Optimal and Bounded-Suboptimal Multi-Agent Motion Planning

Liron Cohen, Tansel Uras, T. K. Satish Kumar and **Sven Koenig**
University of Southern California
{lironcoh, turas}@usc.edu, tkskwork@gmail.com, skoenig@usc.edu

Abstract

Multi-Agent Motion Planning (MAMP) is the task of finding collision-free kinodynamically feasible plans for agents from start to goal states. While MAMP is of significant practical importance, existing solvers are either incomplete, inefficient or rely on simplifying assumptions. For example, Multi-Agent Path Finding (MAPF) solvers conventionally assume discrete timesteps and rectilinear movement of agents between neighboring vertices of a graph. In this paper, we develop MAMP solvers that obviate these simplifying assumptions and yet generalize the core ideas of state-of-the-art MAPF solvers. Specifically, since different motions may take arbitrarily different durations, MAMP solvers need to efficiently reason with continuous time and arbitrary wait durations. To do so, we adapt (Enhanced) Conflict-Based Search to continuous time and develop a novel bounded-suboptimal extension of Safe Interval Path Planning, called Soft Collision Interval Path Planning. On the theoretical side, we justify the completeness, optimality and bounded-suboptimality of our MAMP solvers. On the experimental side, we show that our MAMP solvers are more efficient with increasing suboptimality bounds.

Introduction

Multi-Agent Motion Planning (MAMP) is the task of finding collision-free kinodynamically feasible plans for agents in a shared environment. Each agent has a unique start and a unique goal state. Real-world applications of MAMP include autonomous aircraft towing vehicles (Morris *et al.* 2016), autonomous non-holonomic vehicles (such as forklifts) in industrial applications (Cirillo *et al.* 2014) and traffic management systems for unmanned drones (Prevot *et al.* 2016).

While MAMP is of significant practical importance, existing formulations rely on simplifying assumptions. For example, the Multi-Agent Path Finding (MAPF) problem uses a formulation in which: (1) the environment is captured by a graph with vertices representing locations and edges representing straight-line movements between vertices; and (2) time is discretized into synchronized timesteps. Although MAPF problems are motivated by real-world applications, their solutions may not be kinodynamically feasible for real agents, such as cars or drones, for two major reasons. First, rectilinear movement cannot be executed on agents with non-holonomic constraints. Second, different agents may have different motions of arbitrarily different durations and thus cannot be easily synchronized. To partially alleviate this problem, it is possible to post-process a MAPF solution using simple temporal networks and continuous refinement of trajectories (Hoenig *et al.* 2016; 2018). However, these methods may produce arbitrarily suboptimal plans even if they post-process optimal MAPF solutions.

One way to address the limitations of the post-processing strategy is to work with an enriched formulation of the MAMP problem. A first step in this direction is the formulation of the MAPF$_R$ problem (Walker *et al.* 2018), which is identical to the MAPF problem but allows positive non-uniform edge weights to represent different action durations. However, in MAPF$_R$, vertices still represent locations in metric space and movements are still rectilinear with uniform velocities.

In this paper, we formulate a richer MAMP problem in which vertices represent states and directed edges represent kinodynamically feasible motions. A state corresponds to a point in the configuration space of an agent. For example, it may include the (x, y, z)-coordinates, orientation, steering angle, velocity or other features that characterize it. An edge from one state to another corresponds to a feasible motion between them. The weight of the edge represents the duration of the motion. The environment is discretized into cells. Each edge specifies a list of *swept* cells, each cell with an associated timeinterval during which the agent executing the motion occupies it. As a consequence, in our formulation, an agent is allowed to be of any geometric shape. This formulation naturally lends itself to reasoning over state lattices (Pivtoraiko *et al.* 2009), probabilistic roadmaps (PRMs) (Kavraki *et al.* 1996) and rapidly exploring random trees (RRTs) (Kuffner and LaValle 2000).

Computationally, the richer formulation of the MAMP problem is more challenging, and all current solvers for it are either incomplete or inefficient (Cirillo *et al.* 2014; Salvado *et al.* 2018; Saha *et al.* 2016). In this paper, we develop a significantly more efficient and provably effective

MAMP solver by drawing inspiration from the success of a state-of-the-art MAPF solver, called Conflict-Based Search (CBS) (Sharon *et al.* 2015). While CBS is not directly applicable to the MAMP problem, we successfully extend its core ideas to the richer MAMP domain. To do so, (1) we adapt (Enhanced) CBS (Barer *et al.* 2014) to efficiently reason with continuous time and arbitrary wait durations; (2) we introduce an efficient implementation of reservation tables using interval maps (Cormen *et al.* 2009); and (3) we develop a novel bounded-suboptimal extension of Safe Interval Path Planning (SIPP) (Phillips and Likhachev 2011), called Soft Collision Interval Path Planning (SCIPP). On the theoretical side, we justify the completeness, optimality and bounded-suboptimality of our MAMP solvers on state lattices. On the experimental side, we show that our MAMP solvers scale well with increasing suboptimality bounds.

