Extended Increasing Cost Tree Search for Non-Unit Cost Domains

Thayne T. Walker[1], **Nathan R. Sturtevant**[1], **Ariel Felner**[2]
[1] University of Denver, Denver, CO, USA
[2] Ben-Gurion University, Be'er-Sheva, Israel
thayne.walker@du.edu, sturtevant@cs.du.edu, felner@bgu.ac.il

Abstract

Multi-agent pathfinding (MAPF) has applications in navigation, robotics, games and planning. Most work on search-based optimal algorithms for MAPF has focused on simple domains with unit cost actions and unit time steps. Although these constraints keep many aspects of the algorithms simple, they also severely limit the domains that can be used. In this paper we introduce a new definition of the MAPF problem for non-unit cost and non-unit time step domains along with new multi-agent state successor generation schemes for these domains. Finally, we define an extended version of the increasing cost tree search algorithm (ICTS) for non-unit costs, with two new sub-optimal variants of ICTS: ϵ-ICTS and w-ICTS. Our experiments show that higher quality sub-optimal solutions are achievable in domains with finely discretized movement models in no more time than lower-quality, optimal solutions in domains with coarsely discretized movement models.

1 Introduction

Multi-agent pathfinding (MAPF) has applications in navigation, robotics, games, and planning. Consider the task of coordinating a fleet of robots. When planning is restricted to a 4-connected grid with unit edge costs, under the assumption of deterministic action times, all agents move and arrive at subsequent grid vertices in *lock-step*, that is, all agents' moves start at the same time and end at the same time. However, in order to save time and fuel, it would be prudent to increase the connectedness of the grid, allowing movement that is more direct toward the goal. With this change however, not all edges are of equal length, incurring non-unit costs and an agent may arrive at a vertex at a time which is not in sync with the other agents. There are many situations in which unit-time step assumptions are violated: variable speeds, graphs with variable length edges and variable wait times.

Although some centralized MAPF algorithms have been described for non-unit time step domains [Sturtevant and Buro, 2006; Thomas, Deodhare, and Murty, 2015; Walker, Chan, and Sturtevant, 2017], the effect of such domains for MAPF has not been deeply studied. Introducing non-unit time steps into a MAPF domain brings forth situations where individual agent actions have *partial time overlap*, that is, agents' actions may start and/or end at differing times, causing only a portion of the action durations to have time overlap. This paper formally defines the MAPF problem for such domains and addresses two challenges: (1) *partial time overlap* (PTO) conflict detection and (2) PTO successor generation for multi-agent states in Section 2.

The original increasing cost tree search (ICTS) algorithm [Sharon et al., 2013] is not well-formulated for non-unit costs and non-unit time steps and some reformulation is necessary. In Section 5 we introduce our new extension of the ICTS algorithm and two new bounded sub-optimal algorithms: ϵ-ICTS and w-ICTS. Finally, in Sections 6 and 7 we provide theoretical analysis and show: (1) our extended version of ICTS outperforms the current version of CBS [Sharon et al., 2015] in the domains studied, (2) empirical analysis of two PTO successor generation styles and (3) a particularly interesting result that higher quality sub-optimal solutions are achievable in domains with finely discretized movement models in no more time than lower-quality, optimal solutions in domains with coarsely discretized movement models.

2 Problem Definition

MAPF optimization is NP-hard [Yu and LaValle, 2013]. For a domain with N vertices, the k-agent state space contains $\frac{N!}{(N-k)!}$ states. With a single-agent branching factor of b_{base}, the multi-agent branching factor b is $(b_{base})^k$. Searching to a depth of d yields a total search space of $O(b^d)$ nodes, although many of them may be duplicates.

The unit cost MAPF problem is defined by a graph $G = (V, E)$ with uniform edge costs, a set of k agents, and a set of start and goal locations for each agent i: $start_i \in V$ and $goal_i \in V$ where $start_i \neq start_j, goal_i \neq goal_j$ for all $i \neq j$. A *solution* to a MAPF problem is a set of k single-agent *paths* composed of *states*. A state is a pair containing a vertex $v \in V$ and time t: $s = (v, t)$. A path for agent i is a sequence of states $\{s_i^0, ..., s_i^d\}$ where $s_i^0 = (start_i, 0)$ and $s_i^d = (goal_i, d)$ where each $(s_i^n, s_i^{n+1}) \in E$. Agents transition between states along their individual paths by means of *actions*. There are two types of actions: *movement* and *wait* actions. Movement actions transition the agent via an edge from one vertex to another $v_i^n \rightarrow v_i^{n+1}$ and strictly increase

the time component $t_i^n \rightarrow t_i^{n+1}, t_i^n < t_i^{n+1}$. Wait actions do not alter the vertex of a state but increase time. Note that because all actions have the same duration, actions occur in lock-step, that is, all agents' actions start at the same time and end at the same time.

A *feasible* solution contains only paths where no agents come into *conflict* during the entire duration of all actions in their respective paths. In pathfinding domains, a conflict is the event in which one or more agents attempt to occupy overlapping locations at the same time. Under the stated assumptions, we seek feasible solutions which minimize flowspan (the sum of individual path costs), however the proposed algorithms in this work are easily adapted for minimizing makespan (the maximum individual cost).

