Using Hierarchical Constraints to Avoid Conflicts in Multi-Agent Pathfinding

Thayne T. Walker, David M. Chan, Nathan R. Sturtevant
Department of Computer Science, University of Denver
Denver, CO, USA
thayne.walker@du.edu, davidchan@cs.du.edu, sturtevant@cs.du.edu

Abstract

Recent work in multi-agent path planning has provided a number of optimal and suboptimal solvers that can efficiently find solutions to problems of growing complexity. Solvers based on Conflict-Based Search (CBS) combine single-agent solvers with shared constraints between agents to find feasible solutions. Suboptimal variants of CBS introduce alternate heuristics to avoid conflicts. In this paper we study the multi-agent planning problem in the context of non-holonomic vehicles planning on a lattice. We propose that in addition to using heuristics to avoid conflicts, we can plan using a hierarchy of movement constraints to efficiently avoid conflicts. We develop a new extension to the CBS algorithm, CBS with constraint layering (CBS+CL), which iteratively applies different movement constraint models during the CBS planning process. Our results show that this approach allows us to solve for about 2.4 times more agents in the same amount of time when compared to regular CBS without using a constraint hierarchy.

1 Introduction

Consider the problem of coordinating unmanned aerial vehicles (UAVs) when fighting a forest fire. Each vehicle may have different capabilities for sensing, communicating, or even deploying fire-fighting resources. These vehicles will need to travel from their respective take-off and landing zones to perform sensing, communication, or delivery tasks, and then return without collision. This is just one example of the multi-agent pathfinding (MAPF) problem, where multiple agents must coordinate movement and avoid conflicts. Other examples include package delivery, land surveys, security patrolling, search and rescue, and farm upkeep.

An instance of the MAPF problem is defined by a graph $G = (V, E)$ of the search space, a set of k agents labeled $a_1 ... a_k$, and a set of start and goal locations for each agent $s_i \in V$ and $g_i \in V$ where $s_i \neq s_j, g_i \neq g_j$ for all $i \neq j$. A solution to a MAPF problem is a set of k paths which are non-conflicting. That is, no two agents may traverse intersecting edges or occupy the same vertex at the same time step. We seek solutions which minimize the sum of individual path costs, and are interested in centralized plans, as opposed to distributed agents.

MAPF is an NP-hard problem (Yu and LaValle 2013). For a domain with N vertices and k agents, the state space contains $\frac{N!}{(N-k)!}$ states. However, where each agent can move in b_{base} different ways, the branching factor b is b_{base}^k – exponential in the number of agents. Searching to a depth of t time steps could yield a total search space of $O(b^t)$ nodes, though many of them might be duplicates in the state space.

In this paper we study the MAPF problem in the context of fixed-wing aircraft in a controlled airspace. This environment has non-unit edge costs, non-unit time steps and high dimensionality and thus a high branching factor, and is subject to kinematic, orientation and speed constraints. Our work contrasts from almost all previous work on MAPF which has traditionally focused on two-dimensional grid worlds with low branching factors and unit edge costs with holonomic cardinal direction movement.

We develop a new sub-optimal approach, *Conflict-Based Search with Constraint Layering* (CBS+CL), which uses a hierarchy of edge subgraphs (subgraphs with edge deletions) to simplify the planning process. CBS+CL plans individual agents in the world, successively removing artificially added constraints from each agent as conflicts are discovered. Initially, for instance, an agent may be constrained to only move in the cardinal directions. But when collisions must be resolved, constraints on an agent are relaxed to allow octile movement, giving it more freedom to move and avoid conflicts. Our results show that this approach allows us to solve about 2.4 times more agents than regular CBS on the original graph and over 50% more agents in the same amount of time when compared to regular CBS on edge subgraph abstractions.

2 Background

This paper studies MAPF in the aviation domain. Because of the high branching factor, non-unit edge costs and kinematic movement inherent to the domain, even single-agent search is non-trivial.

There exist two major classes of sub-optimal MAPF algorithms; sub-optimal variants of optimal algorithms which aim for high-quality solutions and polynomial-time solvers which only aim for valid solutions. In the latter class are algorithms such as Parallel Push and Swap (PPS) (Sajid, Luna, and Bekris 2012), Tree-based Agent Swapping Strategy (TASS) (Khorshid, Holte, and Sturtevant 2011) and others, but none of these are formulated for non-holonomic vehicles. In the former class we have algorithms such as M* (Wagner and Choset 2011), ICTS (Sharon et al. 2013) and EPEA* (Goldenberg et al. 2014). But M* and ICTS are not well-formulated for non-unit time step domains. The EPEA* operator selection function (OSF) is extremely non-trivial when dealing with non-unit edge costs and non-unit time steps.

Sub-optimal algorithms which work readily for non-holonomic movement and non-unit cost domains are CBS variants such as GCBS, BCBS, ECBS (Barer et al. 2014) and ECBS+HWY (Cohen, Uras, and Koenig 2015). There are additional A*-based algorithms such as Weighted-A* (Pohl 1970), (Weighted) PEA* (Yoshizumi, Miura, and Ishida 2000) and Maximum Group Size (MGS) (Standley and Korf 2011).

