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

Max Barer
ISE Department
Ben-Gurion University
Israel
max.barer@gmail.com

Guni Sharon
ISE Department
Ben-Gurion University
Israel
gunisharon@gmail.com

Roni Stern
ISE Department
Ben-Gurion University
Israel
roni.stern@gmail.com

Ariel Felner
ISE Department
Ben-Gurion University
Israel
felner@bgu.ac.il

Abstract

The task in the *multi-agent path finding* problem (MAPF) is to find paths for multiple agents, each with a different start and goal position, such that agents do not collide. A successful optimal MAPF solver is the *conflict-based search* (CBS) algorithm. CBS is a two level algorithm where special conditions ensure it returns the optimal solution. Solving MAPF optimally is proven to be NP-hard, hence CBS and all other optimal solvers do not scale up. We propose several ways to relax the optimality conditions of CBS trading solution quality for runtime as well as bounded-suboptimal variants, where the returned solution is guaranteed to be within a constant factor from optimal solution cost. Experimental results show the benefits of our new approach; a massive reduction in running time is presented while sacrificing a minor loss in solution quality. Our new algorithms outperform other existing algorithms in most of the cases.

Introduction

A *multi-agent path finding* (MAPF) problem is defined by a graph, $G = (V, E)$, and a set of k agents labeled $a_1 \dots a_k$, 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, both actions cost 1.0. 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) while aiming to minimize a cumulative cost function. MAPF has practical applications in video games, traffic control (Silver 2005; Dresner and Stone 2008), robotics (Bennewitz, Burgard, and Thrun 2002) and aviation (Pallottino et al. 2007).

Solving MAPF optimally (i.e., finding a conflict-free solution of minimal cost) is NP-Complete (Yu and LaValle 2013b). Therefore, optimal MAPF solvers suffer from a scalability problem. Suboptimal MAPF solvers run relatively fast but have no guarantee on the quality of the returned solution and some of them are not complete. *Bounded suboptimal* algorithms guarantee a solution which is no larger than a given constant factor over the optimal solution cost (Wagner and Choset 2011).

The *Conflict-Based Search* (CBS) algorithm (Sharon et al. 2012a) is a two-level algorithm for solving MAPF problems optimally. The high level searches for a set of constraints to impose on the individual agents to ensure finding an optimal solution. The low level searches for optimal solutions to single agent problems under the constraints imposed by the high level.

In this paper we present several CBS-based unbounded- and bounded-suboptimal MAPF solvers which relax the high- and/or the low-level searches, allowing them to return suboptimal solution. First we introduce *Greedy-CBS* (GCBS), a CBS-based MAPF solver designed for finding a (possibly suboptimal) solution as fast as possible. Then, we propose *Bounded CBS* (BCBS) that uses a focal-list mechanism in both the low- and high-level of CBS. This ensures that the returned solution is within a given suboptimality bound (which is the product of the bounds of the two levels). Finally, we present *Enhanced CBS* (ECBS), a bounded suboptimal MAPF solver in which the high- and low-levels share a joint suboptimality bound.

Experimental results show substantial speedup over CBS for all variants. For example, in one of the domains (DAO) CBS was able to solve problems of up to 50 agents, while GBCS solved all problems with 250 agents, and ECBS solved more than half of the problems with 200 agents while guaranteeing a solution that is at most 1% worse than the optimal solution. Comparing against other suboptimal MAPF solvers, we observe that ECBS almost always outperforms alternative bounded suboptimal solvers, while GCBS performed best in many domains but not in all of them.

Terminology and Background

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 point t.[1] We define the *cost* of a path as the number of actions in it, 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.

[1]Based on the setting, a conflict may also occur over an edge, when two agents traverse the same edge in opposite directions.

MAPF algorithms for finding valid solutions can be classified into three classes: *optimal* solvers, *suboptimal* solvers and *bounded suboptimal* solvers. We cover each of them next. A more detailed survey of these and other MAPF algorithms can be found in (Sharon et al. 2013).

Optimal solvers

A number of MAPF optimal solvers that are based on the A* algorithm have been introduced. The state-space includes all permutations of placing k agents in V locations ($= \binom{V}{k} = O(V^k)$). Standley (Standley 2010) presented the *independence detection* (ID) framework. ID partitions the problem into smaller independent subproblems, if possible, and solves each problem separately. Standley also introduced *operator decomposition* (OD) which considers moving a single agent at a time. OD reduces the branching factor at the cost of increasing the depth of the solution in the search tree. Another relevant A* variant is *Enhanced Partial Expansion A** (EPEA*) (Felner et al. 2012). EPEA* uses *a priori* domain knowledge to sort all successors of a given state according to their f values. When a state is expanded only successors that are *surely needed* by A* (those with $f \leq C^*$) are generated.