Adaptation for Non-Unit Costs

In the case of non-unit cost domains, we use a *weighted* graph $G = (V, E)$ with non-uniform, positive edge weights $w(v) \in \mathbb{R}_{>0}$ for each $v \in V$. Where each v is associated with unique coordinates in metric space. As in unit-cost domains, action durations are determined by edge weights. Thus, the former assumption of lock-step movement is invalidated. We call this version of the MAPF problem *MAPF$_R$*. The subscript R (for $\mathbb{R}$) denotes real-valued non-uniform edge weights.

For MAPF$_\text{R}$ we assume situated agents which occupy a nonzero area or volume. There are many ways to define this such as circles, spheres, polygons, and polygonal meshes. In this work we use circular agents with a center point and radius. When performing a wait action, agents are centered at a vertex and when in motion their center follows a straight, constant velocity motion vector in metric space between vertices v^n and v^{n+1}. We define a *collision* as the condition when one or more agents overlap at the same instant in time. Two agents may collide when traversing an edge in opposite directions, where edges intersect or when two agents are on separate edges near the same vertex.

PTO Collision Detection: Continuous-time collision detection on moving objects has been extensively studied in the fields of computational geometry, robotics, and computer graphics [Kockara et al., 2007; Jiménez, Thomas, and Torras, 2001; Ericson, 2004]. To detect a collision between agents whose actions have partial time overlap, we translate the agent with the earliest action start time forward along its motion vector to match the action start time of the second agent. Then we use an algorithm for continuous-time conflict detection between moving objects [Ericson, 2004] to obtain the projected future time of collision (if any) and check whether it will occur before the earliest ending action time.

PTO Successor Generation: We define two new methods of successor generation for a *joint-state* of k agents, $S = \{s_1, ..., s_k\}$, when using partial time overlap (PTO) actions. Let t_{min} be the minimal time over all $s_i.time$. Joint-state successors can be created via the two following methods: (1) The Cartesian product of the sets of successors $succ(s_i)$ for all s_i where $s_i.time = t_{min}$, and singleton sets $\{s_i\}$ for all s_i where $s_i.time \neq t_{min}$:

$$succ(S) = \prod_{i=0}^{k} \begin{cases} succ(s_i) & \text{if } s_i.time = t_{min} \\ \{s_i\} & \text{otherwise} \end{cases}$$

and (2) the Cartesian product of $succ(s_i)$ of only one s_i with $s_i.time = t_{min}$, and all other $\{s_i\}$. In other words: method (1) performs single-state expansions for all s_i that have minimal time and method (2) performs single-state expansion for only one s_i with minimal time. We will refer to the first method as *full branching* and the second method as *OD-style branching* because of its similarity to operator decomposition (OD) [Standley, 2010].

OD-style branching will yield a branching factor of b_{base}, however, full branching may yield a branching factor of up to $(b_{base})^k$. Although OD-style branching has a smaller joint branching factor, it will extend the depth of a search to the goal by a factor of k. Unless otherwise noted, the analysis in this paper is with reference to OD-style branching. Note that collision checks must be performed when computing the Cartesian product.

3 Background

We categorize MAPF algorithms into two types: *centralized* and *decentralized*. Decentralized algorithms [Chouhan and Niyogi, 2017; 2015; Wang and Botea, 2011; Silver, 2005] find solutions without complete knowledge of the state of other agents in the state space and hence cannot be optimal. Coupled algorithms [Sharon et al., 2015; 2013; Wagner and Choset, 2011; Standley, 2010; Sajid, Luna, and Bekris, 2012] solve for agents jointly with full knowledge of the state of all agents, hence, it is possible for centralized algorithms to have optimality guarantees. In this work we focus on centralized algorithms.

A* [Hart, Nilsson, and Raphael, 1968] and other A*-based algorithms search the *joint* state space, treating a configuration of all k agents as a state. Some A*-based algorithms for multiple agents include M* [Wagner and Choset, 2011] and Enhanced Partial-Expansion A* (EPEA*) [Goldenberg et al., 2014]. Some extensions to A* which are also relevant to ICTS include Independence Detection (ID) [Standley, 2010] and Operator Decomposition (OD) [Standley, 2010]. ID initially finds paths for individual agents and systematically merges the state spaces of conflicting agents until a feasible solution is found, often allowing a solution to be found without merging all of the agents together. OD reduces the branching factor by introducing *intermediate* states during successor generation. Although OD reduces the branching factor, it increases the depth of the search by a factor of k.

Conflict-Based Search (CBS) [Sharon et al., 2015] is a state of the art algorithm for MAPF. Meta-Agent CBS (MA-CBS) [Sharon et al., 2015] and other enhancements such as bypass and conflict prioritization for CBS (ICBS) [Boyarski et al., 2015] have been formulated. Though CBS has been applied to MAPF$_\text{R}$ [Thomas, Deodhare, and Murty, 2015; Walker, Chan, and Sturtevant, 2017], the ICBS enhancements have not been deeply studied in the context of non-unit costs.