We chose the CBS algorithm for the basis of our research because it is well-formulated for non-unit time step and non-unit-cost domains and because of its simplicity at the domain-specific level. CBS allows us to use formulations for a single-agent A* search without having to completely re-formulate the environment and heuristics for the multi-agent case.

The CBS Algorithm

The CBS algorithm (Sharon et al. 2012) is an optimal and complete best-first search algorithm for MAPF. Pseudocode for CBS is in Algorithm 1. In the CBS algorithm, paths are initially found for each agent in a low-level search which does not take other agents into account (line 3). Once individual paths are found, a conflict check is performed (line 8). If no conflicts are found, the algorithm terminates with the solution (line 10). If a conflict is found, the conflict is added as a node in a high-level search tree (line 19).

CBS organizes conflicts into a conflict tree where each node in the tree represents a potential solution (a collection of agent paths) which contains a conflict. The cost of the potential solution is the sum of the individual path costs or SIC. A conflict is defined as a tuple $\langle a_i, a_j, v, t\rangle$ where a_i, a_j are the conflicting agents, t is the time of the conflict, and v is the vertex at which the conflict occurred. Alternatively, a conflict can contain an edge e instead of a vertex if the conflict occurred at an edge. CBS can not tell which agent's path should be altered to achieve an optimal non-conflicting solution, and so it creates two conflict tree (CT) nodes: $\langle a_i, v, t\rangle$ and $\langle a_j, v, t\rangle$ (line 13). Then new constraints are added (line 15) and the low-level search is re-performed (line 17) for the conflicting agent in both contexts to update the potential solution and get the SIC (line 18). Then these nodes are placed into the OPEN list (line 17). The search terminates when a valid solution is found or when the OPEN list is empty.

CBS Enhancements

Several enhancements to the CBS algorithm have been proposed. CBS+BP (Boyarski et al. 2015a) proposes finding an alternate, non-conflicting path called a "bypass", which may allow an early search termination or preclude the creation of extraneous constraint nodes. The Improved-CBS (ICBS) algorithm (Boyarski et al. 2015b), uses the BP notion of conflict count to prioritize the high-level search.

Algorithm 1 CBS

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

The Enhanced-CBS (ECBS) (Barer et al. 2014) group of algorithms focus on sub-optimal search, with algorithms for bounded and unbounded sub-optimal solutions. They observed that weighted A* search at the low-level produces longer paths, and thus increases the probability of conflicts. Instead, a non-weight-based sub-optimal heuristic for the low-level search, along with several heuristics for the high-level search were proposed. One such heuristic is the number of conflicts, NC, rather than the SIC. It was shown that when ordering the OPEN list by NC, a sub-optimal solution was often found in less time on unit-cost domains.

The meta-agent CBS (MA-CBS) (Sharon et al. 2015) algorithm proposed an additional enhancement which uses an adjustable conflict threshold parameter. If the number of conflicts between two agents exceeds the threshold, a new meta-agent is created to handle finding a conflict-free solution for that subset of agents. Meta-agents may be solved via a different MAPF algorithm such as M* (Wagner and Choset 2011), EPEA* (Goldenberg et al. 2014), ICTS (Sharon et al. 2013) or even MA-CBS with different thresholds. Then, the meta-agents are handled as a single agent from the CBS perspective.

3 Method

CBS+CL uses a hierarchy of environment abstractions, but not all abstraction techniques are applicable for use with CBS+CL.

Environment Abstraction

Environment abstractions can be created by forming sub-graphs via node contractions, edge deletions or adding new embeddings to the original search graph (Hoffmann 2005).

Various abstraction techniques have been published viz. Clique Abstraction, Sector Abstraction, Line Abstraction, STAR (Holte et al. 1996; Sturtevant and Jansen 2007) and JPS (Harabor and Grastien 2011). All of these abstractions rely on the downward refinement property which means that a path between nodes in an abstracted graph must be refinable to a path in the original graph.

The intuition behind our approach is to combine the strengths of constrained and unconstrained environments. To mitigate the cost of search in environments with high branching factors it is helpful to perform the low-level search on a constrained/abstracted version of the original environment. By introducing movement constraints into the original environment we reduce the branching factor, allowing better performance in the CBS low-level search step.

We formulate constrained environments as graph abstractions where the graph is the lattice on which agents move. The process of enforcing constraints on an agent is analogous to removing edges from the planning lattice that it moves on. Conversely, adding edges to the planning lattice is analogous to removing constraints.

Formally, graphs formed by edge deletion are known as *edge subgraphs* (Gaschnig 1981). An edge subgraph is a graph $G' \subseteq G$ s.t. $G' = (V, E'), E' = E \backslash X$ where X is the set of edges deleted from the initial graph G.

Search on edge subgraphs does not require special downward refinement as edge subgraphs contain a subset of the edges in the original domain. Additionally, this method of abstraction maintains the original edge lengths which allows conflict detection logic to be the same for all abstractions.

Abstraction For Conflict Avoidance A sub-optimal method of conflict avoidance at the low-level is to direct the search in ways that produce fewer conflicts. Direction maps (Jansen and Sturtevant 2008) have been proposed to augment heuristics. Direction maps provide an underlying flow field which agents are penalized if they do not follow. Direction maps can be formulated as highways or circular movement patterns. In ECBS+HWY, (Cohen and Koenig 2016) direction maps are used to influence agent movement to produce significant performance improvements.