Background and Related Work

The MAPF Problem

The MAPF problem is defined on a graph $G = (V, E)$ with agents $1, \ldots, K$. Each agent j has unique start and goal vertices $s^j, g^j \in V$. At each discrete timestep, each agent can either move to a neighboring vertex or wait at its current vertex, both with unit cost. A *solution* is a set of feasible paths, one path $\{s_0^j, \ldots, s_{T_j}^j, s_{T_j+1}^j, \ldots\}$ for each agent j, such that no two paths collide. A path for agent j is *feasible* iff $s_0^j = s^j$, there exists a smallest T_j such that $s_t^j = g^j$ for $t \geq T_j$, and, for all $t \in \{0, 1, \ldots, T_j - 1\}, \langle s_t^j, s_{t+1}^j \rangle \in E$ or $s_t^j = s_{t+1}^j$. A *collision* between agents j and k is either a *vertex collision* (j, k, s, t) with $s = s_t^j = s_t^k$, or an *edge collision* (j, k, s_1, s_2, t) with $s_1 = s_t^j = s_{t+1}^k$ and $s_2 = s_{t+1}^j = s_t^k$. The travel time of agent j is T_j. For the commonly used objective of minimizing $\sum_{j=1}^{K} T_j$, the MAPF problem is NP-hard (Yu and LaValle 2013b).

MAPF and Related Solvers

Numerous optimal MAPF solvers have been developed in recent years, including reduction-based solvers (Yu and LaValle 2013a; Surynek *et al.* 2016), A*-based solvers (Standley 2010) and dynamically coupled search-based solvers, like M* (Wagner and Choset 2015) and CBS (Sharon *et al.* 2015). See (Felner *et al.* 2017) for a survey.

Various generalizations of the MAPF problem have also been studied. In Any-Angle MAPF (Yakovlev and Andreychuk 2017), agents can move in arbitrary directions but along straight lines. In (Walker *et al.* 2018), a solver based on ICTS (Sharon *et al.* 2013) is presented for the MAPF$_R$ problem. While it can successfully handle arbitrary action durations, it produces rectilinear motions of agents and relies on geometry-based collision checking that is often computationally expensive. In CBS with Constraint Layering (CBS+CL) (Walker *et al.* 2017), a hierarchy of edge subgraphs is used with CBS to plan curvilinear paths for agents. However, CBS+CL sacrifices optimality and does not efficiently reason about arbitrary wait durations.

CBS and ECBS

We now describe CBS and ECBS in more detail since our MAMP solvers generalize them.

CBS is an optimal MAPF solver. It performs high-level and low-level searches. Each high-level node contains a set of constraints and, for each agent, a feasible path that respects the constraints. The high-level root node has no constraints. The high-level search of CBS is a best-first search that uses the costs of the high-level nodes as their f-values. The cost of a high-level node is the sum of the travel times along the agents' paths it contains. When CBS expands a high-level node N, it checks whether the node is a goal node. A high-level node is a goal node iff none of its paths collide. If N is a goal node, then CBS terminates successfully and outputs the paths in N as solution. Otherwise, at least two paths collide. CBS chooses a collision to resolve and generates two high-level children of N, called N_1 and N_2. Both N_1 and N_2 inherit the constraints of N. If the chosen collision is a vertex collision (j, k, s, t), then CBS adds the vertex constraint (j, s, t) to N_1 (that prohibits agent j from occupying vertex s at timestep t) and the vertex constraint (k, s, t) to N_2. If the chosen collision is an edge collision (j, k, s_1, s_2, t), then CBS adds the edge constraint (j, s_1, s_2, t) to N_1 (that prohibits agent j from moving from vertex s_1 to vertex s_2 between timesteps t and $t+1$) and the edge constraint (k, s_2, s_1, t) to N_2. During the generation of a high-level node N, CBS performs a low-level search for the agent i affected by the newly added constraint. The low-level search for agent i is a (best-first) A* search that ignores all other agents and finds a minimum-cost path from the start vertex of agent i to its goal vertex that is both feasible and respects the constraints of N that involve agent i.

ECBS(w) is a w-suboptimal variant of CBS whose high-level and low-level searches are focal searches rather than best-first searches. A focal search (Pearl and Kim 1982), like A*, uses an OPEN list whose nodes n are sorted in increasing order of their f-values $f(n) = g(n)+h(n)$. Unlike A*, a focal search with suboptimality factor w also uses a FOCAL list of all nodes currently in OPEN whose f-values are no larger than w times $f_{\min}$, the currently smallest f-value of any node in OPEN. The nodes in FOCAL are sorted in increasing order according to *secondary* heuristic values. A* expands a node in OPEN with the smallest f-value, but a focal search instead expands a node in FOCAL with the smallest secondary heuristic value. If $h(n)$ is admissible, then focal search is guaranteed to be w-suboptimal. The secondary heuristic values do not have to be consistent (or admissible). The high-level and low-level searches of ECBS(w) are focal searches. During the generation of a high-level node N, ECBS(w) performs a low-level focal search with OPEN list OPEN$^i(N)$ and FOCAL list FOCAL$^i(N)$ for the agent i affected by the newly added constraint. The high-level and low-level focal searches of ECBS(w) use measures related to the number of collisions as secondary heuristic values.

