ICBS: Improved Conflict-based Search algorithm for Multi-Agent Pathfinding

Eli Boyarski
CS Department
Bar-Ilan University (Israel)
eli.boyarski@gmail.com

Ariel Felner
Roni Stern
Guni Sharon
ISE Department
Ben-Gurion University
felner@bgu.ac.il

David Tolpin
Oded Betzalel
Eyal Shimony
CS Department
Ben-Gurion University
shimony@cs.bgu.ac.il

Abstract

Conflict-Based Search (CBS) and its enhancements, *Meta-Agent CBS* and *bypassing conflicts* are amongst the strongest newly introduced algorithms for Multi-Agent Path Finding. This paper introduces two new improvements to CBS and incorporates them into a coherent, improved version of CBS, namely ICBS. Experimental results show that each of these improvements further reduces the runtime over the existing CBS-based approaches. When all improvements are combined, an even larger improvement is achieved, producing state-of-the art results for a number of domains.

1 Introduction and Overview

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 a 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 such that agents do not *conflict*, i.e., occupy the same location at the same time, while aiming to minimize a cumulative cost function. MAPF has practical applications in video games, traffic control, robotics etc. (see Sharon et al. 2013; 2015 for a survey).

In this paper we focus on solving MAPF problems *optimally*, i.e., where the cost of the resulting plan must be minimized. There is a range of algorithms that optimally solve different variants of MAPF using various search techniques [Standley, 2010; Wagner and Choset, 2011; Sharon *et al.*, 2013] or by compiling it into other known NP-hard problems [Surynek, 2012; Yu and LaValle, 2013; Erdem *et al.*, 2013]. Each of these solvers has pros and cons, with no universal winner. Which algorithm performs best under what circumstances is an open research question.

Conflict-Based Search (CBS) [Sharon *et al.*, 2012a; 2015], is a very effective optimal MAPF solver. CBS has two-levels. The low-level finds optimal paths for the individual agents. If the paths include conflicts, the high level, via a *split* action (described below), imposes constraints on the conflicting agents to avoid these conflicts.

Two improvements to CBS were suggested. First, Meta-agent CBS (MA-CBS) [Sharon *et al.*, 2012b; 2015] generalizes CBS by merging small groups of agents into meta-agents when beneficial. The main merge policy is to merge agents for which the number of conflicts seen so far exceeds a given threshold B; for good choices of B, MA-CBS reduces the runtime. Second, the *bypass* (BP) improvement to (MA)CBS (i.e., CBS with or without applying the optional merge action) was recently suggested [Boyarski *et al.*, 2015]. (MA)CBS arbitrarily chooses paths in the low-level. When BP is added on top of (MA)CBS we first try to find an alternative path (bypass) for one of the conflicting agents thus avoiding the need to perform a split and to add new constraints. (MA)CBS+BP further reduces the running time. A variant of CBS that solves MAPF suboptimally also appeared [Barer *et al.*, 2014].

In this paper we introduce *Improved-CBS* (ICBS) which further adds two new improvements to (MA)CBS+BP. Each of these improvements is strongly tied to one of the former improvements (MA and BP) as follows:

- **(1) Merge and restart. (MR)** In MA-CBS, agents are merged locally at each node of the search tree. Instead, when a decision to merge is made, we suggest to restart the search from scratch, with the new merged meta-agent treated as a single agent for the entire search tree.
- **(2) Prioritizing conflicts. (PC)** (MA)CBS (even with BP added) arbitrarily chooses which conflict to split. Poor choices may increase the size of the high-level search tree. To remedy this, we prioritize the conflicts according to three types: *cardinal*, *semi-cardinal* and *non-cardinal*. Cardinal conflicts always cause an increase in the solution cost, so ICBS chooses to split cardinal conflicts first. Additionally, bypasses to cardinal conflict cannot exist. Therefore, the optional BP improvement [Boyarski *et al.*, 2015] should only be applied for semi- or non-cardinal conflicts.

All four enhancements to CBS (i.e., MA-CBS, BP, PC and MR) are optional and can be added separately or in conjunction with the others, except for MR which is only relevant to MA-CBS. In this paper we show how to combine them all into a coherent improved version of CBS, namely ICBS.

Experimental results show that MR and PC each provide a significant speedup over the previous best CBS variant, namely (MA)CBS+BP [Boyarski *et al.*, 2015]. Even further speedup is achieved when both improvements are combined and ICBS outperforms other related algorithms in many cases where previous variants of CBS performed poorly.