M* (Wagner and Choset 2011) is an A*-based algorithm that dynamically changes the branching factor based on conflicts. In general, nodes are expanded with only one child where each agent makes its optimal move towards the goal. When conflicts occur between q agents, the branching factor is increased to capture all internal possibilities. An enhanced version, called *Recursive M** (RM*) divides the q conflicting agents into subgroups of agents each with independent conflicts. Then, RM* is called recursively on each of these groups. A variant called ODRM* (Ferner, Wagner, and Choset 2013) combines Standley's OD on top of RM*.

Recently, two optimal MAPF search algorithms were proposed that are not based on A*. **(1)** The *increasing cost tree search* (ICTS) algorithm solves MAPF optimally by converting it into a set of fast-to-solve decision problems (Sharon et al. 2013). **(2)** The *conflict based search* (CBS) (Sharon et al. 2012a) and its extension *meta-agent CBS* (MA-CBS)(Sharon et al. 2012b). CBS is the main focus of this paper and is detailed below.

Another approach for solving MAPF that was taken recently is to reduce it to other problems that are well studied in computer science. Prominent examples include reducing the MAPF problem to Boolean Satisfiability (SAT) (Surynek 2012), Integer Linear Programming (ILP) (Yu and LaValle 2013a) and Answer Set Programming (ASP) (Erdem et al. 2013). These methods are usually designed for the *makespan* cost function. They are less efficient or even not applicable for the sum of cost function. In addition, these algorithms are usually highly efficient only on small graphs. On large graphs the translation process from a MAPF instance to the required problem has a large overhead. Adapting these algorithms to be bounded/suboptimal for the sum of cost function is not trivial.

Unbounded Suboptimal Solvers

Many existing MAPF solvers aim at finding a solution fast while allowing the returned solution to be suboptimal. Most suboptimal MAPF solvers are *unbounded*, i.e., they do not provide any guarantee on the quality of the returned path.

Early work presented a theoretical polynomial-time algorithm for finding valid solutions to MAPF (Kornhauser, Miller, and Spirakis 1984). This algorithm is complete for any MAPF problem. However, the solution it returns can be far from optimal and the returned plan moves agents subsequently, i.e. one at each time step.

Many other suboptimal solvers, that work well in practice, were presented in research and industry.

An important group of suboptimal MAPF algorithms are based on search. A prominent example is the *Cooperative A** (CA*) algorithm and its variants HCA* and WHCA* (Silver 2005). CA* and its variants plan a path for each individual agent sequentially. Every planned path is written to a global *reservation table* and subsequent agents are not allowed to plan a path that conflicts with the paths in the reservation table. Importantly, CA* and its variants are not complete. Standley and Korf (Standley and Korf 2011) proposed a complete suboptimal MAPF solver called MSG1 that is a relaxed version of Standley's A*-based algorithm mentioned above.

Another group of suboptimal algorithms are *rule based*; they include specific movement rules for different scenarios. Rule-based solvers are usually complete under some restrictions and they aim to find a valid solution fast, but the quality of the returned solution can be far from optimal. The *push and swap* (PS) algorithm (Luna and Bekris 2011) uses two "macro" operators which "push" an agent to an empty location, and "swap" the locations of two agents. PS is complete for graphs with at least two empty locations. *Parallel push and swap* (PPS) (Sajid, Luna, and Bekris 2012) is an advanced variant of PS that plans parallel routes for the different agents instead of moving only one agent at a time. [2] Hybrids of search-based and rule-based algorithms also exist (see (Sharon et al. 2013) for details).

Bounded Suboptimal Solvers

A *bounded suboptimal* search algorithm accepts a parameter w (sometimes known as $1 + \epsilon$) and returns a solution that is guaranteed to be *less than or equal to* $w \cdot C^*$, where C^* is the cost of the optimal solution. Bounded suboptimal search algorithms provide a middle ground between optimal algorithms and unbounded suboptimal algorithms. Setting different values of w allows the user to control the tradeoff between runtime and solution quality.

Several general-purpose bounded suboptimal search algorithms exist such as WA* (Pohl 1970) and others (Pearl and Kim 1982; Ghallab and Allard 1983; Thayer and Ruml 2008; 2011; Hatem, Stern, and Ruml 2013). These algorithms usually extend optimal search algorithms (e.g., A*

[2]PS and PPS were found to have a shortcoming which caused it to be incomplete in some cases. A variant of PS called *push and rotate* PR (de Wilde, ter Mors, and Witteveen 2013) was shown to solve this problem and is thus complete.

or IDA*) by considering an inflated version of an admissible heuristic. Thus, one can apply these algorithms to any A*-based MAPF solver, such as EPEA* (Felner et al. 2012), A* with OD (Standley 2010), and M* (Wagner and Choset 2011). We implemented and experimented with such bounded suboptimal MAPF solvers below.

It is not clear, however, how to modify MAPF solvers that are not based on A* to (bounded) suboptimal versions. In particular, the method of inflating the heuristic is not applicable in CBS as it does not use heuristic. In this paper we show how to extend CBS to unbounded- and bounded suboptimal MAPF algorithms. As a preliminary, we first describe the basic CBS algorithm and then move to our new unbounded- and bounded versions of CBS.