Edge subgraphs are similar to flow fields because possible movements are restricted. Flow-Annotation Replanning (FAR) (Wang and Botea 2008) performs static analysis of the search graph and creates edge subgraphs, favoring directed edges so that collisions are less likely.

Instead of augmenting heuristics or abstraction via node contraction, we have taken an approach similar to FAR, which is abstraction via the creation of edge subgraphs. However, unlike FAR, we do not perform static analysis and augmentation of the environment. Instead, we apply uniform edge deletions across the entire search graph.

Constraint Layering for Conflict Avoidance

Consider an environment where agents are allowed to move on an octile grid. The search will have a maximum branching factor of 9 if waiting is allowed. If we restrict agents to cardinal directions or wait, the maximum branching factor is reduced to 5 which may speed up pathfinding, however

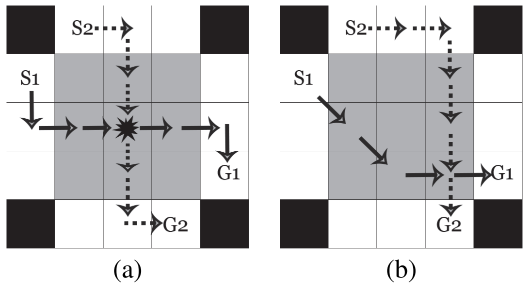

Figure 1: Comparison of grid-based search with differing movement constraints. In (a), movement for both agents is constrained to be 5 connected, thus paths must conflict in any of the grey squares. In (b), agent S_2 moves on a 5 connected grid, but agent S_1 moves on a 9 connected grid.

collisions become more likely. This scenario is illustrated in Figure 1, where both agents may only choose to move in a cardinal direction or wait. In this example, on any optimal path the agents will collide in the gray area. In fact, CBS will exhaustively examine all optimal paths for both agents before trying any sub-optimal paths, producing a CT node for each square in the gray area. Figure 1b shows a new scenario where agent A is allowed to use octile movement rules. Now, both agents can reach their goals optimally in their own environment without collision.

The proposed modification to the CBS algorithm CBS+CL is shown in Algorithm 2. We first define several environment abstractions, ranging from coarser (more constrained) to finer (less constrained) which we reference in the algorithm (line 2). Then we set a conflict threshold for each environment (line 3). When the number of conflicts between two agents meets the threshold for a particular environment, we still create two CT nodes as in traditional CBS, except in CBS+CL we re-plan the conflicted agent path in each CT node using the new environment (line 21). If we set our thresholds to be incremental, our search environments become "layered" in the sense that we can search on increasingly fine environments.

4 Theoretical Analysis

To begin our theoretical analysis, we introduce the following two concepts:

Definition 4.1. *Edge Subgraph Abstraction*

Let M be a graph $M = (V, E)$. Then an *edge subgraph abstration* M' of M is a subgraph of M s.t. $M' = (V, E'), E' \subseteq E$.

Definition 4.2. *Abstraction Set*

An *Abstraction Set* of the graph M_0 with size n is a partially ordered set $\mathcal{M}$ of edge subgraph abstractions of M_0, such that $\mathcal{M} = \{M_0, \dots, M_{n-1}\}$ where the ordering of the set is with respect to the subgraph ($\leq$) operation. Note that $\forall i \neq 0\ M_i \leq M_0$. If for some $0 \leq i, j < n, M_i \leq M_j$, we call M_i a *finer abstraction* with respect to M_j, and we call M_j a *coarser abstraction* with respect to M_i. We call M_0 (the finest abstraction) the *base abstraction*.

Algorithm 2 CBS+CL

```
1: Input: MAPF instance
2: $Env$ = List of Environments
3: $T$ = List of conflict thresholds per environment
4: $R$.constraints = $\emptyset$
5: $R$.solution = find individual paths using the low-level() on $Env[0]$
6: $R$.cost = SIC($R$.solution)
7: insert $R$ to OPEN
8: while OPEN not empty do
9:     $P \leftarrow$ best node from OPEN // lowest solution cost
10:    Validate the paths in $P$ until a conflict occurs.
11:    if $P$ has no conflict then
12:        return $P$.solution // P is goal
13:    end if
14:    $C \leftarrow$ first conflict ($a_i$, $a_j$ , $v$, $t$) in $P$
15:    for agent $a_i$ in $C$ do
16:        $A \leftarrow$ new node
17:        $A$.constraints $\leftarrow$ $P$.constraints + ($a_i$, $s$, $t$)
18:        $n \leftarrow size\,of\,A$.constraints
19:        $i \leftarrow$ get-environment-index$(n, T)$
20:        $A$.solution$\leftarrow$ $P$.solution.
21:        Update $A$.solution by invoking low-level($a_i$) on $Env[i]$
22:        $A$.cost = SIC($A$.solution)
23:        Insert $A$ to OPEN
24:    end for
25: end while
```