CBS, M*, ICTS, MDD-SAT [Surynek et al., 2016] and EPEA* have been well-studied in unit-cost domains and are shown to have similar performance overall with various strengths and weaknesses depending on the characteristics of the search domain [Felner et al., 2017]. In this work, we focus on ICTS, but note that for MAPF$_\text{R}$, PTO conflict detec-

tion must be used for all of the algorithms. Additionally, PTO branching must be used for all of the algorithms except CBS.

4 ICTS Algorithm

ICTS [Sharon et al., 2013] is a two-level search algorithm.

High Level: At its high level, ICTS searches the increasing cost tree (ICT). Every node in the ICT consists of a k-ary vector $\langle C_1, ..., C_k \rangle$ which represents the question: *Is there a feasible solution where the path cost of each agent a_i is exactly C_i?* The total cost of an ICT node **I** is $C = C_1 + ... + C_k$. The objective is to find an ICT goal node of minimal C. The ICT root contains the k-ary vector of the shortest path cost from $start_i$ to $goal_i$ in G for each agent, ignoring other agents. A child in the ICT is generated by increasing the cost for one of the agents by an increment value $\delta = 1$. Figure 1(a) depicts an ICT with 3 agents. The root node contains the optimal path costs for each agent: $\langle 10, 10, 10 \rangle$. The leftmost child is created by incrementing the first element to yield $\langle 11, 10, 10 \rangle$. Dashed lines indicate duplicate children which are pruned.

An ICT node containing $\langle C_1, ..., C_k \rangle$ is a goal if there is a complete feasible solution such that the individual path cost for each agent a_i is exactly C_i. We use Δ to denote the depth of the lowest cost ICT goal node. Since all nodes at the same height have the same total cost, a breadth-first search of the ICT will find the optimal solution.

Low Level: The low-level acts as a goal test oracle for the high-level. For each ICT node generated by the high-level, the low-level is invoked. Its task is to find a feasible solution such that the cost of the individual path of agent a_i is exactly C_i. For each agent a_i, ICTS stores all single-agent paths of cost C_i as a directed acyclic graph with no duplicated edges or vertices. This compact representation is also known as a multi-value decision diagram (MDD) [Srinivasan et al., 1990]. The low-level searches the cross product of the MDDs in order to find a set of k feasible paths. If a feasible set of paths exists, the low-level returns true and the high-level halts. Otherwise, false is returned and the high-level resumes its search. In Figure 1(a) The low-level returned false for $\langle 10, 10, 10 \rangle$ (the root of the ICT) so 3 successors are generated. The next node visited by the high-level is $\langle 11, 10, 10 \rangle$. Assuming the low-level returns true at this node the high-level would then halt.

Pruning Enhancements: Several pruning enhancements have been introduced for ICTS [Sharon et al., 2013]. These techniques search for a solution for $m < k$ agents. If there exists a subset of m agents for which no valid solution exists, there cannot exist a valid solution for k agents. Thus, the low-level can immediately terminate with false.

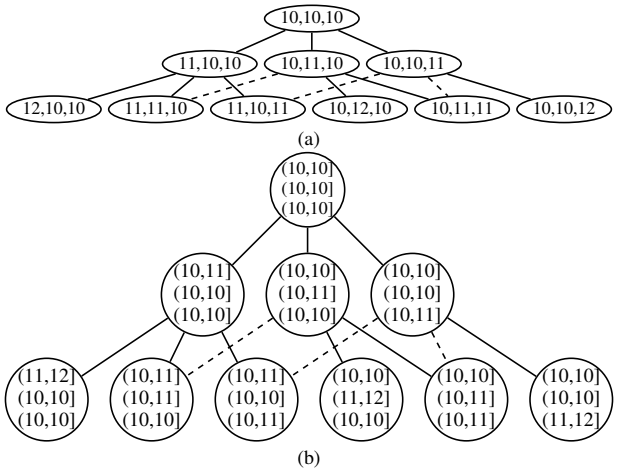

Figure 1: (a) ICT with cost vectors and (b) reformulated ICT with cost interval vectors

5 Extended ICTS For Non-Unit Costs

Several changes to the original ICTS algorithm are necessary for MAPF$_R$. The formulation of the new high level algorithm is dependent on the structure of the MDDs built by the low-level. Figure 2(a) depicts a single-agent pathfinding problem on a grid where the agent must move from the start coordinates B1 to the goal coordinates A3. Figures 2(b) and 2(c) depict the MDD for optimal paths when the grid is 4-connected, and fully connected – where every grid square is directly connected to all other grid squares. The x-axis shows cost which increases as the agent moves from the start toward the goal with Euclidean costs. Because the 4-connected MDD has unit costs, it results in a single sink node at the goal. However, when fully-connected, the resulting MDD has multiple sink nodes in the highlighted interval $(\sqrt{5}, \sqrt{5}+1]$. This leads to the simple observation that with non-unit costs, for each ICT node multiple goals may be found in the *interval* $(C, C+\delta]$.