Algorithm 1: High-level of ICBS

```
1  Main(MAPF problem instance)
2      Init R with low-level paths for the individual agents
3      insert R into OPEN
4      while OPEN not empty do
5          N ← best node from OPEN // lowest solution cost
6          Simulate the paths in N and find all conflicts.
7          if N has no conflict then
8              return N.solution // N is goal
9          C ← find-cardinal/semi-cardinal-conflict(N) // (PC)
10         if C is not cardinal then
11             if Find-bypass(N, C) then // (BP)
12                 Continue
13         if should-merge(a_i, a_j) then // Optional, MA-CBS:
14             a_ij = merge(a_i,a_j)
15             if MR active then // (MR)
16                 Restart search
17             Update N.constraints()
18             Update N.solution by invoking low-level(a_ij)
19             Insert N back into OPEN
20             continue // go back to the while statement
21         foreach agent a_i in C do
22             A ← Generate Child(N, (a_i, s, t))
23             Insert A into OPEN
24 Generate Child(Node N, Constraint C = (a_i, s, t))
25     A.constraints ← N.constraints + (a_i, s, t)
26     A.solution ← N.solution
27     Update A.solution by invoking low level(a_i)
28     A.cost ← SIC(A.solution)
29     return A
```

2 The Conflict Based Search Algorithm (CBS)

A sequence of individual agent move/wait actions leading an agent from s_i to g_i is referred to as a *path*, and the term *solution* refers 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. A solution is *valid* if it is conflict-free. The *cost* of a path is the number of actions in it (including wait), and the cost of a solution is the sum of the costs of its constituent paths.

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. 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.

The high-level of CBS searches the *constraint tree* (CT). The CT is a binary tree, in which each node N contains: **(1)** A set of constraints imposed on the agents ($N.constraints$), **(2)** A single solution ($N.solution$) consistent with these constraints, **(3)** The cost of $N.solution$ ($N.cost$).

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.

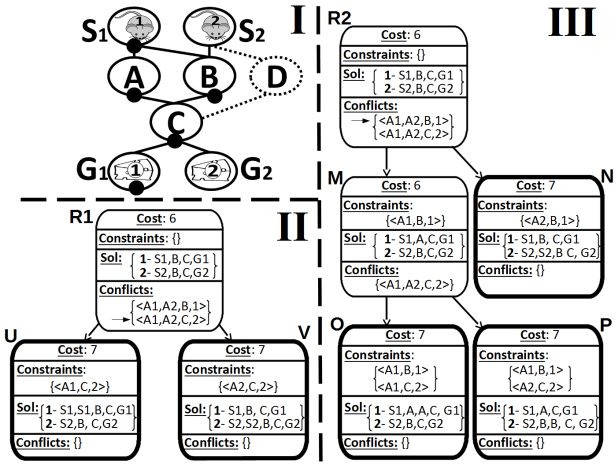


Figure 1: An MAPF instance and two possible CTs

$N.solution$ 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 has no conflicts. The high-level of CBS performs a best-first search on the CT where nodes are ordered by their costs ($N.cost$).

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).[1]

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, N is declared as the goal node, and $N.solution$ is returned. If, however, while performing the validation, a conflict is found for two (or more) agents, the validation halts and the node is declared as non-goal.

Resolving a conflict - the *split* action: 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, $\langle a_i, v, t\rangle$ or $\langle a_j, v, t\rangle$, must be satisfied. Consequently, CBS *splits* N and generates two new CT nodes as children of N, each adding one of these constraints to the previous set of constraints, $N.constraints$. Note that for each (non-root) CT node the low-level search is activated only for one agent – the agent for which the new constraint was added.

CBS Example: Pseudo-code for CBS is shown in Algorithm 1. It is explained by the example in Figure 1(I). The mice need to get to their respective pieces of cheese. The shaded lines (9-20) are all optional (except for part of line 9; *find-conflict* is mandatory) and include: the *merge* action

[1] Ties between low-level nodes are broken using a *conflict-avoidance table* (CAT) [Standley, 2010] which keeps track of the conflicts between agents. Paths with fewer conflicts are preferred.

for the optional MA-CBS (lines 13-20), the BP improvement (lines 11-12) as well as the two new improvements (PC: lines 9-10, MR: 15-16), each discussed in its own section below. The corresponding CT is shown in Figure 1(II). The root ($R1$) contains an empty set of constraints and the low-level returns path $P_1 = \langle S_1, A, C, G_1 \rangle$ for agent a_1 and path $P_2 = \langle S_2, B, C, G_2 \rangle$ for agent a_2 (line 2). Thus, the total cost of $R1$ is 6. $R1$ is then inserted into OPEN and will be expanded next. When validating the two-agents solution (line 6), a conflict $\langle a_1, a_2, C, 2 \rangle$ is found. As a result, $R1$ is declared as non-goal. $R1$ is split and two children are generated (via the *generate-child()* function, also shown in Algorithm 1) to resolve the conflict (line 22). The left child U adds the constraint $\langle a_1, C, 2 \rangle$ while the right child V adds the constraint $\langle a_2, C, 2 \rangle$. The low-level search is now invoked (line 27) for U to find an optimal path for agent a_1 that also satisfies the new constraint. For this, a_1 must wait one time step at S_1 and the path $\langle S_1, S_1, A, C, G_1 \rangle$ is returned for a_1. The path for a_2, $\langle S_2, B, C, G_2 \rangle$, remains unchanged in U. Since the cost of a_1 increased from 3 to 4 the cost of U is now 7. In a similar way, the right child V is generated, also with cost 7. Both children are added to OPEN (line 23). In the final step U is chosen for expansion, and the underlying paths are validated. Since no conflicts exist, U is declared as a goal node (lines 6-8) and its solution is returned.