Traditional CBS runs as a best first search over sets of paths found in the base abstraction M_0. In CBS+CL, however, we identify each agent with a member of an abstraction set $\mathcal{M}$ of M_0 which that agent uses for planning. Agents can be *promoted* from a coarser abstraction in $\mathcal{M}$ to a finer abstraction in $\mathcal{M}$, creating additional paths for agents to explore, but simultaneously increasing the branching factor of the single-agent search. In CBS+CL, we define the set $\mathcal{K} \subseteq \mathcal{P}(\mathbb{N}) = \{\kappa_0, \ldots, \kappa_{n-1}\}$ to be a set of conflict thresholds (an algorithm hyper-parameter) such that if an agent participates in κ_i conflicts or more it must be promoted to an abstraction at least as fine as M_i. $\mathcal{K}$ must have the properties $\kappa_i > \kappa_{i+1}$ and $\kappa_{n-1} = 0$. An agent will be promoted from M_i to M_{i+1} when the number of conflicts reaches the threshold κ_{i+1}. In our current implementation of CBS+CL, we enforce a total ordering on $\mathcal{M}$ and begin with all agents in the coarsest abstraction M_{n-1}, however the proofs below are for an arbitrary partial ordering where any agent begins in a coarse abstraction of $\mathcal{M}$. CBS+CL terminates only if a solution is found by the algorithm, and a solution exists.

Unlike CBS, the suboptimality of CBS+CL is not easily bounded. Consider Figure 2. In this figure, agent s_1 is planning to goal state g_1, and agent s_2 is planning to goal state g_2. The optimal paths for both agents conflict in state B. The agents cannot see the edge between states A and C because they are planning in an edge subgraph abstraction, so either agent s_2 is forced to take a high cost path to g_2 through state F or s_1 must take the high cost path through state G - even

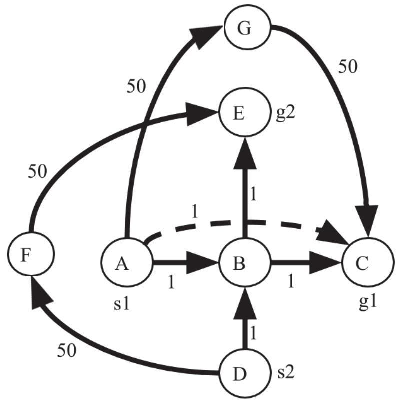

Figure 2: Example showing that in some general directed graphs, edge deletion abstraction may lead to large suboptimality. The dotted line between s_1 and g_1 lies in a finer abstraction, but because the abstraction is never visited and in the directed graph, all agents must move, one of the cost 100 loops must be taken.

though a very quick path exists for agent s_1. The path directly from A to C for agent s_1 is never explored in this example, because CBS+CL has found a valid solution in a coarser abstraction. Because path costs like those through nodes F and G in this example may be altered arbitrarily, it is impossible to trivially bound the suboptimality of the algorithm.

Thus, we see that because agents plan independently from each other, and in independent environments, an agent may have a cheaper optimal path in a finer abstraction, however that path may never be explored as the agent may not reach the number of conflicts necessary to investigate that path.

CBS+CL is, however, guaranteed to find the solution if one exists, which follows directly from the fact that CBS has the same guarantee (Sharon et al. 2012). The intuition for such a proof is that given any set of agents, if no solution is found, each agent will eventually be promoted into the base environment M_0. In the base environment, the operation of CBS+CL is identical to the operation of CBS, as no promotion is possible for any agent (and agent promotion is the only way that CBS+CL differs from CBS in any abstraction). The proof, and all assumptions are as follows:

Assumption 1. *There is a solution in* M_0.

In the case of an unsolvable instance, CBS+CL will run forever, generating conflicts out to future time steps ad infinitum. Fortunately, (Yu and Rus 2015) have shown a polynomial time algorithm to determine solvability of a MAPF instance. We make the assumption that problem instances are checked for solvability before running CBS+CL.

Assumption 2. $\mathcal{M}$ *is finite.* M_0 *has finite branching factor.*

We require that there be a finite number of abstractions, as otherwise CBS+CL is unable to guarantee termination in the case of a solution, as it may run forever examining abstractions in which there are no possible solutions. We require that M_0 have a finite branching factor, otherwise A^* is not guaranteed to return even in the case of single agent searches.

Assumption 3. *Let $M_i \in \mathcal{M}$. Then for all $a = (s, g) \in A$ if there exists a path from s to g in M_0 then there exists a path from s to g in M_i.*

The above assumption requires that any M_i is connected for each pair of start and goal vertices in the graph if the original graph M_0 is connected in those vertices. In fact, this is very difficult to accomplish for a general abstraction. In our implementation of CBS+CL, we enforce that within a fixed radius of all start and goal vertices, agents have free range of motion to ensure this property. Note that this is not necessary in all abstractions, however making it the case allows each abstraction to be solved in any single-agent context, greatly increasing the performance of the algorithm.