The Conflict Based Search (CBS) Algorithm

In CBS, agents are associated with constraints. 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 step t.[3] 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, constraints are generated for the different agents. At the low-level, paths for individual agents are found that are consistent with their respective constraints. If these paths conflict, new constraints are added to resolve one of the conflicts and the low-level is invoked again.

The high-level: At the high-level, CBS searches the *constraint tree* (CT). The CT is a binary tree. Each node N in the CT contains:

(1) A set of constraints ($N.constraints$), imposed on each agent.

(2) A solution ($N.solution$). A single consistent solution, i.e., one path for each agent that is consistent with $N.constraints$.

(3) The total cost ($N.cost$). 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. $N.solution$ is found by the low-level search described below. A CT node N is a goal node when $N.solution$ 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 modified for 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 ($N.solution$). If all agents reach their goal without any conflict, this CT node N is declared as the goal node, and $N.solution$ 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, $N.solution$, 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, (a_i, v, t) or (a_j, v, t), 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, $N.constraints$. Note that for each CT node (except for the root) the low-level search is only activated for one single agent – the agent for which the new constraint was added.

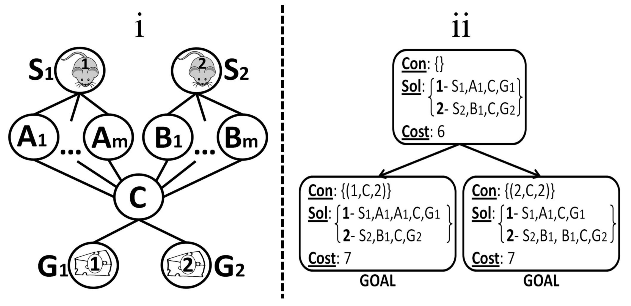


Figure 1: (i) MAPF example (ii) CT

CBS Example

Pseudo-code for CBS is shown in Algorithm 1. We cover it using the example in Figure 1(i), where the mice need to get to their respective pieces of cheese. The corresponding CT is shown in Figure 1(ii). The root contains an empty set of constraints. At the beginning, the low-level returns an optimal solution for each agent, $\langle S_1, A_1, C, G_1\rangle$ for a_1 and $\langle S_2, B_1, C, G_2\rangle$ for a_2 (line 2). Thus, the total cost of this node is 6. All this information is kept inside this node. The root is then inserted into the sorted OPEN list and will be expanded next.

When validating the two-agents solution (line 7), a conflict is found when both agents arrive to vertex C at time step 2. This creates the conflict $\langle a_1, a_2, C, 2\rangle$. As a result, the root is declared as non-goal and two children are generated in order to resolve the conflict (Line 11). The left child, adds the constraint $\langle a_1, C, 2\rangle$ while the right child adds the constraint $\langle a_2, C, 2\rangle$. The low-level search is now invoked (Line 15) for the left child to find an optimal path that also satisfies the new constraint. For this, a_1 must wait one time step either at S_1 (or at A_1) and the path $\langle S_1, A_1, A_1, C, G_1\rangle$ is returned for a_1. The path for a_2, $\langle S_2, B_1, C, G_2\rangle$ remains unchanged in the left child. The cost of the left child is now 7, where the cost is computed as the sum of the individual single-agent cost (SIC). In a similar way, the right child is generated, also with cost 7. Both children are added to

[3]Depending on the problem's setting constraints on edges are also possible.

Algorithm 1: high-level of CBS

```
Input: MAPF instance
1  $R.constraints = \emptyset$
2  $R.solution$ = find individual paths using the low-level()
3  $R.cost = SIC(R.solution)$
4  insert R to OPEN
5  while OPEN not empty do
6      P ← best node from OPEN // lowest solution cost
7      Validate the paths in P until a conflict occurs.
8      if P has no conflict then
9          return P.solution // P is goal
10     C ← first conflict $(a_i, a_j, v, t)$ in P
11     foreach agent $a_i$ in C do
12         A ← new node
13         A.constraints ← P.constraints + $(a_i, s, t)$
14         A.solution ← P.solution.
15         Update A.solution by invoking low-level($a_i$)
16         $A.cost = SIC(A.solution)$
17         Insert A to OPEN
```

OPEN (Line 17). In the final step the left child is chosen for expansion, and the underlying paths are validated. Since no conflicts exist, the left child is declared as a goal node (Line 9) and its solution is returned. CBS was proven to be both optimal and complete(See (Sharon et al. 2012a)).