3 Meta-agent merging improvements

In this section we deal with improvements that are based on merging agents into a meta-agent. We first describe the basic merge operation and then provide our new improvement.

3.1 Previous work: MA-CBS

MA-CBS(B) [Sharon *et al.*, 2012b; 2015] generalizes CBS by adding the option to *merge* the conflicting agents a_i and a_j into a *meta-agent* (Lines 13-20) instead of the *split* action. A meta-agent is logically treated as a single composite agent, whose state consists of a vector of locations, one for each individual agent. A meta-agent is never split in the subtree of the CT below the current node; it may, however, be merged with other (meta)agents into new meta-agents. The following actions are performed for a new meta-agent. (1) The constraints from the two merged agents are unified (Line 17, see [Sharon *et al.*, 2012b; 2015] for details). (2) We now call the low-level search again for this new meta-agent only (Line 18), as nothing has changed for the other agents that were not merged. In fact, the low-level search for a meta-agent of size M faces an optimal MAPF problem for M agents, and is solved with any coupled MAPF solver (e.g., A*).

Merge policy: The optional *merge* action is performed only if the *should-merge()* function returns *True* (Line 13). Many merging policies are possible for the *should-merge()* function. [Sharon *et al.*, 2012b; 2015] presented a simple, experimentally-effective merge policy. Two agents a_i and a_j are merged into a meta-agent a_{ij} if the number of conflicts between a_i and a_j seen so far during the search exceeds a predefined parameter B. Otherwise, a regular split is performed. This merge policy is denoted by MA-CBS(B). MA-CBS was shown to outperform CBS (without the merge option).

3.2 ICBS Improvement 1: Merge and Restart

Drawback

Merging agents into meta agents reduces the size of the CT at the cost of higher computational effort by the low-level solver. In MA-CBS this tradeoff is handled ad-hoc by requiring B splits before performing a merge. Nevertheless, MA-CBS may still duplicate much search effort.

As an example, consider a problem with only two agents (a_1, a_2). Suppose the agents were merged after encountering B conflicts and performing B split actions. At this point OPEN contains $B + 1$ CT nodes. A merge action may now be activated in all these nodes as the number of seen conflicts between these agents already exceeds B. As a result, exactly the same merge action (with the same meta-agent low-level searches) will be duplicated $B + 1$ times for all these nodes when they appear at the top of OPEN. Each of these merges produces identical CT nodes, i.e., one meta-agent $a_{1,2}$ and no constraints (as all internal constraints between a_1 and a_2 are deleted; see [Sharon *et al.*, 2015] on merge action details).[2]

Remedy

Had we known that a pair of agents are due to be merged, significant computational effort would have been saved by performing the merge ab-initio, at the root node of the CT. This "clairvoyant" merging policy is clearly observed to dominate MA-CBS(B) in auxiliary experiments we performed for various values of B. Naturally, "clairvoyant" is not realizable. Instead, we suggest a simple ad-hoc merge policy with a good balance between simplicity and effectiveness. Once a merge decision has been reached inside a CT node N, discard the current CT and *restart* the search from a new root node, where these agents are merged into a meta agent at the beginning of the search. We call this the *merge and restart* (MR) scheme. It is optionally applied in lines 15-16 of Algorithm 1.

MR is simple to implement, and saves much effort duplicated in the multiple CT nodes. In our example we would initially still generate $B + 1$ copies of the same problem, but then restart and only solve one instance of the problem for the meta-agent, thereby reducing the low-level effort for meta-agent $a_{1,2}$ by a factor of B. In contrast, MR might lose information when more agents are considered. For example, if our problem also contained a third agent a_3, which already conflicted with a_1 before the formation of meta-agent $a_{1,2}$ then split actions between a_1 and a_3 were also performed. Once a_1 and a_2 are merged, the entire process restarts and all constraints imposed on a_3 may have to be rediscovered.

Nevertheless, in most cases the gains outweigh the additional overhead. As it is very hard to model these quantities theoretically, we tested MR empirically (Section 5), and indeed it speeds up the search by a significant factor.

4 Improvements on the split action

CBS is very sensitive to the paths found by the low level and to the conflicts chosen for the splits. To this end we first sum-

[2] Duplicate detection and pruning can be applied in this extreme example because it only includes agents a_1 and a_2. But in general, when other agents exist this is not possible.

