $W$ for all the agents to $1.3$ in $8 \times 8$, $3$ in Kiva-like map, $2$ in $32 \times 32$ map, and $3.25$ in Roundabout map. We have chosen the weights because other works that analyze the ECBS algorithm have also used similar weights [31], [34], [35]. The experimental results of average runtime and the low-level search space of each agent are depicted in Fig. 6.

1) RESULTS

The experimental results in all four benchmark maps (Fig. 6) indicate that in the case of an increasing number of agents ASB-ECBS outperforms ECBS and BCBS both in terms of runtime and low-level search space. In $8 \times 8$, Kiva-like, $32 \times 32$, and Roundabout maps, the improvement is around a factor of $1.5$ for the number of agents $20, 75, 140$, and $150$, respectively. The experimental results also suggest that among the two versions of ASB-ECBS, in $8 \times 8$ and Kiva-like maps, even though SASB-ECBS performs similar to ECBS on several occasions, DASB-ECBS produces better runtime and low-level search space than the other algorithms. On the other hand, in $32 \times 32$ and Roundabout maps, SASB-ECBS outperforms other algorithms most of the time.

2) DISCUSSION

This improved runtime and low-level search space performances of our proposed algorithms can be explained by the consequence of the spread of a large number of agents. The ASB-ECBS algorithms control the spread of the agents adaptively by considering each agent’s conflict separately, which reduces the number of conflicts. As a consequence, ASB-ECBS algorithms have improved runtime and low-level search space performances. However, ECBS and BCBS do not change the agent-specific sub-optimal bounds to resolve the agent-specific conflict requirement. This degrades the overall runtime and low-level search space performances by increasing the number of conflicts that need to be resolved by the high-level search of the solvers.

In the small and complex environments like $8 \times 8$ and Kiva-like maps, SASB-ECBS takes more runtime and low-level search space than ECBS for a large number of agents due to a large number of conflicts. Because SASB-ECBS assigns weight at the beginning of the search and maintains the same weights throughout the execution time. Consequently, the variation of the number of conflicts among the agents during the execution time is not considered by SASB-ECBS. As a result, SASB-ECBS can not find a solution fast in these maps. Since this variation of conflicts is considered by DASB-ECBS, it can find the best sub-optimal bound for each agent throughout the execution time. Therefore, DASB-ECBS outperforms the state-of-the-art MAPF algorithms in $8 \times 8$ and Kiva-like maps.

On the other hand, in both $32 \times 32$ and Roundabout maps, the number of conflicts almost remains consistent throughout the execution time due to the sufficient free space of the maps. Therefore, the requirement of the agents also remains the same. Consequently, it is not necessary to calculate weights at each iteration step which includes additional runtime. As a result, SASB-ECBS finds the solution faster than
DASB-ECBS. However, for a large number of agents, the free space of the maps is reduced. Therefore, the conflicts among the agents vary throughout the execution time and result in improved runtime and low-level search space performances by DASB-ECBS than SASB-ECBS.

VI. QUALITATIVE ANALYSIS

We have qualitatively analyzed the ECBS and ASB-ECBS algorithms in Fig. 7. We have experimented with the same four benchmark maps mentioned in Section V. In this figure, the light green cells are explored less by the agents than the deep green cells, while the red cells are the blocked cells.

Results and Discussion: The results in Fig. 7 show that ECBS produces large and deep green clusters. This implies that a greater proportion of nodes are explored more than the remaining nodes of the map. These often explored nodes cause several unwanted conflicts. To resolve those conflicts, ECBS needs to explore additional low-level nodes, resulting in a higher runtime.

On the other hand, the results in Fig. 7 suggest that the ASB-ECBS algorithms produce smaller and light green clusters compared to ECBS. That means, in ASB-ECBS, a smaller proportion of nodes are explored more than the remaining nodes of the map. Moreover, the expansion of nodes across the map is evenly distributed compared to ECBS. Because ASB-ECBS leverages agent-specific requirements to find the conflict-free path, which leads to fewer conflicts and a smaller low-level search space. Additionally, as DASB-ECBS considers the agent-specific requirement throughout the execution time, it performs well in small and complex environments (8 × 8 and Kiva-like). Contrarily, SASB-ECBS considers the agent-specific requirement only at initial phase of the execution time (32 × 32 and Roundabout). Therefore, it performs better in large and simple environments.