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. As any best-first search, this causes extra work due to abandoning nodes which might have solutions very close below them, only because their cost is high. *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 prioritize CT nodes that seem closer to a goal node (in terms of depth in the CT, AKA *distance-to-go* (Thayer and Ruml 2011)). Every non-goal CT node contains an invalid solution that contains internal conflicts. We developed a number of *conflict heuristics* (designated as h_c) that enables to prefer "less conflicting" CT nodes which are more likely to lead to a goal node. The high-level in GCBS favors nodes with minimal conflict heuristic, i.e., it chooses the node with minimal h_c. We experimented with the following heuristics for h_c:

- h_1**: Number of conflicts:** this heuristic counts the number of conflicts that are encountered in a specific CT node.
- h_2**: Number of conflicting agents:** this heuristic counts the number of agents (out of k) that have at least one conflict.
- h_3**: Number of pairs:** this heuristic counts the number of pairs of agents (out of $\binom{k}{2}$) that have at least one conflict between them.

h_4**: Vertex cover:** we define a graph where the agents are the nodes and edges exist between agents that have at least one conflict between them. In fact, h_2 identifies the nodes and h_3 identifies the edges of this graph. h_4 computes a vertex-cover of this graph.

h_5 **Alternating heuristic:** (Röger and Helmert 2010; Thayer, Dionne, and Ruml 2011) showed that when more than one heuristic exists, better performance can be attained by alternating through the set of different heuristics in a round robin fashion. This approach is known as *Alternating Search*. It requires implementing several heuristic functions and maintaining a unique open list for each, making it more complex to encode.

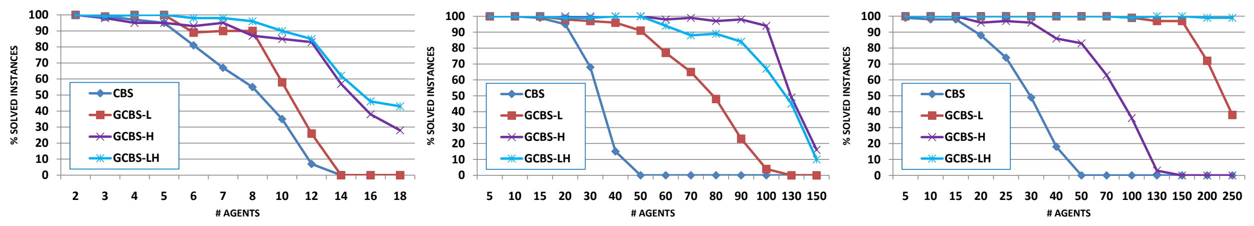
We experimented with these heuristics and found that the alternating heuristic (h_5) is the best in performance. Vertex cover heuristic (h_4) produces better quality solutions but requires a large computational overhead. Number of conflicting agents heuristic (h_2) runs fast on some problem instances but is very slow on other instances. Number of conflicts heuristic (h_1) runs slightly faster than number of pairs heuristic (h_3) on average but h_3 is more robust across different instances. Since the difference in performance between the different heuristics was minor we choose to only report h_3, because it gives the best balance between simplicity and performance. Therefore, whenever we refer to h_c we relate to h_3. Note that all the heuristics above are not admissible and not even bounded admissible. Consequently, performing a greedy search according to them will not result in an optimal or bounded solution.

Relaxing the Low Level: Another way to relax the optimality conditions of CBS is to use a sub-optimal low-level solver. One might be tempted to use WA* or greedy-BFS (which favors nodes with low h-values) for the low level. This will certainly return a solution faster than any optimal low-level solver. However, this is not enough. Such algorithms return longer paths with many new future conflicts. This may significantly increase the number of future high-level nodes and proved ineffective in our experiments.

An effective alternative is to again use conflict heuristics for the low-level. In optimal CBS, the low-level for agent a_i returns the *shortest individual path* that is consistent with all the known constraints of a_i. We suggest modifying the low-level search for agent a_i to a best-first search that prioritizes a path with minimum number of conflicts with other agents assigned path. That is, we run A* for a single agent where states that are not in conflict with other agents paths are preferred. For example, assume agent a_1 is assigned a path $\langle S_1, A, G_1 \rangle$ and a different agent, a_2, is now considered by the low-level. The A* low-level search would give preference to any location except for location A on the first step, even if optimality is sacrificed. By doing so the low-level returns a path that has the minimal number of conflicts[4] with other previously assigned agents.

Completeness of GCBS: GCBS is not optimal but is complete if an upper bound B exists on the cost of a valid

[4]Number of conflicts may refer to any of the previously defined heuristics h_1, h_2, h_3, h_4, h_5.



(a) 5 × 5 - 20% obstacles (b) 32 × 32 - 20% obstacles (c) DAO Map - BRC202

Figure 2: Greedy CBS - Success Rate. Comparison between all the greedy versions and optimal CBS

solution and assuming that GCBS will prune CT nodes with cost higher than B. As shown for CBS by Sharon et al. (Sharon et al. 2012a), any valid solution is consistent with at least one CT node in OPEN. As GCBS is a best-first search, it will eventually expand all the CT nodes with cost $\leq B$, finding any valid solution below that cost.