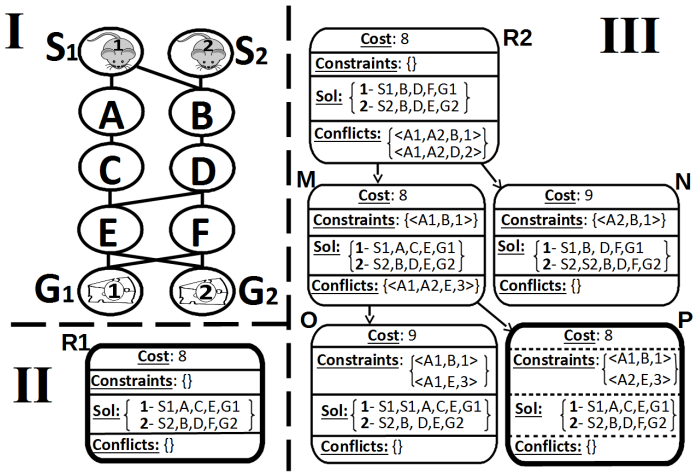


Figure 2: (I) MAPF instance (II) CT (III) Alternative CT

marize a previously published improvement (BP) and then introduce the new improvement (PC).

4.1 Previous work: the bypass improvement (BP)

[Boyarski *et al.*, 2015] introduced the *bypass* improvement (BP), which improves the low-level paths in a number of ways, thereby reducing the CT size. Here, we summarize the simplest variant and use it to build the final Improved-CBS.

Drawback

Consider Figure 2(I). Suppose that at the root of the CT ($R1$) the low-level is first activated for a_2 and that path $P_2 = \langle S_2, B, D, F, G_2 \rangle$ is found. a_1 uses the *conflict avoidance table* (CAT, described above) and finds path $P_1 = \langle S_1, A, C, E, G_1 \rangle$. There are no conflicts between these two paths and $R1$ is declared as the goal as shown in Figure 2(II).

Now suppose that a_1 goes first and finds path $P_1' = \langle S_1, B, D, F, G_1 \rangle$. Suppose that a_2 then finds path $P_2' = \langle S_2, B, D, E, G_2 \rangle$ as shown in the root ($R2$) of Figure 2(III). At $R2$ two conflicts are found $C_1 = \langle a_1, a_2, B, 1 \rangle$ and $C_2 = \langle a_1, a_2, D, 2 \rangle$. Suppose that C_1 is chosen for a split at $R2$. The left child M is generated with path $P_1 = \langle S_1, A, C, E, G_1 \rangle$ (cost 8) and both conflicts (C_1 and C_2) are resolved. But M includes another conflict $C_3 = \langle a_1, a_2, E, 3 \rangle$. Since a_1 must be at E at time step 3 then in its left child O, a_1 is forced to wait and a cost of 9 is achieved. In its right child P a_2 will choose path $P_2 = \langle S_2, B, D, F, G_2 \rangle$ which is the goal node.

We showed two scenarios. In the first, a single CT node was generated while in the second scenario 5 CT nodes were generated only due to poor choices of the paths at the root.

Remedy: Bypassing Conflicts

It is sometimes possible to prevent a split action and bypass the conflict by modifying the chosen path of one of the agents. For this we first make the following definitions.

(1) For each CT node N we use $N.NC$ to denote the total *number of conflicts* of the form $\langle a_i, a_j, v, t \rangle$ between the paths in $N.solution$. Calculating $N.NC$ is trivial.

(2) A path P_i' is a *valid bypass* to path P_i for agent a_i with respect to a conflict $C = \langle a_i, a_j, v, t \rangle$ and a CT node N, if the following conditions are satisfied: **(i)** Unlike P_i, P_i' does not include conflict C, **(ii)** $cost(P_i') = cost(P_i)$ and **(iii)** P_i and P_i' are both consistent with $N.constraints$.

(3) *Adopting* path P_i' by a CT node N means replacing P_i with a valid bypass P_i' for agent a_i.

(4) Adopting a valid bypass may introduce more conflicts compared to the original path and potentially lead to worse overall runtime. Thus, we only allow adopting bypasses that reduce $N.NC$. These are called *helpful bypasses*.

Bypassing conflicts (BP): For a given CT node N, BP peeks at either of the immediate children of N in the CT (by invoking $generate{-}child()$, presented in Algorithm1). If the path of a child includes a *helpful bypass* for the conflict of N this path is *adopted* by N without the need to split N and add new nodes to the CT. This can potentially save a significant amount of search due to a smaller size CT. BP is optionally added to the CBS pseudo code (lines 11-12 in Algorithm 1).

Example: consider again the root $R2$ of Figure 2(III). $R2$ is split into two nodes as described above. In the left child M agent a_1 finds path $P_1 = \langle S1, A, C, E, G1 \rangle$. However, $R2$ can *adopt* path P_1 for agent a_1. P_1 is a valid bypass because it does not include conflict $C_1 = \langle a_1, a_2, B, 1 \rangle$ which caused the split. In addition, it is a helpful bypass because $R2.NC = 2$ (conflicts C_1 and C_2) but after adopting the path from M, $R2.NC$ will decrease to 1 (conflict C_3). Now, instead of a CT with 5 nodes we get a CT with only 3 nodes.