Assumption 4. *For any $M_i \in \mathcal{M}$ all edge costs are strictly greater than zero.*

This assumption is necessary for A^*, but also guarantees that at some point in the graph, we will have non-decreasing node costs, as demonstrated in a later lemma. Further, this allows us to bound the amount of time to solution.We continue the proof, and begin with a pair of lemmas:

Lemma 4.1. *If a solution exists in M_0, any k agents cannot conflict forever to the complete exclusion of another set of agents.*

Proof. Because κ_0 is finite, any set of agents will eventually be promoted to M_0 after enough conflicts occur between them. Therefore following the proof that regular CBS will find a solution if one exists under assumption 1, k agents cannot conflict forever without eventually finding a feasible solution. Thus any other set of agents will eventually be allowed to plan to a solution or potentially be promoted. □

Lemma 4.2. *Every path (possibly infinite length) from the root of the conflict tree of CBS+CL to a potential leaf of the confict tree of CBS+CL either terminates with a feasible solution, or there exists an earliest node N on the path of finite depth such that all of the paths in N are planned in the base abstraction M_0.*

Proof. Suppose first that the path does not terminate at a feasible solution. We will prove by contradiction that N as defined above must exist. Suppose that N does not exist, thus for every node in the path from root to leaf there must be at least one agent, a, that is not in M_0 and has no solution in its current abstraction M_i. Because CBS+CL requires a conflict to create a child CT node, the number of conflicts must increase for some agent at each level in the tree. Additionally, CBS+CL requires an agent to be promoted to abstraction M_i when the number of conflicts reaches κ_i. Because $\mathcal{K}$ is finite and κ_0 is fixed, it is impossible for a to remain at $< \kappa_0$ conflicts under the condition that a solution is not found at a higher level in the conflict tree. In other words, at some point along a traversal from the root of the conflict tree toward a leaf, the number of conflicts involving a must exceed κ_0 and a will be promoted, contradicting the fact that a is never promoted to a finer abstraction. Thus, every path from the root of the conflict tree of CBS+CL to a potential leaf of the conflict tree of CBS+CL either terminates with a feasible solution, or there exists an earliest node N on the path of finite depth such that all of the paths in N are planned in the base abstraction M_0. □

Corollary 4.3. *After a finite amount of time, CBS+CL has the property that nodes on the frontier of the search have children of non-decreasing cost.*

Proof. By lemma 4.2 each path from a root to a potential leaf eventually (in finite time) has all agents in the M_0 environment with no possibility of promotion. CBS re-plans only one agent at a time (with an added constraint) when there is a conflict. Because of the added constraint any re-planned path will necessarily have cost greater than or equal to the cost of its path in the parent node. Thus it is not possible to have a child node with cost less than its parent. □

Lemma 4.4. *There is always a valid path from the root of the CBS+CL tree to the abstraction-relative optimal solution, if a solution exists.*

Proof. Consider a traversal from the root of the tree to the optimal solution. At each node in the conflict tree of CBS+CL, a conflict in a location L is resolved between agent a_1, or agent a_2. Three scenarios are possible - $1)$ In the optimal set of paths a_1 is present in location L, $2)$ in the optimal set of paths a_2 is present in location L or $3)$ neither a_1 nor a_2 are present in location L in the optimal set of paths. If scenario three occurs, then both branches of the tree may contain the optimal set of paths as a leaf node. If scenario one or two happens, then exclusively either the left or right branch must contain the optimal set of paths as a leaf node. In no scenario does neither path contain the optimal solution (if it exists), thus, there must be some traversal in which the optimal solution (if it exists) is valid as a leaf node. □

Theorem 4.5. *CBS+CL will terminate with a solution if such a solution exists.*

Proof. By corollary 4.3, we know that after a finite amount of time, either a solution has been reached, or the costs in the CBS+CL tree are non-decreasing for all of the nodes on the frontier of the search. If no solution has been found, then by lemma 4.4 the optimal solution (if it exists) must still be in the sub-trees of one of the nodes on the frontier of the search. The optimal solution must be identified with some cost c, and by assumption 4, we can conclude that there are finite number of nodes per cost (Sharon et al. 2012), and that we must examine a finite number of nodes before arriving at c as the smallest node cost on the frontier. Thus, in finite time, we will arrive at the node containing the optimal solution, and CBS+CL will terminate. □

Thus, we have shown theoretically that even though it is difficult to easily bound the suboptimality of CBS+CL, we guarantee that CBS+CL terminates with a solution if one exists. Further, our empirical results show that CBS+CL does

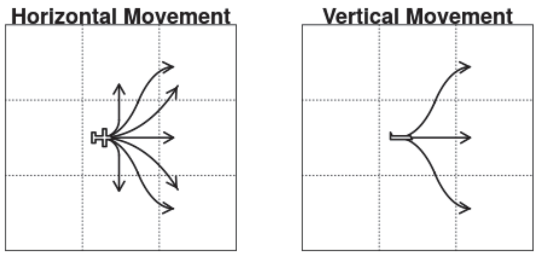

Figure 3: Grid-based vertical and horizontal aircraft movement model

not, in practice, generate extremely costly paths for agents compared to optimal algorithms.

5 Domain

In this paper we apply both traditional CBS and our proposed improvements to a model of an aviation environment. Agents in this model are non-holonomic, having kinematic, orientation and speed constraints. One might think that collisions are unlikely in 3D airspace. However, there are scenarios where traffic can be extremely congested, such as landing and takeoff points, urban environments, choke-points, etc. While ensuring vertical separation could be an easy solution (e.g. assigning a unique altitude for each agent), such solutions are wasteful as time and fuel must be spent to climb to the assigned heights.

Note that fixed-winged aircraft (as opposed to helicopters or balloon-type aircraft), cannot hover to avoid collisions. Furthermore, such aircraft cannot slow down below their stall speed, nor speed up past their maximum speed, and operate most efficiently at a unqiue cruise speed. Finally, winged aircraft are limited in their turn radius and require time and space to perform turns.