Greedy-CBS: Experiments

We experimented on many types of grids and maps. We report results on the following domains which are good representatives of the other domains:

- **4-connected grids with 20% obstacles -** Grids of size 5×5 and 32×32 were used.
- **Dragon Age Origins (DAO) Maps -** we experimented with many maps from Sturtevant's repository (Sturtevant 2012) but report results on the BRC202D map which has relatively many conflicts due to corridors and bottlenecks. The trends reported below were also observed for other maps.

Figure 2 compares three variants of GCBS to the original CBS:

1. *GCBS-H* which uses h_c for choosing CT nodes at the high level but the low level runs ordinary A*.
2. *GCBS-L* which, similar to the original CBS, uses the g-values for choosing CT nodes at the high level but the low level uses h_c for its expansions.
3. *GCBS-HL* which uses h_c for both levels.

Figure 2 shows the percentage of instances solved within a time limit of 5 minutes (similar to (Sharon et al. 2013; Standley 2010)) as a function of the number of agents.

Clearly, optimal CBS is the weakest. GCBS-HL was the best for the 5×5 grid and for the DAO map, while GCBS-H slightly outperformed it for the 32×32 grid. The relative performance of GCBS-H and GBCS-L varies on these domains. The 5×5 grid is very dense with agents and there are many conflicts. The low level cannot find a solution for a single agent that avoids this large set of time-space dense conflicts. The high level has a more global view and it directs the search towards areas with less conflicts. By contrast, the DAO maps are larger and even when many conflicts exist they are distributed both in time and in space. Therefore it is easier for the low level to bypass conflicts. GBCS-HL was weak in the 32×32 grid because its low-level may return long paths in order to avoid conflicts. This further constrains the motion of other agents and may cause more highl-level conflicts. In general, GCBS-LH is more robust than GCBS-H and GCBS-L, and we recommend it for general usage.

Domain	Agnts	Inst.	CBS	GCBS-L	GCBS-H	GCBS-LH
5 × 5	8	48	37	40	41	45
32×32	30	67	669	675	672	675
DAO	30	49	12,067	12,071	12,072	12,071

Table 1: Comparison of cost

Table 1 shows the average solution cost over all instances (column 3) that were solved by all variants. GCBS-H and GCBS-L tend to provide solutions within 10% of optimal. For the 5 × 5 grid GCBH-HL provides longer solutions because it relaxes the optimality conditions of both levels. Since the DAO maps and the 32×32 grids are less dense, all variants provided solutions within 5% of optimal.

Bounded Suboptimal CBS

A commonly used and general bounded-suboptimal search algorithm is *Weighted A** (WA*) (Pohl 1970). Since the high-level of CBS does not use heuristic guidance WA* is not applicable to CBS. We thus use *Focal search*, an alternative approach to obtain bounded suboptimal search algorithms, based on the A^*_ϵ (Pearl and Kim 1982) and A_ϵ (Ghallab and Allard 1983) algorithms.

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 $f_{1_{min}}$ be the minimal f_1 value in OPEN. Given a suboptimality factor w, FOCAL contains all nodes n in OPEN for which $f_1(n) \leq w \cdot f_{1_{min}}$. f_2 is used to choose which node from FOCAL to expand. We denote this as *focal-search*(f_1,f_2). If f_1 is admissible then we are guaranteed that the returned solution is at most $w \cdot C^*$.[5]

Importantly, f_2 is not restricted to measure cost and it can use other relevant measures. For example, *explicit estima-*

[5] A^*_ϵ(Pearl and Kim 1982) can be written as focal-search(f,h), that is, A^*_ϵ chooses to expand the node in FOCAL with the minimal h-value.

tion search (EES) (Thayer and Ruml 2011) considers an estimation of the *distance-to-go*, d (i.e., the number of hops to the goal) as well as an inadmissible heuristic.

BCBS

To obtain a bounded suboptimal variant of CBS we can implement both levels of CBS as a focal search:

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 single agent path, 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 the agent that the low level is planning for.

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. $BCBS(w, 1)$ and $BCBS(1, w)$ are special cases of $BCBS(w_H, w_H)$ where focal search is only used for the high or low level. In addition, GCBS is a special case that uses $w = \infty$ for one or both levels, i.e., $BCBS(\infty, \infty)$ is GCBS-LH. In this case, FOCAL is identical to OPEN.