While processing a node N, BP incurs no extra overhead due to peek operations. Basic CBS generates both children and adds both to OPEN. In the worst case when both peek operations fail, BP does exactly the same as CBS, i.e., generates and adds these children to OPEN. But if one of these nodes is found helpful then BP adds no new nodes to OPEN and might even avoid the need to generate and run a low-level search for the 2nd child if the first child includes a helpful bypass.

Additionally, note that node N in its "new suit", i.e., after adopting a path, will now be the best node in OPEN and will be chosen for expansion next.[3] Therefore, if it has a helpful child again then BP will be repeated here. In fact, the next *real* split will occur only when a bypass call fails.

4.2 ICBS Improvement 2: Prioritize Conflicts (PC)

Drawbacks of (MA)CBS and (MA)CBS+BP

To guarantee optimality, CBS expands nodes in a best-first order according to the cost function. While A* expands *all* nodes with $f < C^*$ (= the optimal solution cost) to guarantee optimality of the solution no such *mandatory nodes* are known for CBS. Consequently, CBS (even with BP) is very sensitive to the conflicts chosen to *split* on, as these can significantly influence the number of CT nodes. In its basic form, CBS arbitrarily picks a conflict for splitting (e.g., the first one seen). This can lead to poor performance and CBS can be stuck in an area with similar cost as demonstrated below.

Remedy: Splitting cardinal conflicts first

To address this drawback we distinguish between three types of conflicts and show that prioritizing them may immediately

[3] A mechanism like *immediate expand* [Stern *et al.*, 2010] which bypasses OPEN can be efficiently activated here.

increase the solution cost in the sub-tree below the current node thereby significantly reducing the size of the CT.

- **(1) Cardinal conflict.** A conflict $C = \langle a_1, a_2, v, t\rangle$ is *cardinal* for a CT node N if adding any of the two constraints derived from C ($\langle a_1, v, t\rangle$, $\langle a_2, v, t\rangle$) to N and invoking the low-level on the constrained agent, the cost of its path is increased when compared to its cost in N. An equivalent definition is that C is *cardinal* if all the consistent optimal paths for both a_1 and a_2 include location v at time step t. For example, in Figure 1(I) the conflict $\langle a_1, a_2, C, 2\rangle$ is cardinal. None of the agents has a path to its goal of length 3 which doesn't pass through C in time step 2. Therefore, one of these agents must travel more than three steps in order to resolve this conflict.
- **(2) Semi-cardinal conflict.** A conflict $C = \langle a_1, a_2, v, t\rangle$ is *semi-cardinal* for a CT node N if adding one of the two constraints derived from C increases $N.cost$ but adding the other leaves $N.cost$ unchanged. Equivalently, C is *semi-cardinal* if all the consistent optimal paths of one agent include location v at time step t, but the other agent has such a path that does not include v at time step t. For example, the conflict $\langle a_1, a_2, B, 1\rangle$ in Figure 1(I) is semi-cardinal (assuming the paths are $P_1 = \langle S_1, B, C, G_1\rangle$ and $P_2 = \langle S_2, B, C, G_2\rangle$). a_2 must use B at time step 2, but a_1 can choose A instead.
- **(3) Non-cardinal conflict.** A conflict $C = \langle a_1, a_2, v, t\rangle$ is *non cardinal* for a CT node N if **neither** of the two constraints derived from C causes an increase of $N.cost$. For example, in Figure 1(I), if the dotted lines (path through D) are enabled then the conflict $\langle a_1, a_2, B, 1\rangle$ is non-cardinal. Now agent a_2 can choose to go via node D at the same cost.

We now make the following observations: **(1)** If a cardinal conflict exists for CT node N then the cost of a valid solution below N must be greater than $N.cost$. **(2)** In any best-first search increasing a lower bound of a node in OPEN is beneficial as this lower bound is more informed.

Prioritization of conflicts (PC):

This new improvement works as follows. When a node N with $N.cost = c$ is chosen for expansion by CBS, we examine all its conflicts. If a cardinal conflict is encountered, it is immediately chosen for the split action. This generates two children with cost $> c$. In this case if another node N' in OPEN has cost c, then N' can be immediately chosen for expansion next without further developing nodes below N.[4] In contrast, if a cardinal conflict exists but instead we choose a semi-cardinal or a non-cardinal conflict, we may generate a large subtree below N with the same cost c. Furthermore, the cardinal conflict will reappear in all of the nodes in the subtree below N until it is chosen and resolved.