Setting $\delta = \epsilon$, the smallest possible increment, will ensure optimal results, but may cause the ICT depth Δ to be extremely large. On the other hand, if we set δ to a large value and change the low-level search to solve an optimization problem (instead of a satisfaction problem), the result will be optimal. While this would incur a smaller ICT, our new choice of δ might push the value of C significantly past the optimal solution cost C^*, causing a large computational cost at the low-level. Since the value of C^* is usually unknown, we recommend setting δ to be a moderate value in order to mitigate the size of Δ and reduce the risk of drastically overshooting C^*.

Reformulated High Level Search

Algorithm 1 shows pseudo code for the reformulated high level search. In order to find a solution with cost C^* in the in-

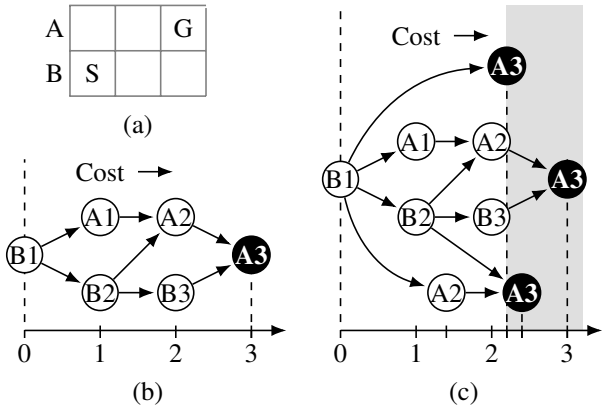

Figure 2: Problem instance (a) and associated MDD for all paths of cost between 2 and 3 for 4-connected grid (b) and fully connected grid (c)

terval $(C, C+\delta]$, we generalize ICT nodes to have a vector of cost *intervals*: $\langle (lb_1, ub_1], ..., (lb_k, ub_k] \rangle$. The root node now consists of the vector $\langle (opt_1, opt_1], ..., (opt_k, opt_k] \rangle$, where opt_i is the cost of the optimal path for agent i, ignoring other agents (line 4). A child ICT node $\mathbf{I}'$ is generated from its parent $\mathbf{I}$ by setting $\mathbf{I}'.lb_i$ to $\mathbf{I}.ub_i$ and incrementing $\mathbf{I}'.ub_i$ by δ (lines 20, 21). Figure 1(b) shows an example of an ICT with $\delta = 1$. The root node of the tree contains vectors where both lb_i and ub_i contain optimal costs of 10 (with the abuse of notation $(10, 10]$) for each agent. Now, instead of reporting the *existence* of a solution in the ICT, the low-level will detect and report the *best cost* C, in the summed-interval $(lb, ub] = (lb_1 + ... + lb_k, ub_1 + ... + ub_k]$ if a feasible solution exists, ∞ otherwise (line 10). With cost-interval ICT nodes, it is now most efficient to search the ICT in a *best-first* manner. We define the minimum-cost single agent solution in the interval $(lb_i, ub_i]$ from each MDD_i as $best_i$ and use this for a lower-bound heuristic for an ICT node: $h(\mathbf{I}) = best_1 + ... + best_k$ (line 22).

Sufficient conditions for optimality: In unit-cost domains, the first feasible solution found in the high-level is guaranteed optimal. This is not necessarily the case in non-unit cost domains. If the low-level reports that $\mathbf{I}$ in level ℓ of the ICT has a feasible solution of cost C, there are two possibilities:

1. $C = h(\mathbf{I})$: $\mathbf{I}$ is a goal (line 11). Because OPEN is ordered by $h(\mathbf{I})$, there can be no other node in the OPEN list that contains a better solution. Therefore, in this case optimality is guaranteed.
2. $C > h(\mathbf{I})$: $\mathbf{I}$ may not contain an optimal solution. We set $incumbent \leftarrow C$ (line 15), and then continue the search until a new ICT node with a lower cost is found, in which case $incumbent$ is updated again, or $h(\mathbf{I}) \geq incumbent$ (line 7), at which point we are guaranteed that $incumbent$ is the best cost.