Theorem 1 *For any $w_H, w_L \geq 1$, the cost of the solution returned by $BCBS(w_H, w_L)$ is at most $w_H \cdot w_L \cdot C^*$*

Proof: Let N be a CT node expanded by the high level. Let $cost^*(N)$ denote the sum of the lowest cost path for each agent to its goal, under the constraints in N, and let C^* be the cost of the optimal solution. Until a goal is found, C^* is larger than or equal to $\min_{x \in OPEN} cost^*(x)$, and $N.cost \leq w_L \cdot cost^*(N)$, since the low level solver used a focal search with w_L. N is a member of FOCAL. Therefore:

$$N.cost \leq w_H \cdot \min_{x \in OPEN} x.cost \tag{1}$$

$$\frac{N.cost}{w_L} \leq w_H \cdot \min_{x \in OPEN} \frac{x.cost}{w_L} \tag{2}$$

$$\leq w_H \cdot \min_{x \in OPEN} cost^*(x) \leq w_H \cdot C^* \tag{3}$$

$$N.cost \leq w_L \cdot w_H \cdot C^* \tag{4}$$

$$\tag{5}$$

Thus, when expanding a node N, $N.cost$ is guaranteed to be at most $w_L \cdot w_H \cdot C^*$ $\square$

Based on Theorem 1, to find a solution that is guaranteed to be at most $w \cdot C^*$ one can run $BCBS(w_H, w_L)$ for any values of w_H and w_L for which $w_H \cdot w_L = w$.

Enhanced CBS

How to distribute w between w_H and w_L is not trivial. The best performing distribution is domain dependent. Furthermore, BCBS fixes w_H and w_L throughout the search; this might be inefficient. To address these issues we propose *Enhanced CBS* (ECBS). ECBS runs the same low level search as $BCBS(1, w)$. Let OPEN$_i$ denote the OPEN used in CBS's 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 $LB(n) \leq n.cost \leq LB(n) \cdot w$.

In ECBS, for every generated CT node n, the low level returns two values to the high level: **(1)** $n.cost$ 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 $n.cost$ as follows:

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

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 $w \cdot C^*$.

The advantage of ECBS over BCBS is that while allowing the low level the same flexibility as $BCBS(1, w)$, it provides additional flexibility in the high level when the low level finds low cost solutions (i.e, when $LB(n)$ is close to $n.cost$). This theoretical advantage is also observed in practice in the experimental results section below.

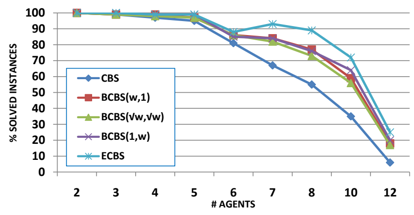
Both BCBS and ECBS never expand nodes with cost higher than w times the optimal solution. In addition, all valid solutions are always consistent with at least one of the CT nodes in OPEN. As such, and since both are systematic searches, they will eventually find a solution if such exists. Thus, BCBS and ECBS are complete.

Experimental results

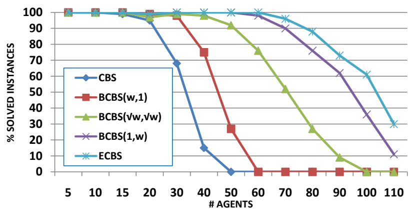
Next, we experimentally compare our CBS-based bounded suboptimal solvers on a range of suboptimality bounds (w) and domains. Specifically, for every value of w we run experiments on **(1)** $BCBS(w, 1)$, **(2)** $BCBS(1, w)$, **(3)** $BCBS(\sqrt{w}, \sqrt{w})$, and **(4)** $ECBS(w)$. We also added CBS (=$BCBS(1, 1)$) as a baseline. Different w values are presented for the different domains, as the impact of w varies greatly between domains. For example, extremely small w values allowed faster solution times in DAO, while in smaller domains only larger w values had a substantial impact.

The success rates on the same three domains are shown in Figure 3. 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 BCBS variants.

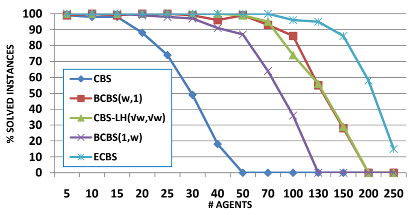
Comparing the performance of the different BCBS versions ($BCBS(w, 1)$, $BCBS(1, w)$, and $BCBS(\sqrt{w}, \sqrt{w})$) provides an insight into the effect of w on the different CBS levels. Setting w in the high level ($BCBS(1, w)$) performed best in the 5×5 and 32×32 grid, while setting w in the low level ($BCBS(w, 1)$) performed best in the DAO map. We explain this by considering the properties of the different domains. The 5×5 and 32×32 grids are substantially smaller than the large DAO map. Thus the paths found by the agents are longer. On the other hand, the DAO maps are less dense, and thus less conflicts occur. When agents have longer paths and conflicts are rare,



(a) 5×5 - 20% obstacles. $w = 1.5$

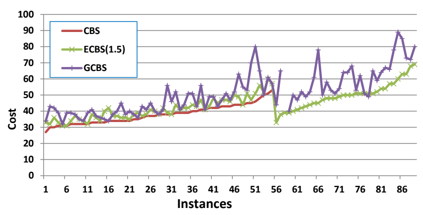


(b) 32×32 - 20% obstacles. $w = 1.1$

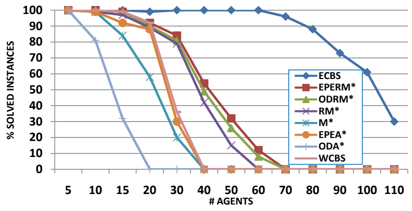


(c) DAO Map - BRC202. $w = 1.01$

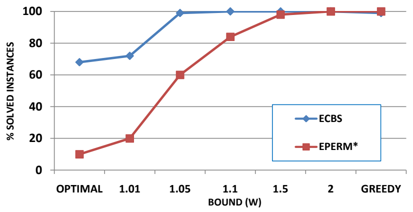
Figure 3: Success rate of Bounded CBS versions.



