Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem

Max Barer ¹ and Guni Sharon and Roni Stern and Ariel felner

1 Introduction

A multi-agent path finding (MAPF) problem is defined by a graph, \( G = (V, E) \), and a set of \( k \) agents labeled \( a_1 \ldots a_k \), where each agent \( a_i \) has a start position \( s_i \in V \) and goal position \( g_i \in V \). At each time step an agent can either move to an adjacent location or wait in its current location. The task is to plan a sequence of move/wait actions for each agent \( a_i \) moving it from \( s_i \) to \( g_i \) while avoiding conflicts with other agents (i.e., without occupying the same location at the same time) and minimizing a cumulative cost function.

Conflict-Based Search (CBS) \([2]\) is a two-level algorithm for solving MAPF problems optimally. The high level imposes constraints on the individual agents in order to find a conflict free set of paths. The low level searches for a single agent path that is consistent with the constraints imposed by the high level. In this paper we present several CBS-based unbounded- and bounded-suboptimal (where a bound on the quality is given) MAPF solvers which relax the high- and/or the low-level searches, allowing them to return suboptimal solution. We then present experimental results that show the benefits of our new approaches.

2 The Conflict Based Search (CBS) Algorithm

A sequence of single agent wait/move actions leading an agent from \( s_i \) to \( g_i \) is referred to as a path, and we use the term solution to refer to a set of \( k \) paths, one for each agent. A conflict between two paths is a tuple \( \langle a_i, a_j, v, t \rangle \) where agent \( a_i \) and agent \( a_j \) are planned to occupy vertex \( v \) at time \( t \). We define the cost of a path as the number of actions in it (including wait), and the cost of a solution as the sum of the costs of its constituent paths. A solution is valid if it is conflict-free. A constraint for agent \( a_i \) is a tuple \( \langle a_i, v, t \rangle \) where agent \( a_i \) is prohibited from occupying vertex \( v \) at time \( t \). A consistent path for agent \( a_i \) is a path that satisfies all of \( a_i \)'s constraints, and a consistent solution is a solution composed of only consistent paths.

Note that a consistent solution can be invalid if despite the fact that the paths are consistent with the individual agent constraints, they still have inter-agent conflicts.

CBS works in two levels. At the high-level, CBS searches a binary tree called the constraint tree (CT). Each node \( N \) in the CT contains:

1. A set of constraints, \( N\text{-constraints} \), imposed on each agent.
2. A solution, \( N\text{-solution} \). A single consistent solution, i.e., one path for each agent that is consistent with \( N\text{-constraints} \).
3. The total cost, \( N\text{-cost} \). The cost of the current solution.

The root of the CT contains an empty set of constraints. A successor of a node in the CT inherits the constraints of the parent and adds a single new constraint for a single agent. \( N\text{-solution} \) is found by the low-level search described below. A CT node \( N \) is a goal node when \( N\text{-solution} \) is valid, i.e., the set of paths for all agents have no conflicts. The high-level of CBS performs a best-first search on the CT where nodes are ordered by their costs.

Processing a node in the CT: Given a CT node \( N \), the low-level search is invoked for individual agents to return an optimal path that is consistent with their individual constraints in \( N \). Any optimal single-agent path-finding algorithm can be used by the low level of CBS. We used A* with the true shortest distance heuristic (ignoring constraints). Once a consistent path has been found (by the low level) for each agent, these paths are validated with respect to the other agents by simulating the movement of the agents along their planned paths (\( N\text{-solution} \)). If all agents reach their goal without any conflict, this CT node \( N \) is declared as the goal node, and \( N\text{-solution} \) is returned. If, however, while performing the validation a conflict, \( \langle a_i, a_j, v, t \rangle \), is found for two (or more) agents \( a_i \) and \( a_j \), the validation halts and the node is declared as non-goal.

Resolving a conflict: Given a non-goal CT node, \( N \), whose solution, \( N\text{-solution} \), includes a conflict, \( \langle a_i, a_j, v, t \rangle \), we know that in any valid solution at most one of the conflicting agents, \( a_i \) or \( a_j \), may occupy vertex \( v \) at time \( t \). Therefore, at least one of the constraints, \( \langle a_i, v, t \rangle \) or \( \langle a_j, v, t \rangle \), must hold. Consequently, CBS generates two new CT nodes as children of \( N \), each adding one of these constraints to the previously set of constraints, \( N\text{-constraints} \).

3 Greedy-CBS (GCBS): Suboptimal CBS

To guarantee optimality, both the high- and the low-level of CBS run an optimal best-first search: the low level searches for an optimal single-agent path that is consistent with the given agent’s constraints, and the high level searches for the lowest cost CT goal node. Greedy CBS (GCBS) uses the same framework of CBS but allows a more flexible search in both the high- and/or the low-level, preferring to expand nodes that are more likely to produce a valid (yet possibly suboptimal) solution fast.

Relaxing the High-Level: The main idea in GCBS is to prefer to expand CT nodes that seems closer to a goal node (in terms of depth in the CT). We developed a number of conflict heuristics that enables to prefer “less conflicting” CT nodes which are more likely to lead to a goal node. We designate the best one by \( h_c \). \( h_c \) counts the number of pairs of agents (out of \( \binom{k}{2} \)) that have at least one conflict within the pair. \( h_c \) chooses the CT node with the minimal count.