Aviation Domain Model Implementation

Our model is based on the kinematics of fixed-wing aircraft. We modeled the airspace with a three-dimensional grid, the width of which will allow an aircraft moving at maximum speed to negotiate a 90° turn. Each grid cell forms a cube with all three dimensions being the same.

In our model, we have a maximum branching factor of 63 - all combinations of heading changes: {0°, +45°, -45°, +90°, -90°, left shift, right shift}, speed changes: {no change, speed up, slow down} and height changes: {no change, climb, descend}. A *shift* is a maneuver where the aircraft moves in the diagonal direction but does not change heading. This environment which allows simultaneous change in heading, height and speed will be referred to in this paper as the "base" environment. See Figure 3 for a representation of the movement model.

Our cost function is the same for all environment types and is based on fuel consumption. We base our fuel consumption on distance (liters per grid). The cost function is implemented with rules derived from (Jameson 2009):

- Climbing adds fuel cost. (We used $c_{climb} = 0.001L$.)
- Descending saves fuel to half the cost of climbing. (We used $c_{desc} = -0.0005L$.)
- Traveling faster or slower than cruise speed decreases the fuel efficiency. We used 5 speeds with consumption rates of: $c_{speed} = [0.008, 0.007, 0.006, 0.007, 0.008]$. The middle speed is cruise speed.
- Diagonal moves (45° turn and shift maneuver) multiply the cost by $\sqrt{2}$ except for the cost of vertical movement, which is not affected by diagonal movement.

Given vector-valued aircraft states containing the variables: $\langle x, y, z, heading, speed \rangle$, the cost function $C(to, from)$ (the cost from the state "from" to the state "to") is described by:

$$C = c_{speed}(\text{to.speed}) \cdot \alpha + \beta$$

Where:

$$\alpha = \begin{cases} 1 & |\text{from.x} - \text{to.x}| \neq |\text{from.y} - \text{to.y}| \\ \sqrt{2} & \text{otherwise} \end{cases}$$

$$\beta = \begin{cases} c_{climb} & \text{to.z} - \text{from.z} > 0 \\ c_{desc} & \text{to.z} - \text{from.z} < 0 \\ 0 & \text{to.z} - \text{from.z} = 0 \end{cases}$$

Time is incremented based on speed and distance traveled. Thus, agents moving at higher speeds arrive at the next lattice point sooner and agents moving on a diagonal edge require additional traversal time.

Domain-Specific Heuristics

We formulate single-agent search goals in this environment to include constraints on both heading and speed in addition to x, y and z position. Because of the goal restrictions on heading and speed, a simple octile distance heuristic performs poorly when the direction of travel does not align with the goal heading, as extra maneuvering becomes necessary in order to get the agent pointing in the goal direction. Heading and speed constraints at the goal are important for some use-cases, such as landing or sensing with fixed sensor orientations. Some scenarios such as parachute drops or liquid dispensing may not require such restrictions.

This problem can be mitigated by the use of a perimeter heuristic designed for movement-constrained domains (Manzini 1995). We found that states within a 2-grid-cell radius of the goal and oriented within 90° of facing the goal position, regardless of the goal orientation, are able to reach the goal with the right speed and heading without leaving a 2-grid radius of the goal position. By computing a small perimeter heuristic with radius 2 and combining it with a computed estimate up to the perimeter edge, we have a strong admissible heuristic.

Given a distance estimation function $est(s, p)$ and a perimeter database lookup function $perim(p, g)$, we compute:

$$MIN_{p \in P}(est(s, p) + perim(p, c))$$

where s is the start location, g is the goal location and P is the set of all nodes on the edge of the perimeter. It is possible to select a subset of P on a case-by-case basis, however the method is domain-specific and too complex for the scope of this paper.

Abstractions

All environments used in our experiments are abstractions formed by deleting edges from the "base environment". Because edge deletions never decrease path lengths from start to goal, admissible heuristics in the base environment remain admissible in an abstraction. It should be clear that the base heuristic may be weak in the abstracted environment, necessitating unique heuristics on a per-abstraction basis.

Highway Abstractions

We form highway abstractions by converting all bi-directional edges in the x, y and z dimensions to directional ones. We formulate vertically separated highways where agents flying at the same height fly in the same direction. The highway above or below a given height has edges pointing in an adjacent horizontal direction. Thus if an agent needs to turn, it must simultaneously change altitude to enter the highway for the desired direction. Thus we retain edges that simultaneously change altitude and heading into the highway just above and below a given highway.

Because agent heading is restricted, some goal or start states may be invalid. We cannot simply restrict all start/goal states to be within the highway abstraction, thus we relax the abstraction via adaptive dimensionality (Gochev et al. 2011) for such states near the start and goal of the search:

- If an agent's start state does not have a heading that conforms to the highway based on its height, it is only allowed to make moves that put it in alignment with the highway which lies in the direction of its goal.
- If an agent's goal state is invalid with respect to the highway system, the agent is allowed to move freely when it is within 2 grid spaces from it's goal.