(a) Cost comparison, 8 agents, $w = 1.5$



(b) Bounded algorithms, $w = 1.1$



(c) Different w values

Figure 4: Various results comparing ECBS to other bounded suboptimal algorithms

adding w to the low level allows it to circumvent potential conflicts. By contrast, in smaller and denser maps avoiding conflicts is harder and therefore the low level is not able to avoid all conflicts even if allowed to take a longer path. In such cases, w would be more effective in the high level, allowing it to ignore more conflicts.

In our experiments, we observed that GCBS often returns low cost solutions even though its solution quality is theoretically unbounded. The difference, on average, between the costs of the solution found by GCBS and by ECBS were often small, in our domains. Figure 4(a) shows the costs of the solution returned by CBS, GCBS and ECBS with $w = 1.5$ on 89 instances of the 5×5 grid with 8 agents. Every data point represents the cost of the solution to a single 5×5 instance. The instances are ordered according to the optimal solution found by CBS. Instances 56-89 were not solved by CBS, and are ordered according to the solution cost found by ECBS. While GCBS often returned close to optimal results, it had a larger variance compared to the bounded solver ECBS. This illustrates the choice to be made by an application designer requiring a MAPF solver. If the main purpose is runtime and occasional high cost solution is tolerable, then GCBS would be the algorithm of choice, as it is often faster than ECBS. However, if stability, reliability and guaranteeing the bound is important – ECBS provides a more appropriate solver.

Comparison to other MAPF solvers

Based on our experiments, ECBS is the best bounded suboptimal MAPF solver among the CBS family. Next, we compare ECBS to other bounded suboptimal MAPF solvers that are not based on CBS. To our knowledge, the only bounded suboptimal MAPF solvers previously proposed are M* variants: M*, RM* (Wagner and Choset 2011) and ODRM* (Ferner, Wagner, and Choset 2013). Moreover, we combined RM* with EPEA* (Felner et al. 2012), which is currently the best A*-based MAPF solver. This is denoted by EPERM*. In addition, we implemented several adaptations of weighted A* to a bounded suboptimal MAPF solver. We used ODA* (A*+OD) and EPEA* algorithms, where node ordering is determined using the $g + w \cdot h$ evaluation function instead of the regular $g+h$. Another one is WCBS* which is CBS using WA* in the low level.

Figure 4(b) shows the success rate for all these algorithms as a function of the number of participating agents on the 32×32 grid and $w = 1.1$. The results show a clear advantage of ECBS over all other bounded suboptimal solvers. Similar results were obtained in the DAO map and 5×5 grid, except for large w values in the DAO map, where EPERM* and ECBS performed similarly. We omit these results due to space constraints.

Next, we evaluated the effect of w on ECBS and EPERM*, which were the two best bounded suboptimal algorithms. Figure 4(c) shows the results for 32×32 grid with 30 agents. As can be seen, ECBS is able to solve more instances consistently, until reaching $w \geq 1.5$, where both algorithms perform similarly.

In most settings ECBS performed best. However, there are many parameters, problems types and settings where other algorithms might prevail. We leave deeper analysis for future work.

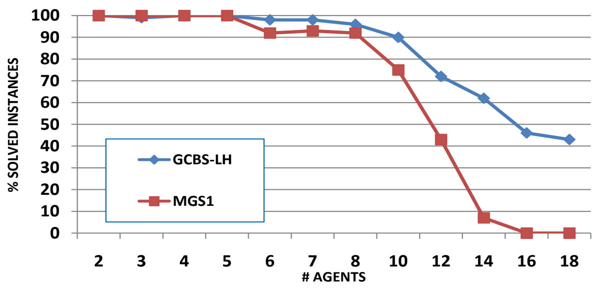


Figure 5: 5 × 5 grid, 20% obstacles, success rate

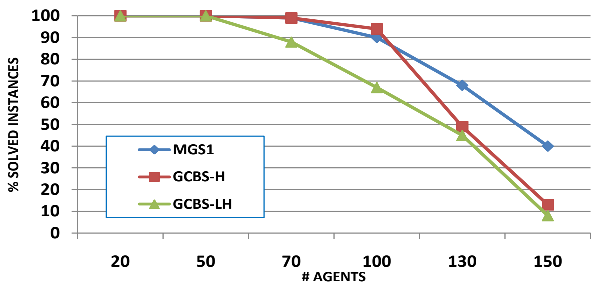


Figure 6: 32 × 32 grid, 20% obstacles, success rate

Comparison to unbounded suboptimal solvers.
Next, we compared GCBS-HL with *parallel push and swap*(PPS) (Sajid, Luna, and Bekris 2012) and Standley's MGS1 (Standley and Korf 2011), which are state-of-the-art unbounded suboptimal solvers. PPS was significantly faster than GCBS-HL and MGS1 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 MGS1 were almost identical (GCBS costs was reported in Table 1).