To illustrate this, consider again Figure 1(I) (assuming the paths chosen at R1 are $P_1' = \langle S_1, B, C, G_1\rangle$ and $P_2 = \langle S_2, B, C, G_2\rangle$). Two conflicts exist in the root node: a semi-cardinal conflict $C_1 = \langle a_1, a_2, B, 1\rangle$ and a cardinal conflict $C_2 = \langle a_1, a_2, C, 2\rangle$. If C_2 were chosen (Figure 1(II)) the costs increase immediately and the left child is declared as the goal. In contrast, Figure 1(III) illustrates the case where C_1 is chosen. In a simple implementation, it will be chosen as it is seen first. In its left child M the constraint $(a_1, B, 1)$ is added and a_1 will choose the path $P_1 = \langle S_1, A, C, G_1\rangle$. However, the cardinal conflict C_2 still exists and will be again chosen for a split. There are 5 nodes in the corresponding CT (Figure 1(III)). In contrast, the CT of Figure 1(II) has 3 nodes.

If no cardinal conflicts exist, the next preference is to choose semi-cardinal conflicts. Here, we know that the cost of at least one child will immediately increase. Finally, if only non-cardinal conflicts exist, one of them is chosen arbitrarily.

PC is shown in line 9, where we seek a cardinal/semi-cardinal conflict. [Boyarski *et al.*, 2015] suggested to apply BP for *every* conflict. However, in this paper we note that valid bypasses (with the same cost) can only be found for *semi-* or *non-cardinal conflicts* but not for *cardinal conflicts*. Therefore, in ICBS BP is applied more conservatively: with a cardinal conflict, we jump to line 13 and BP is not activated. Otherwise (only semi- or non-cardinal conflicts were found) we continue with the pseudo code allowing to perform BP on the chosen conflict.

Identifying a cardinal conflict using MDDs

The *multi-value decision diagram* MDD_i^c [Sharon *et al.*, 2013] is a DAG which compactly stores all possible paths of given cost c for a given agent a_i while completely ignoring the other agents. Nodes of the MDD can be distinguished by their depth below the source node. Every node at depth t of MDD_i^c corresponds to a possible location of a_i at time t that is on a path of cost c from s_i to g_i. MDD_i^c has a single *source node* at level 0, corresponding to agent a_i located at s_i at time t_0, and a single *sink node* at level c, which corresponds to agent a_i located at g_i at time t_c. The nodes of MDD_1^3 which corresponds to paths of length 3 for agent a_1 have a small black circle in their lower part in Figure 1(I).

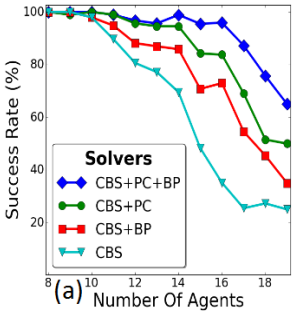
Observation: If, for a CT node N with $N.cost = c$, there exists a cardinal conflict C of the form $\langle a_i, a_j, D, t\rangle$, then the widths of $MDD_i^{c_i}$ and $MDD_j^{c_j}$ at time-step t must be 1.

Therefore, to find cardinal conflicts in CT node N we iterate over all conflicts C in N and test each conflict $C = \langle a_i, a_j, D, t\rangle$ for cardinality by building $MDD_i^{c_i}$ and $MDD_j^{c_j}$ (c_i is the cost of path of a_i in $N.cost$) and checking whether their width at depth t is 1. This is repeated until a cardinal conflict is found or all the conflicts in N have been checked. If no cardinal conflict is found, a semi cardinal conflict is chosen if it was encountered during the iteration. Otherwise, a non-cardinal conflict is arbitrarily chosen.

Maintaining and checking all MDDs incurs an overhead per CT node. But, this pays off in dense environments with many bottlenecks where many cardinal conflicts exist.[5]

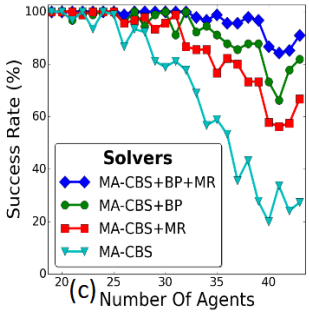
[4] Based on this, our optimized implementation pushes N back into OPEN with the new cost without even generating the two children. This proved beneficial in our experiments.

[5] In a practical implementation MDDs can be reused from parents when applicable. In addition, to find a cardinal conflict quickly, we check MDDs of agents in decreasing order of the number of conflicts in which they participate.



k	BP	CBS	PC	PC+BP
	Runtime (ms)			
8	173	250%	75%	**57%**
10	1066	357%	114%	**21%**
12	2668	357%	**29%**	36%
14	1241	714%	64%	**43%**
	CT nodes			
8	464	333%	86%	**57%**
10	2759	435%	113%	**17%**
12	5931	500%	35%	**35%**
14	2750	1111%	89%	**33%**