Algorithm 1 Reformulated ICTS High Level Search

ICTS

1: Input: A MAPF instance, δ: Increment value
2: $incumbent \leftarrow \infty$: Best solution cost so far
3: $best \leftarrow \emptyset$: Best solution so far
4: Build and push the root ICT node onto OPEN
5: **while** $OPEN$ not empty **do**
6: $\quad \mathbf{I} \leftarrow OPEN.pop()$
7: $\quad$ **if** $h(\mathbf{I}) \geq incumbent$ **then**
8: $\quad\quad$ **return** $best$
9: $\quad$ **end if**
10: $\quad C \leftarrow$LOW-LEVEL($\mathbf{I}$,$incumbent$)
11: $\quad$ **if** $C = h(\mathbf{I})$ **then** ▷ Was goal found?
12: $\quad\quad best \leftarrow \mathbf{I}$
13: $\quad\quad$ **return** $best$ ▷ This is the optimal solution
14: $\quad$ **else if** $C < incumbent$ **then** ▷ New incumbent?
15: $\quad\quad incumbent \leftarrow C$
16: $\quad\quad best \leftarrow \mathbf{I}$
17: $\quad$ **else**
18: $\quad\quad$ **for** i in 1 to k **do** ▷ Generate successors
19: $\quad\quad\quad \mathbf{I}' \leftarrow \mathbf{I}$
20: $\quad\quad\quad \mathbf{I}'.lb_i \leftarrow \mathbf{I}.ub_i$
21: $\quad\quad\quad \mathbf{I}'.ub_i \leftarrow \mathbf{I}.ub_i + \delta$
22: $\quad\quad\quad$ Compute $h(\mathbf{I}')$ by building ub_i-limited MDD_i
23: $\quad\quad\quad OPEN.push(\mathbf{I}')$
24: $\quad\quad$ **end for**
25: $\quad$ **end if**
26: **end while**

Continuing the high level search until $C = h(\mathbf{I})$, or $h(\mathbf{I}) \geq incumbent$, may cause up to k-1 additional levels past ℓ to be searched to ensure optimality in the worst case. This is due to the fact that the cost difference between C^* and C could be divided between all k agents.

Reformulated Low Level Search

The low-level determines the best cost of a feasible solution for $\mathbf{I}$ if one exists. First, the low-level builds MDD_i for each agent from $start_i$ to $goal_i$ respectively. This can be done using a depth-first or breadth-first search. In this process, the best path cost in the interval $(lb_i, ub_i]$ is saved as $best_i$ for use in the heuristic function $h(\mathbf{I})$.

In order to find the best cost solution, a search of the joint k-MDD space (Algorithm 2) is performed. The root node of the low-level, $\mathbf{I}_{root} = \{MDD_1^{root}, ..., MDD_k^{root}\}$, is a joint-state containing the root nodes from MDD_1,...,MDD_k. $\mathbb{S}$, the set of joint-state successors of $\mathbf{S}$ (line 10) are generated using PTO branching as defined in Section 2. A feasible solution is found when a joint-state is visited such that all $s_i \in \mathbf{S} = goal_i$ (line 6).

The low-level continues until one of the following occurs: (1) the search is exhausted or (2) a solution that is optimal in the joint-MDD space is found based on $h(\mathbf{I})$ (line 17). If no solution was found, ∞ is returned. If a feasible solution was found with $C > h(\mathbf{I})$ (line 6) it is saved as the new $incumbent$, but it is not necessarily optimal and the low-level must continue. If the low-level reaches exhaustion after finding a feasible solution with $C > h(\mathbf{I})$, more ICT nodes may need to be explored at the high-level to ensure optimality.

Sub-Optimal Variants

ϵ-ICTS: Instead of searching the k-MDD space for an optimal solution, one can treat the low-level as a satisficing search and exit upon finding the first feasible solution (See algorithm 2 line 14). Assuming a solution with $C \geq h(\mathbf{I})$ is found at the low-level, an immediate exit may result in a significant time savings in the low-level as well as a significant pruning of ICT nodes in the high-level.

Consider the following example of near worst-case: Let $\delta = 1$ and an optimal solution exists in an ICT node at $\Delta = 12$: $\mathbf{I}_{opt} = \langle (13, 14], (13, 14], (13, 14] \rangle$, $C^* = 13.3 + 13.3 + 13.2 = 39.8$. Also let there be a sub-optimal solution in $\mathbf{I}_{sub} = \langle (19, 20], (10, 10], (10, 10] \rangle$, $C = 19.9 + 10 + 10 = 39.9$ at depth $\ell_{sub} = 10$. If we accept $\mathbf{I}_{sub}$ as a sub-optimal solution, up to k lowest levels of the ICT may be pruned.

For flowspan, the cost of each single-agent path is at most δ greater than optimal, thus the overall bound on sub-optimality is guaranteed to be no greater than $\epsilon = k\delta$. Therefore a specific ϵ can be achieved by setting δ, though very large or small values may negatively impact performance. Additionally, a more precise bound on sub-optimality is returnable as $C - h(\mathbf{I})$; the difference between the actual cost and the lower bound.

Algorithm 2 reformulated ICTS Low Level Search

JOINTDFS

```
Input: S: A joint-state, incumbent: Best cost so far
C ← Σ_{i=1}^{k} COST(s_i ∈ S)
if incumbent < C then                          ▷ Not a better solution
    return incumbent
end if
if All agents are at their goal then
    incumbent ← C                              ▷ Mark as best so far
    return incumbent
end if
𝕊 ← JOINTEXPANSION(S)                          ▷ Expand S
for S' ∈ 𝕊 do
    C = JOINTDFS(S', incumbent)
    if C < incumbent then
        if satisficing then                    ▷ Return first solution...
            return C
        end if
        if C = h(I) then                       ▷ Return optimal solution
            return C
        end if
        incumbent ← C
    end if
end for
return incumbent
```

w-ICTS: In order to obtain a weighted bound on sub-optimality, as the sub-optimality bound for ϵ-ICTS is $k\delta$, δ can be adjusted on the fly for each ICT node to guarantee a weighted bound $w>1$. We initialize the root in the same way as in the optimal algorithm, then for the generation of each ICT node $\mathbf{I}'$ thereafter, set $\delta=(w-1)h(\mathbf{I})/k$.