VII. CONCLUSION AND FUTURE WORK

In this work, we have introduced a novel adaptive agent-specific sub-optimal bounding approach, ASB-ECBS, to solve the MAPF problem. Specifically, we propose two variants of the ASB-ECBS, SASB-ECBS and DASB-ECBS, based on the agent’s conflict resolution requirement that aim to improve runtime performance. In theoretical evaluation, we prove that ASB-ECBS is bounded sub-optimal. The extensive experimental results on four benchmark maps suggest that SASB-ECBS and DASB-ECBS outperform the state-of-the-art MAPF algorithms by reducing the runtime and low-level search space in simple and complex environments, respectively. These results demonstrate the significance of leveraging the adaptive agent-specific sub-optimal bounding algorithms in solving the MAPF problems in multi-robot systems.

In future, we intend to extend our proposed approaches to incorporate other constraints, such as the agents’ shape, speed, and smaller sub-optimal bound. Furthermore, we plan to deploy and evaluate our proposed algorithms in real-world settings to assist the robot in finding conflict-free paths.

ACKNOWLEDGMENT

This research is supported by the Information and Communication Technology (ICT) Innovation Fund (2020-2021) of Bangladesh Government.

REFERENCES

[1] P. R. Wurman, R. D’Andrea, and M. Mountz, “Coordinating hundreds of cooperative, autonomous vehicles in warehouses,” AI Mag., vol. 29, no. 1, p. 9, 2008.
MD. AHASANUL ALAM received the bachelor’s degree in computer science and engineering from the University of Dhaka, Bangladesh. He is currently a Research Student with the Cognitive Agents and Interaction Laboratory (CAIL), University of Dhaka. His research interests include multi-agent systems, reinforcement learning, and machine learning.

MD. MOFIJUL ISLAM (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in computer science and engineering from the Department of Computer Science and Engineering, University of Dhaka, Bangladesh. He is currently pursuing the Ph.D. degree in system engineering with the University of Virginia. He was a Lecturer with the Department of Computer Science and Engineering, University of Dhaka. His research interests include multimodal learning, multitask learning, artificial intelligence, optimization, and human-centered AI.

ISMAT RAHMAN received the bachelor’s and master’s degrees in computer science and engineering from the University of Dhaka, Bangladesh, and the Ph.D. degree in operations and information management system from Aston University, U.K., in 2019. She is currently an Associate Professor with the Department of Computer Science and Engineering, University of Dhaka. She is also working as a Supervisor with the Cognitive Agents and Interaction Laboratory (CAIL) Research Group, Computer Science and Engineering Department. Her work focuses specifically on cloud computing, multi-agent systems, block chaining, distributed problem solving, causal inference, and the Internet of Things.

MD. MOSADDEK KHAN received the bachelor’s and master’s degrees in computer science and engineering from the University of Dhaka, Bangladesh, and the Ph.D. degree in computer science from Southampton University, U.K., in 2018. He is currently an Assistant Professor with the University of Dhaka, where he directs the Research Group on Cognitive Agents and Interaction Laboratory (CAIL). His research interests include multi-agent systems, distributed problem solving in artificial intelligence, causal inference, and multi-agent deep reinforcement learning. He has also served as a Regular Reviewer for a number of leading AI conferences and journals, including AAAI, IJCAI, AAMAS, AIJ, JAIR, and TOIT.

TARIQ IQBAL received the B.S. degree in computer science and engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, the M.S. degree in computer science from The University of Texas at El Paso, El Paso, TX, USA, and the Ph.D. degree in computer science from the University of California at San Diego, La Jolla, CA, USA. He was a Postdoctoral Associate with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently an Assistant Professor in systems engineering with the University of Virginia, Charlottesville, VA, USA, and directs the Collaborative Robotics Laboratory. His research interests include artificial intelligence, machine learning, and robotics. He serves on the Editorial Board of the ACM Transactions on Human-Robot Interaction and the program committees of several conferences.

* * *