(b)



k	BP	MA-CBS	MR	BP+MR
	Runtime (ms)			
25	8,367	352%	67%	**60%**
30	3,857	829%	157%	**79%**
35	7,594	935%	113%	**63%**
40	37,615	N/A	104%	**41%**
	CT nodes			
25	128	310%	114%	**68%**
30	108	614%	212%	**82%**
35	146	839%	171%	**77%**
40	344	N/A	165%	**65%**

(d)

Figure 3: CBS variants: (a) success rate (b) runtime and nodes. MA-CBS variants: (c) success rate (d) runtime and nodes

Meta-agent conflicts. Building an MDD for meta-agents is expensive. Therefore in our implementation for MA-CBS we first build MDDs for meta-agents of size 1 (individual agents) which participate in a conflict. If a cardinal conflict was not found, we then try to find cardinal or semi-cardinal conflicts for larger meta-agents. We do this by logically generating the two children of the conflict and observing their costs.

5 Experimental Results

We performed extensive experiments on many possible parameter settings of the environment and of internal algorithmic parameters. We report representative results. First, we concentrate on each of our improvement alone compared to previous CBS solvers. We then take our best CBS variant (ICBS) and compare it to other known MAPF solvers.

The code for all the experiments was written in C# and all our experiments were conducted on Amazon Elastic Compute Cloud C4 servers running Microsoft Windows Server 2012 R2. Each High frequency Intel Xeon E5-2666 v3 (Haswell) processor operates at 2.9GHz, and has 3.33GB of RAM.

5.1 Each improvement vs. baseline

Our first experiment compares PC and MR against the corresponding baseline, i.e., CBS+BP and MA-CBS+BP, respectively.[6] Figure 3(a) shows the *success rate* (=number of instances solved by each algorithm within 5 minutes) for CBS, CBS+BP, CBS+PC and CBS+PC+BP (both improvements) on 8x8 4-connected grids with 15% obstacles averaged on 100 random instances.[7] CBS+PC clearly outperforms CBS+BP [Boyarski *et al.*, 2015], the previous best CBS variant. CBS+PC+BP even further increases the success rate. Figure 3(b) presents a table with the runtime (in ms) and number of generated CT nodes averaged over those instances that were solved by *all* the algorithms (=success rate of CBS). CBS+BP (=previous state-of-the-art CBS solver) serves as the baseline (the 'BP' column) and the other columns present percentages (%) relative to CBS+BP . Clearly CBS+PC and CBS+PC+BP outperform CBS+BP by up to a factor of 5 on this set. The improvement in nodes is nicely reflected in the runtime (e.g., 17% vs. 21%). This suggests that the overhead per node for PC and for BP is minor.

Figure 3(c,d) shows the success rate and runtime/nodes results when comparing MA-CBS(64)+BP to MA-CBS(64), MA-CBS(64)+MR and MA-CBS(64)+BP+MR on a 20x20 empty grid. Here too, the percentages are relative to MA-CBS(64)+BP. Again, the advantage of adding MR can be clearly seen in all measures. Importantly, note that here, the runtime improvement of MR is larger than the improvement in nodes. This is because by restarting, MR cancels many CT nodes that have heavy computations.

5.2 Comparison with other algorithms

Finally, based on extensive experiments we fine-tuned a number of leading algorithms with the best parameters. We compare ICBS(25) (=MA-CBS(25)+BP+PC+MR), to the following other MAPF solvers: **(1)** MA-CBS(25)+BP, the best former CBS variant. **(2)** MA-CBS(25) **(3)** EPEA* [Felner *et al.*, 2012; Goldenberg *et al.*, 2014], an enhanced version of A* designed for cases with large branching factors, the best A* variant. **(4)** ICTS+p [Sharon *et al.*, 2013; 2011], the best ICTS variant.

Figure 4 presents the success rate (left) and runtime (right) for the three standard benchmark maps (brc202d:top, ost003d:middle, den520d:bottom) of the game *Dragon Age: Origins* (DAO) [Sturtevant, 2012] that were used by Sharon et al. [2013; 2015], 100 instances per map.[8]

In all three maps, ICBS is clearly the best CBS variant and it clearly outperforms the previous best CBS variant, namely MA-CBS+BP. ICBS is also the best algorithm amongst the ones compared here in runtime. ICBS is also the best in suc-

[6] We followed Sharon et al. [2013; 2015] and for this experiment we activate the *Independence Detection* framework [Standley, 2010] in a preprocessing level and identify problem instances for which the agents are dependent. These were called *type 1* experiments by Sharon et al. [2013; 2015].

[7] The starting positions for agents were chosen randomly and their goal positions were chosen by a 100000-step random walk from the starting position.

[8] Following Sharon et al. [2012b; 2013], here we enhanced the various algorithms by the *Independence Detection* framework (ID) [Standley, 2010] which identifies independent groups of agents and runs the solver for each group (Type 2 experiments). ICBS was an exception as ID did not provide any speedup for ICBS. Therefore, the results presented for ICBS are without ID. MR is reminiscent of ID as they both run new solvers for a number of coupled agents. Thus, there is no point of applying them together.