Figures 5 and 6 show the success rate of GCBS-HL and MSG1 on the 5 × 5 and 32 × 32 grids. In the 5 × 5 grid, GCBS-HL outperforms MSG1, while in the 32 × 32 grid MSG1 outperforms GBCS-HL. For the 32 × 32 we also experimented with GCBS-H, which was shown to be effective in this domain (see Figure 3). Indeed, here, GCBS-H is better than GCBS-HL, but it is outperformed by MSG1 on problems with more than 100 agents. We also compared GCBS and MSG1 in the DAO map. There, both algorithms were able to solve all instances up to 250 agents (under 5 minutes). Further identifying which unbounded MAPF algorithm performs best in which domain and performing DAO experiments with more agents are left for future work.

Conclusions

We presented a range of unbounded and bounded suboptimal MAPF solvers based on the CBS algorithm. Our bounded suboptimal solvers, BCBS and ECBS, use a focal list in potentially both of CBS's levels. We proposed a heuristic to estimate which state in the focal would lead to a solution fast. Experimental results show several orders of magnitude speedups while still guaranteeing a bound on solution quality. ECBS clearly outperforms all other bounded suboptimal solvers. While PPS was the fastest unbounded suboptimal search, its solution cost is much higher. GCBS returned close to optimal solutions, and outperformed MSG1 in some domains. In other domains, MSG1 was better, demonstrating that there is no universal winner. This is a known phenomenon for MAPF on optimal solvers too (Sharon et al. 2013; 2012a; 2012b). Fully identifying which algorithm works best under what circumstances is a challenge for future work. In addition, in this paper we focused on the modification of CBS to its suboptimal variants. Similar treatment should be given in the future to other known optimal solvers such as ICTS, MA-CBS and the other solvers that are based on SAT, ILP and ASP. In fact, such non-search methods are probably relevant to other search problems and are not limited to MAPF. Time will tell how these new methods compare in general to traditional search approaches.

Acknowledgments

This research was supported by the Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Bennewitz, M.; Burgard, W.; and Thrun, S. 2002. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems* 41(2-3):89–99.

de Wilde, B.; ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 87–94. International Foundation for Autonomous Agents and Multiagent Systems.

Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *JAIR* 31:591–656.

Erdem, E.; Kisa, D. G.; Oztok, U.; and Schueller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *Proc. of AAAI*.

Felner, A.; Goldenberg, M.; Sturtevant, N.; Stern, R.; Sharon, G.; Beja, T.; Holte, R.; and Schaeffer, J. 2012. Partial-expansion A* with selective node generation. In *AAAI*.

Ferner, C.; Wagner, G.; and Choset, H. 2013. ODrM* optimal multirobot path planning in low dimensional search spaces. In *ICRA*, 3854–3859.

Ghallab, M., and Allard, D. G. 1983. Aean efficient near admissible heuristic search algorithm. In *Proc. 8th IJCAI, Karlsruhe, Germany*, 789–791.

Hatem, M.; Stern, R.; and Ruml, W. 2013. Bounded suboptimal heuristic search in linear space. In *SOCS*.

Kornhauser, D.; Miller, G.; and Spirakis, P. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS*, 241–250. IEEE.

Luna, R., and Bekris, K. E. 2011. Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, 294–300.

Pallottino, L.; Scordio, V. G.; Bicchi, A.; and Frazzoli, E. 2007. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics* 23(6):1170–1183.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4:392–400.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3):193–204.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. *Alternation* 10(100s):1000s.

Sajid, Q.; Luna, R.; and Bekris, K. 2012. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *SOCS*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012a. Conflict-based search for optimal multi-agent path finding. In *AAAI*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012b. Meta-agent conflict-based search for optimal multi-agent path finding. In *SoCS*.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.* 195:470–495.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

Standley, T. S., and Korf, R. E. 2011. Complete algorithms for cooperative pathfinding problems. In *IJCAI*, 668–673.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144–148.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI 2012: Trends in Artificial Intelligence*. Springer. 564–576.

Thayer, J., and Ruml, W. 2008. Faster than weighted A*: An optimistic approach to bounded suboptimal search. In *ICAPS*, 355–362.

Thayer, J., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, 674–679.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *ICAPS*.

Wagner, G., and Choset, H. 2011. M*: A complete multi-robot path planning algorithm with performance bounds. In *IROS*, 3260–3267.

Yu, J., and LaValle, S. M. 2013a. Planning optimal paths for multiple robots on graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 3612–3617. IEEE.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*.