Relaxing the Low-Level: GCBS relaxes the low-level by giving preferences to a single-agent path that is involved in less conflicts with paths of other agents. The number of conflicts may be counted in several ways (again \( h_c \) was the best evaluation method). Note that
the path returned by the low level must be consistent but, unlike CBS, it may be suboptimal.

GCBS has the flexibility of using $h_c$ for either the high-level or the low-level or for both. This last variant, designated by GCBS-HL turned out to be the best.

### 3.1 Bounded suboptimal CBS

To obtain a bounded suboptimal variant of CBS we can implement both levels of CBS by focal search. Focal search maintains two lists of nodes: OPEN and FOCAL. OPEN is the regular OPEN-list of $A^*$. FOCAL contains a subset of nodes from OPEN. Focal search uses two arbitrary functions $f_1$ and $f_2$. $f_1$ defines which nodes are in FOCAL, as follows. Let $f_{min}$ be the minimal $f_1$ value in OPEN. Given a suboptimality factor $w$, FOCAL contains all nodes $n$ in OPEN for which $f_1(n) \leq w \cdot f_{min}$, $f_2$ is used to choose which node from FOCAL to expand. We denote this as focal-search($f_1$, $f_2$).

**High level focal search:** apply focal-search($g$, $h_c$) to search the CT, where $g(n)$ is the cost of the CT node $n$, and $h_c(n)$ is the conflict heuristic described above.

**Low level focal search:** apply focal-search($f$, $h_c$) to find a consistent path for agent $a_i$, where $f(n)$ is the regular $f(n) = g(n) + h(n)$ of $A^*$, and $h_c(n)$ is the conflict heuristic described above, considering the partial path up to node $n$ for $a_i$.

We use the term $BCBS(w_H, w_L)$ to denote CBS using a high level focal search with $w_H$ and a low level focal search with $w_L$. $BCBS(w_H, 1)$ and $BCBS(1, w_L)$ are special cases of $BCBS(w_H, w_L)$ where focal search is only used for the high or low level. In addition, GCBS is $BCBS(\infty, \infty)$. For any $w_H, w_L \geq 1$, the cost of the solution returned by $BCBS(w_H, w_L)$ is at most $w_H \cdot w_L \cdot C^*$, where $C^*$ is the cost of the optimal solution.

### 3.2 Enhanced CBS

ECBS runs the same low level CBS as $BCBS(1, w)$. Let OPEN$_i$ denote the OPEN used in the low level when searching for a path for agent $a_i$. The minimal $f$ value in OPEN$_i$, denoted by $f_{min}(i)$ is a lower bound on the cost of the optimal consistent path for $a_i$ (for the current CT node). For a CT node $n$, let $LB(n) = \sum_{i=1}^{k} f_{min}(i)$. It is easy to see that $LB(n) \leq cost(n) \leq LB(n) \cdot w$.

In ECBS, for every generated CT node $n$, the low level returns two values to the high level: (1) $cost(n)$ and (2) $LB(n)$. Let $LB = \min(LB(n) | n \in OPEN)$ where OPEN refers to OPEN of the high level. Clearly, $LB$ is a lower bound on the optimal solution of the entire problem ($C^*$). FOCAL in ECBS is defined with respect to LB and $cost(n)$ as follows:

$$FOCAL = \{ n | n \in OPEN, cost(n) \leq LB \cdot w \}$$

Since $LB$ is a lower bound on $C^*$, all nodes in FOCAL have costs that are within $w$ from the optimal solution. Thus, once a solution is found it is guaranteed to have cost that is at most $w \cdot C^*$.

### 4 Experimental results

We experimentally compared our CBS-based bounded suboptimal solvers on a range of suboptimality bounds ($w$) and domains. Specifically, for every value of $w$ we run experiments on (1) $BCBS(w, 1)$, (2) $BCBS(1, w)$, (3) $BCBS(\sqrt{w}, \sqrt{w})$, and (4) $ECBS$($w$). We also added CBS (=$BCBS(1, 1)$) as a baseline.

The success rate on a $32 \times 32$ grid are shown in Figure 1. The most evident observation is that ECBS outperforms all the other variants. This is reasonable as having $w$ shared among the low and high level allows ECBS to be more flexible than the static distribution of $w$ to $w_L$ and $w_H$ used by the different CBS variants. We also compared ECBS to other bounded suboptimal search algorithms that are based on $A^*$; results are omitted. ECBS tends to outperform these algorithms in most of our settings but not in all of them.

We also compared GCBS-HL with parallel push and swap (PPS) [1] and MSG1 [5], which are state-of-the-art unbounded suboptimal solvers. PPS was significantly faster than GCBS-HL and MSG1 but returned solutions that were far from optimal, and up to 5 times larger than the solution returned by GCBS-HL. Thus, if a solution is needed as fast as possible and its cost is of no importance then PPS, as a fast rule-based algorithm, should be chosen. The cost of the solutions returned by GCBS-HL and MSG1 were almost identical.

On a $5 \times 5$ grid GCBS-HL outperforms MSG1 as shown in Figure 2. In a $32 \times 32$ grid MSG1 outperforms GCBS-HL. They were both equal on game maps (not shown). There is no universal winner here as was also observed for MAPF optimal solvers [4, 2, 3]. Fully identifying which algorithm works best under what circumstances is a challenge for future work. In addition, other optimal MAPF algorithms can be modified to their suboptimal counterparts.

**Acknowledgments:** this research was supported by the Israeli Science Foundation under grant #417/13 to Ariel Felner.

**REFERENCES**