Pairwise Pruning Enhancement

Simple Pairwise Pruning (SPP) and Enhanced Pairwise Pruning (EPP) were proposed and outlined in the original ICTS paper [Sharon et al., 2013]. These enhancements are still valid for MAPF$_R$ with no significant changes. Our empirical results include the SPP enhancement.

6 Theoretical Analysis

The time complexity of the high level search is a combination of three factors: (1) The size of the MDDs used at the low-level and the search space required to build the MDDs; (2) The computational complexity of the low level search; (3) The computational complexity of the high level search.

MDD Size: When building an MDD for agent i at the low-level, a *cost-limit* parameter ub_i is supplied, yielding $MDD_i = (V_i, E_i)$, a time-extended directed acyclic graph for all paths from $start_i$ to $goal_i$ with cost $\leq ub_i$. ub_i is incremented by δ as the high level search proceeds, causing the size of MDD_i to increase. Eventually, V_i will span every vertex in V and after that point, $|V_i|$ will only increase by $|V|$ with each increase of ub_i. Hence in unit-cost domains, the change in $|V_i|$ between ub_i and $ub_i+\delta$ is bounded by $|V|$. For example, on a 5x5 grid, the change in $|V_i|$ as ub_i increases by 1 will never be greater than 25.

In non-unit time step domains, assuming that there are at least two discrete action durations allowed (e.g. 1 and $\sqrt{2}$ as in 8-connected grids), and at least one of the action durations shares no common denominator with the others, the increase in $|V_i|$ is upper-bounded by $|V|/r$ where r is the resolution of cost. Continuing our example with the 5x5 grid, with $r=10^{-3}$, the change in $|V_i|$ can be no greater than 25,000. Although the rate of MDD growth is still linearly bounded, there is a much steeper growth. The number of operations required to build the MDD is, in the worst case, linear in $|V|$, d and r. If a very fine resolution is supplied for r, e.g. IEEE floating-point precision, optimal ICTS may spend a lot of time to save a very small amount of cost. Fortunately, a coarse setting of r may be feasible for many applications.

Low Level Search Complexity: Because MDDs contain only ub_i-bounded paths the average branching factor for MDDs, b_{mdd}, is typically much smaller than b_{base}. Our experiments showed an average OD-style branching factor of only 1.54 at the low-level on 8x8, 8-connected grids with 10 agents. With OD-style branching, the depth of the low level search is dk where d is the max depth of all MDDs, resulting in a complexity of $O((b_{mdd})^{dk})$.

High Level Search Complexity: The number of nodes at level ℓ of the ICT (with duplicates removed) is the same as the number of terms in a multinomial coefficient [Mazur, 2010], the number of ways of adding k positive integers that add up to ℓ. Hence the size of the ICT is: $\sum_{\ell=0}^{\Delta}\binom{\ell+k-1}{k-1}=\frac{(k+\Delta)!}{k!\Delta!}=O(\text{MIN}(\Delta^k,k^\Delta))$. Assuming the number of agents is fixed, this is Δ^k. When a candidate solution is found at depth Δ, an additional $k-1$ levels of the ICT may need to be explored in the worst case to prove optimality. Therefore the overall complexity of ICTS is: $O((b_{mdd})^{dk}(\Delta+k-1)^k)$.

Sub-Optimal Variants: Let $\ell \leq \Delta$ be the shallowest level of a feasible solution in the ICT. In the best case, the k deepest levels in the ICT tree may be pruned, including all remaining nodes in level ℓ plus $k-1$ levels past ℓ. Let the average MDD depth at $\ell-1$ be d_{-1} and at $\ell+k-1$ be d_{k-1}. With OD-style branching, the amount of savings could be up to $(\ell+k-1)^k(b_{mdd})^{kd_{k-1}}-(\ell-1)^k(b_{mdd})^{kd_{-1}}$.

7 Experimental Results and Analysis

All empirical tests were conducted on a machine with 64 Intel Xeon (r) cores at 2.2GHz with 128 GB of RAM. Test sets consist of 100 random instances of the MAPF problem with varying numbers of agents on 4, 8, 16, and 32-connected grid domains also known as 2^k neighborhoods [Rivera, Hernández, and Baier, 2017] with wait actions allowed. Any instances taking longer than 300 seconds to complete were terminated and marked as a failure.