6 Experimental Results & Analysis

All of our experiments are for a set of k agents with random start and goal locations inside an 80x80x20 three-dimensional grid world. Each configuration was run on a set of 100 MAPF instances with random start and goal positions. Our current implementation also assumes that agents disappear, i.e. do not remain in the collision space once reaching their goal. We tested on a progressively increasing number of agents. If a problem takes longer than five minutes to terminate, we mark it as a failure and set its completion time to five minutes. All experimental results shown were run on servers with Intel Xeon processors running at 2.4GHz with 12GB of memory.

Experimental Environments

All environments were benchmarked using the regular CBS algorithm. We also benchmarked against an unbounded sub-optimal variant of ECBS called Greedy-CBS (GCBS).

Base (8-Way) Environment This is the environment described at the beginning of this section. Agents are allowed to turn, change height and/or change speed simultaneously. The maximum branching factor is 63.

4-Way Environment This abstraction is the same as the "Base" environment except turns are restricted to 90°. The maximum branching factor is 27.

Simple Environment This abstraction restricts actions to one change per movement: heading, speed or height. The maximum branching factor is 11.

Highway-8 Environment We implemented an 8-directional highway system which enforces height-separated directional highways where agent heading is forced to be congruent to height modulo 8. The maximum branching factor is 9.

Highway-4 Environment This is similar to "Highway-8" except directions are restricted to height modulo 4. The maximum branching factor is 9.

Experimental Implementations

CBS We implemented CBS+BP (Boyarski et al. 2015a) with the SIC heuristic in the high-level search with tie breaking toward CT nodes with lower conflict counts.

Greedy-CBS We consider GCBS (Barer et al. 2014) to be state-of-the-art for unbounded sub-optimal MAPF solvers which are formulated for non-holonomic vehicles. It was shown that the performance of GCBS is on par with or better than MGS1 (Standley and Korf 2011) for their experiments. We implemented GCBS-H which uses the number of conflicts heuristic at the high level. We include the performance of GCBS on the H4 environment as a benchmark because it is empirically the most performant abstraction.

Qualitative & Performance Results

Benchmark Environments Figure 4 shows the results for mean time-to-solution. These results show that when using the traditional CBS algorithm, using any of the abstractions decreases the time-to-solution relative to the "Base" environment. The "Highway-4" abstraction is the best performer, allowing us to solve roughly twice as many agents in the same amount of time when compared to the "Base" environment.

We attribute the better run-times to reduced branching factor and reduced conflicts due to highway traffic flows. Analysis of the number of conflicts that occurred during the search in each environment indicated that the time-to-solution strongly correlates to the number of conflicts.

We also saw that using GCBS on the Highway-4 environment improved performance, allowing us to solve for about 30% more agents in the same amount of time.

We also note in passing that MA-CBS was also tested and found to be less than adequate. When implemented with a pure A* meta-agent solver it performed poorly due to large branching factors, the meta-agent solver suffered due to a very large OPEN list. (Note that neither M*, ICTS were attempted due to difficulties with the formulation for non-unit time steps. EPEA* was not attempted due to the requirement of an extremely non-trivial operator selection function.)

Analysis of path lengths showed that solutions produced using the "Highway-4" environment are maximally 18% suboptimal when compared to the "Base" environment. GCBS seemed to have a minimal impact on solution quality. Table 2 shows the path quality and time to solution for various configurations. These results are from a subset of the 100 instances with 40 agents in which all environments were able to terminate with a result under the 5-minute time limit.

Layered Environments Finally, we show our results for CBS+CL. We experimented with various abstraction layerings and various switching thresholds. We found that it is generally most beneficial to set layer promotion thresholds at increments of 1 so that upon encountering a conflict, the environment is switched immediately. We found that "Highway-4→Highway-8" had the best overall performance as shown in Figure 4. These combinations allowed us to find solutions for roughly 50% more agents when compared to just using "Highway-4", and 20% more agents when Compared to GCBS on "Highway-4".

Table 1 shows CT node and A* expansion counts for a set of problems which were solvable under the time limit by all configurations. Analysis of work done at the high and low level searches revealed that CBS+CL consistently lowered the number of conflicts found in the high-level search. Low-level expansion counts are affected both by the branching factors of the mixture of environment abstractions used in the test and the number of CT nodes. For example, although $H4 \rightarrow H8 \rightarrow Base$ had fewer conflicts overall than $H4 \rightarrow H8 \rightarrow Simple$, the number of expansions is higher due to the high branching factor induced by switching some of the agents into the Base environment. Notice however that both choices are better than using $H4$ alone.

We also experimented with GCBS+CL on "Highway-4→Highway-8" and found that it provided a slight improvement over CBS+CL on the same configuration. We believe the set of conflicts resolved by GCBS and CBS+CL have overlap, and thus using them in conjunction does not give a large incremental improvement.

Table 2 shows solution qualities for the same set of problems as Table 1. We found that in the "Highway-4→Highway-8" configuration, optimality varied between 18% and 3% sub-optimal depending on the percentage of agents in the search instance that switched to the "Highway-8" environment. Not only does switching into less-constrained abstractions reduce the number of conflicts and time-to-solution, it also improves the optimality of the solutions.

7 Conclusions And Future Work

In this work we found that leveraging the strengths of highways and various granularities of environment abstractions can result in significant performance improvements. The CBS+CL algorithm leverages these strengths by refining environments in an iterative fashion during the CBS replanning stage. Using this approach, our results show that with CBS+CL we can solve roughly 2.4 times more agents than we could with regular CBS on the base environment, and about 54% more than on the H4 environment.