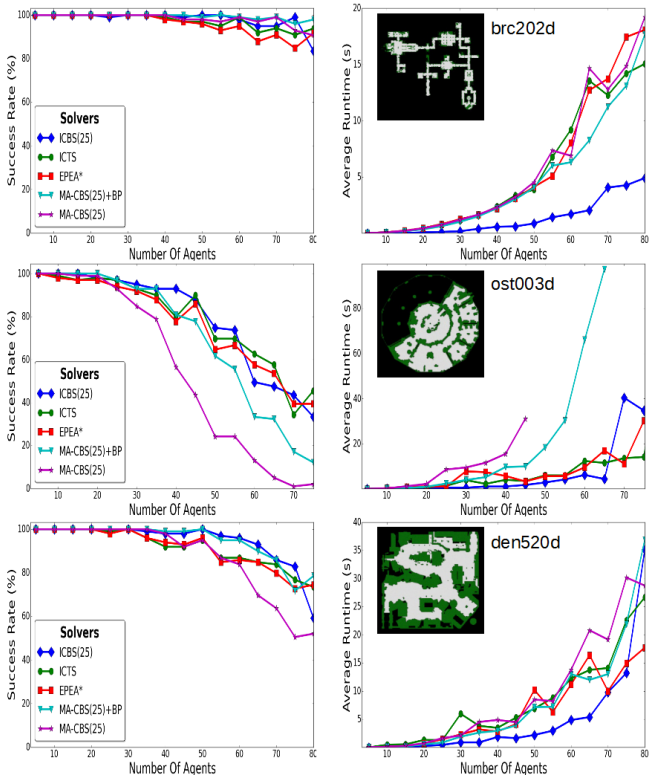


Figure 4: Comparison of all algorithms, DAO maps

cess rate for brc202d and for den520d. In ost003d there is a tie between ICBS, EPEA* and ICTS in the success rate. In this map the number of conflicts is very large due to open spaces (see [Sharon *et al.*, 2015]). Thus, the previous CBS variants spawn a very large CT in this map and were worse than the other algorithms. But, ICBS provides a great improvement and was able to tie with the other algorithms on this map. It is important to note that no MAPF solver is best across *all* circumstances and *all* topologies.

6 Conclusions and Future Work

When CBS is used, adding all our improvements proved beneficial. ICBS is the best or ties with other solvers as demonstrated empirically on standard benchmarks. Future work will: (1) Further study the attributes and the relations between these enhancements. (2) Find a better merge policy than a fixed B parameter. (3) Compare and better understand the pros and cons of the various MAPF algorithms under different circumstances.

7 Acknowledgements

The research was supported by the Israeli Science Foundation (ISF) under grant #417/13 to Ariel Felner and Eyal Shimony.

References

[Barer *et al.*, 2014] M. Barer, G. Sharon, R. Stern, and A. Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *SOCS*, 2014.

[Boyarski *et al.*, 2015] E. Boyarski, A. Felner, G. Sharon, and R. Stern. Don't split, try to work it out : Bypassing conflicts in multi-agent pathfinding. In *ICAPS-2015, Jerusalem, Israel, June 7-11*, 2015.

[Erdem *et al.*, 2013] Esra Erdem, Doga G. Kisa, Umut Oztok, and Peter Schueller. A general formal framework for pathfinding problems with multiple agents. In *AAAI*, 2013.

[Felner *et al.*, 2012] A. Felner, M. Goldenberg, R. Stern, G. Sharon, T. Beja, R. Holte, J. Schaeffer, N. Sturtevant, and Z. Zhang. Partial-expansion A* with selective node generation. *AAAI*, 2012.

[Goldenberg *et al.*, 2014] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer. Enhanced partial expansion A. *J. Artif. Intell. Res. (JAIR)*, 50:141–187, 2014.

[Sharon *et al.*, 2011] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. Pruning techniques for the increasing cost tree search for optimal multi-agent pathfinding. In *SOCS*, pages 150–157, 2011.

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

[Sharon *et al.*, 2012b] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. In *SOCS*, 2012.

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

[Sharon *et al.*, 2015] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent path finding. *Artificial Intelligence Journal (AIJ)*, 218:40–66, 2015.

[Standley, 2010] T. S. Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 2010.

[Stern *et al.*, 2010] Roni Stern, Tamar Kulberis, Ariel Felner, and Robert Holte. Using lookaheads with optimal best-first search. In *AAAI*, 2010.

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

[Surynek, 2012] Pavel Surynek. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*, pages 564–576. 2012.

[Wagner and Choset, 2011] G. Wagner and H. Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *IROS*, pages 3260–3267, 2011.

[Yu and LaValle, 2013] Jingjin Yu and Steven M. LaValle. Planning optimal paths for multiple robots on graphs. In *International Conference on Robotics and Automation (ICRA)*, pages 3612–3617, 2013.