OD-Style Versus Full Branching: In order to quantify the differences between OD-style and full branching, we configured the planner in two ways: 1) Worst-case simulation, where ID is turned off and the low-level is set not to exit immediately when finding a solution, but to search the entire joint-MDD space; and 2) average case simulation, where ID is turned on and the low level is allowed to exit as soon as an optimal solution is found. We ran 100, 10-agent tests in 8x8 grids with both branching styles. Table 1 displays the mean statistics for various grid connectivity settings. In the

Worst-Case Simulation

	Search Nodes (in thousands)				Collision Checks (in millions)			
Conn.	4	8	16	32	4	8	16	32
OD	**21.6**	3.7	166.3	311.4	**1.18**	**.50**	**80.27**	**198.30**
Full	22.7	**1.6**	**102.3**	**183.5**	1.25	.51	115.68	666.31

Average-Case Simulation

	Search Nodes (in thousands)				Collision Checks (in thousands)			
OD	**.16**	**.91**	9.00	**17.16**	**4.33**	**81.05**	231.15	**371.47**
Full	.24	.96	**7.45**	17.53	4.49	91.31	**223.07**	384.90

Table 1: Comparison of average and worst-case scenario for OD-style versus full branching for 10 agents on 8x8 grids

worst-case simulation, we see a tradeoff between the number of node generations and collision checks. OD-style branching generates more nodes in non-unit time step domains, but incurs fewer overall collision checks, especially with higher branching factors. This tradeoff suggests that in general, when collision checks are expensive, OD-style branching is preferred and when node generations are expensive, full branching is preferred.

Let $\mathcal{O}$ and $\mathcal{F}$ be the sets of vertices of the search trees created at the low-level by OD-style branching and full branching respectively in the worst-case scenario. OD-style branching only uses $succ(s_i)$ for one $s_i \in S$ with $s_i.time = t_{min}$ in the Cartesian product. Hence $t_{min} \leq t'_{min}$ for all S and S' in $\mathcal{O}$. Because full branching incorporates $succ(s_i)$ for *all* s_i having minimum time, $t_{min} < t'_{min}$ for all S and S' in $\mathcal{F}$. Hence $\mathcal{F} \subseteq \mathcal{O}$, therefore $|\mathcal{F}| \leq |\mathcal{O}|$. This explains the difference in the number of search nodes in the worst-case as shown in Table 1. However, as evidenced by the statistics for the average-case simulation, the choice of branching style may not make much of a difference because the entire joint-MDD space may not be searched in all cases.

ICTS Versus A* and CBS: As an initial test, we compared A* and CBS against the reformulated ICTS algorithm. Both A* and ICTS use OD-style branching and the ID framework, solving only conflicting agents jointly. The CBS algorithm is using continuous-time collision detection and the conflict prioritization (PC) enhancement from ICBS [Boyarski et al., 2015]. Our initial analysis shows that other CBS enhancements are less effective for MAPF_R, hence are not used. All algorithms use a time resolution of r=10^{-3}. The ICTS algorithm is using an increment of $\delta = 1$ and the simple pairwise pruning enhancement.

Figure 3 shows the mean time to solution (with failure times of 300 sec averaged in) of 100 trials (y-axis) and the number of agents (x-axis). ICTS and CBS clearly dominate A* and have nearly identical run-times in 8x8 grids, but in 64x64 grids ICTS clearly dominates CBS for 8 and 16-connected domains (non-unit costs), but not in the 4-connected domain (unit costs). These results may indicate a strength in ICTS for non-unit cost domains, however, this is an area that needs further research.

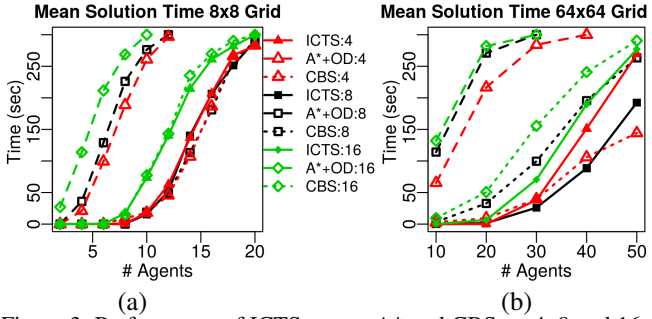

Figure 3: Performance of ICTS versus A* and CBS on 4, 8 and 16-connected grids

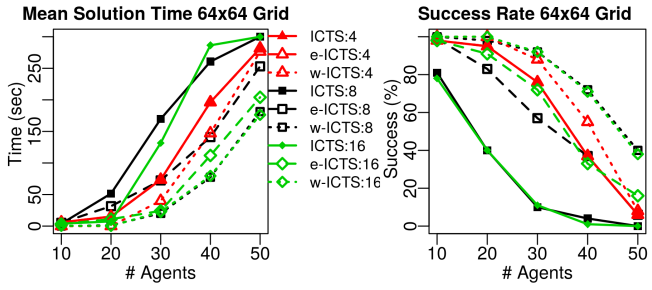

Figure 4: Performance of sub-optimal variants versus ICTS on 4, 8 and 16-connected grids

Sub-Optimal Variants: We ran both ICTS and the sub-optimal variants on 64x64 grids with $\delta = 1.0$, $w = 1.5$, and both $r = 10^{-3}$ and $r = 10^{-6}$. The results for the latter setting of r are shown in Figure 4. The results for the former setting are not as dramatic, but show the same trend. Figure 4(a) displays the mean time to solution with failure times of 300 seconds averaged in (y-axis) and number of agents (x-axis) and Figure 4(b) shows the percentage of problems solved in under 300 seconds. Run times for both ϵ-ICTS and w-ICTS are better for higher branching factor domains compared to run times in 4-connected domains.