There are a large number of extensions possible to CBS+CL. One of the major sources of conflict in the domain comes from highly constrained resources such as landing strips. Further research into CBS+CL will investigate using additional constraints locally to avoid conflicts. Another channel for future research involves calculating respective optimality using FOCAL list search such as in (Barer et al. 2014). A final possible extension to this work would be to prepare the CBS+CL algorithm for use in cooperation with human traffic controllers by examining the importance of human understandability. Using pre-formulated highways and/or circular paths may be easier for humans to visualize and thus help build more understandable solvers.

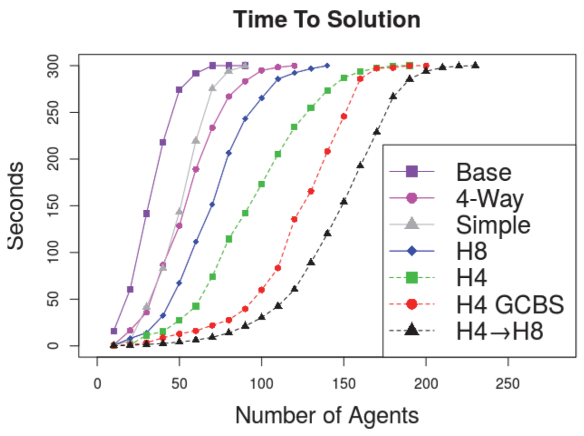

Figure 4: Comparison of performance of different environment configurations

8 Acknowledgements

This work was supported in part by NSF IUCRC grant #1439693.

Table 1: High and Low-Level Work by Configuration

Configuration	CT Nodes	Low-Level Expansions
H4→H8→Base	5.53	312099
H4→H8→Simple	5.61	294901
H4→H8→4-Way	5.61	301112
H4→H8	5.69	294712
H4→H8 GCBS	6.56	324818
H8→Simple→Base	8.62	303680
H4 GCBS	9.64	426309
H4	20.76	427492
H8 GCBS	28.62	305683
Base	29.32	2834728
4-Way	69.77	1219804
H8	84.62	471414
Simple	105.69	1372531

Table 2: Quality and Performance by Configuration

Configuration	Sol. Cost	Optimality	Time
H4→H8 GCBS	4.44	0.83	0.84
H4→H8	4.43	0.83	0.85
H4→H8→4-Way	4.43	0.83	0.88
H4→H8→Base	4.43	0.83	0.91
H4→H8→Simple	4.43	0.83	0.92
H4	4.46	0.82	0.99
H4 GCBS	4.46	0.82	1.04
H8→Simple→Base	3.78	0.97	1.42
H8 GCBS	3.79	0.97	4.36
H8	3.79	0.97	6.92
Simple	3.72	0.99	10.77
4-Way	4.28	0.86	22.43
Base	3.67	1.00	67.86

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014.*

Boyarski, E.; Felner, A.; Sharon, G.; and Stern, R. 2015a. Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015b. Icbs: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*.

Cohen, L., and Koenig, S. 2016. Bounded suboptimal multi-agent path finding using highways. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016.*

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In Lelis, L., and Stern, R., eds., *SOCS*, 2–8. AAAI Press.

Gaschnig, J. 1981. A problem similarity approach to devising heuristics: First results. In *Readings in Artificial Intelligence*.

Gochev, K.; Cohen, B. J.; Butzke, J.; Safonova, A.; and Likhachev, M. 2011. Path planning with adaptive dimensionality. In *SOCS*. AAAI Press.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced partial expansion a*. *Journal of Artificial Intelligence Research (JAIR)* 50:141–187.

Harabor, D. D., and Grastien, A. 2011. Online graph pruning for pathfinding on grid maps. In *AAAI*.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Holte, R.; Mkadmi, T.; Zimmer, R.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence (AIJ)*.

Jameson, T. 2009. A fuel consumption algorithm for unmanned aircraft systems. Technical report, DTIC Document.

Jansen, M., and Sturtevant, N. 2008. Direction maps for cooperative pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Khorshid, M. M.; Holte, R. C.; and Sturtevant, N. R. 2011. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011, Castell de Cardona, Barcelona, Spain, July 15.16, 2011.*

Manzini, G. 1995. Bida: An improved perimeter search algorithm. *Artif. Intell.*

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*.

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

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2012. Conflict-based search for optimal multi-agent path finding. In *AAAI Conference on Artificial Intelligence*, 563–569.

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

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. In *Artificial Intelligence*.

Standley, T. S., and Korf, R. E. 2011. Complete algorithms for cooperative pathfinding problems. In *IJCAI*. IJCAI/AAAI.

Sturtevant, N., and Jansen, R. 2007. An analysis of map-based abstraction and refinement. *Symposium on Abstraction, Reformulation and Approximation (SARA)* 344–358.

Wagner, G., and Choset, H. 2011. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011.*

Wang, C., and Botea, A. 2008. Fast and memory-efficient multi-agent pathfinding. In *In ICAPS*.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, 1443–1449. AAAI Press.

Yu, J., and Rus, D. 2015. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Algorithmic Foundations of Robotics XI, Springer Tracts in Advanced Robotics (STAR)*.