Table 2 displays partial results from Figure 4 for 30 agents. Note that not only is the time to solution in the sub-optimal algorithms lower than for 4-connected grids, but the solution quality is better. For example ϵ-ICTS on 16-connected grids yields a mean 21% improvement on solution quality *and* a 3× improvement on solution time versus ϵ-ICTS on 4-connected grids. This surprising result suggests that higher quality paths can be achieved in less time by using finer discretization in agent actions and using a sub-optimal solver.

8 Conclusions And Future Work

This paper contributes a new definition of MAPF called MAPF_R for non-unit cost domains, definitions for PTO successor generation, a formulation of ICTS for MAPF_R and new bounded sub-optimal variants for ICTS. In future work we will explore the effect of non-unit cost domains on other

Performance on 2^k Neighborhoods

	Cost				Time (sec)			
Conn.	4	8	16	32	4	8	16	32
ICTS	1283	1013	1012	1005	**73**	170	132	181
ϵ-ICTS	1283	1041	1013	1010	73	72	**24**	54
w-ICTS	1284	1051	1051	1020	40	**20**	22	71

Ratio Versus Optimal ICTS, 4-Connected

	Cost ratio				Time ratio			
ICTS	1.0	.79	.79	.78	1.0	2.33	1.81	2.48
ϵ-ICTS	1.0	.81	.79	.79	1.0	.99	.33	.74
w-ICTS	1.0	.82	.82	.80	.55	.27	.30	.97

Table 2: ICTS and sub-optimal variant performance for various branching factors planning for 30 agents on a 64x64 grid

MAPF algorithms.

References

[Boyarski et al., 2015] Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. Icbs: Improved conflict-based search algorithm for multi-agent pathfinding. In *IJCAI*, 223–225.

[Chouhan and Niyogi, 2015] Chouhan, S. S., and Niyogi, R. 2015. Dmapp: A distributed multi-agent path planning algorithm. In *Australasian Joint Conference on Artificial Intelligence*, 123–135. Springer.

[Chouhan and Niyogi, 2017] Chouhan, S. S., and Niyogi, R. 2017. Dimpp: a complete distributed algorithm for multi-agent path planning. *Journal of Experimental & Theoretical Artificial Intelligence* 1–20.

[Ericson, 2004] Ericson, C. 2004. *Real-time collision detection*. CRC Press.

[Felner et al., 2017] Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Proceedings of the Tenth International Symposium on Combinatorial Search, Edited by Alex Fukunaga and Akihiro Kishimoto, 16-17 June 2017, Pittsburgh, Pennsylvania, USA.*, 29–37.

[Goldenberg et al., 2014] 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.

[Hart, Nilsson, and Raphael, 1968] Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

[Jiménez, Thomas, and Torras, 2001] Jiménez, P.; Thomas, F.; and Torras, C. 2001. 3d collision detection: a survey. *Computers & Graphics* 25(2):269–285.

[Kockara et al., 2007] Kockara, S.; Halic, T.; Iqbal, K.; Bayrak, C.; and Rowe, R. 2007. Collision detection: A survey. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, 4046–4051. IEEE.

[Mazur, 2010] Mazur, D. R. 2010. Combinatorics : a guided tour. 240–243.

[Rivera, Hernández, and Baier, 2017] Rivera, N.; Hernández, C.; and Baier, J. A. 2017. Grid pathfinding on the 2k neighborhoods. In *AAAI*, 891–897.

[Sajid, Luna, and Bekris, 2012] Sajid, Q.; Luna, R.; and Bekris, K. E. 2012. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *SOCS*, 88–96.

[Sharon et al., 2013] 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 et al., 2015] Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

[Silver, 2005] Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, volume 1, 117–122.

[Srinivasan et al., 1990] Srinivasan, A.; Ham, T.; Malik, S.; and Brayton, R. K. 1990. Algorithms for discrete function manipulation. In *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, 92–95. IEEE.

[Standley, 2010] Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, volume 1, 28–29. Atlanta, GA.

[Sturtevant and Buro, 2006] Sturtevant, N., and Buro, M. 2006. Improving collaborative pathfinding using map abstraction. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 80–85.

[Surynek et al., 2016] Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, 810–818.

[Thomas, Deodhare, and Murty, 2015] Thomas, S.; Deodhare, D.; and Murty, M. N. 2015. Extended conflict-based search for the convoy movement problem. *IEEE Intelligent Systems* 30(6):60–70.

[Wagner and Choset, 2011] 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*, 3260–3267.

[Walker, Chan, and Sturtevant, 2017] Walker, T. T.; Chan, D.; and Sturtevant, N. R. 2017. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 316–324.

[Wang and Botea, 2011] Wang, K.-H. C., and Botea, A. 2011. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research* 42:55–90.

[Yu and LaValle, 2013] 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.