State Lattices

State lattices (Pivtoraiko *et al.* 2009) are extensions of grids that are able to model kinodynamic motion constraints and

are therefore well suited to planning for agents with limited maneuverability (such as non-holonomic mobile robots). A state lattice is constructed by discretizing the configuration space into a high-dimensional grid and connecting the cells of the grid with motion primitives. A motion primitive models kinodynamically feasible motions of the agent. A state in a state lattice is a tuple of the form $(x, y, z, \theta, v, \ldots)$, where x, y and z are the coordinates of the agent's center, θ is the agent's orientation, v is the agent's velocity, etc. An edge in a state lattice represents a motion primitive and is associated with a duration and a list of cells that are swept by the agent when the motion is executed. Motion primitives have successfully been used for autonomous cars in DARPA's urban challenge (Ferguson *et al.* 2008) and quadrotors (Liu *et al.* 2018). A state lattice facilitates the application of heuristic search algorithms to find optimal or bounded suboptimal trajectories[1].

SIPP

SIPP (Phillips and Likhachev 2011) is a search-based single-agent path planner designed to handle dynamic obstacles efficiently. In SIPP, each state is associated with a fixed list of *safe timeintervals* during which an agent in that state does not collide with any dynamic obstacles. Timeintervals allow SIPP to reason about wait durations "in bulk," making it more efficient than A* in the presence of arbitrary wait durations. SIPP has already been successfully used for Multi-Agent Any-Angle Path Finding (Yakovlev and Andreychuk 2017) and Multi-Agent Pickup and Delivery problems (Ma *et al.* 2019).

Problem Formulation

We define the MAMP problem to be a generalization of the MAPF problem. Thus, the objective of minimizing the sum of travel times in the MAMP problem, as defined below, is also NP-hard. Unlike MAPF, the MAMP problem is posed on states instead of locations. A state specifies discretized values of an agent's location, orientation, velocity, etc. An edge represents a kinodynamically feasible motion of a given arbitrary duration. The sequence of motions in a feasible plan leads an agent from its start state to its goal state. An agent is allowed to be of any geometric shape, implicitly specified by a set of occupied cells.

We formally define the MAMP problem as follows. We are given an environment represented by a set of cells $\mathcal{C}$. We are given agents $1, \ldots, K$, each with an associated graph $G^j = (V^j, E^j)$ and start and goal vertices, $s^j, g^j \in V^j$. Each vertex $s \in V^j$ represents a *state* and is associated with a list of cells $\{c^s_1, \ldots, c^s_{m(s)}\} \subseteq \mathcal{C}$ occupied by the agent while at s. Each edge $e \in E^j$ represents a *motion* and has an associated weight $w(e) > 0$, that represents its duration. e is also associated with a multiset of cells $\{c^e_1, \ldots, c^e_{m(e)}\} \subseteq \mathcal{C}$. Each cell c^e_i is associated with a timeinterval $[lb^e_i, ub^e_i]$ during which it is swept by agent j after the start of execution of the motion represented by e. Thus $\min_{1 \le i \le m(e)} lb^e_i = 0$ and $\max_{1 \le i \le m(e)} ub^e_i = w(e)$.

[1] optimality with respect to the state lattice discretization

A sequence $\pi^j = \{\langle e^j_1, t^j_1 \rangle, \ldots, \langle e^j_{T_j}, t^j_{T_j} \rangle\}$ is a plan for agent j, where $e^j_i = (s^j_{i-1}, s^j_i) \in E^j$ and $t^j_i \ge 0$ is the beginning time of the execution of e^j_i. π^j is *feasible* iff $e^j_1 \ldots, e^j_{T_j}$ is a path from s^j to g^j in G^j and, for all $i \in \{2, \ldots, T_j\}$, $t^j_i \ge t^j_{i-1} + w(e^j_{i-1})$. This means that agent j waits at state s^j_{i-1} and therefore occupies cells $\{c^s_1, \ldots, c^s_{m(s)}\}$ for $s = s^j_{i-1}$ during the timeinterval $[t^j_{i-1} + w(e^j_{i-1}), t^j_i]$.[2] We assume that agent j occupies $\{c^s_1, \ldots, c^s_{m(s)}\}$ for $s = g^j$ during the timeinterval $[t^j_{T_j} + w(e^j_{T_j}), \infty]$. The travel time of agent j in π^j is given by $t^j_{T_j} + w(e^j_{T_j})$. A *collision* between two agents occurs iff the timeintervals in which they sweep or occupy the same cell overlap and has a duration larger than 0. A *solution* to the MAMP problem is a set of feasible plans such that no two agents collide. We focus on minimizing the sum of travel times of all agents.

In this paper, we consider a state lattice representation of the MAMP problem, even though one could also consider a PRM or RRT representation.

ECBS for MAMP

In this section, we present ECBS-CT,[3] a generalization of ECBS for the MAMP problem. Algorithm 1 shows the pseudocode for the high-level search of ECBS-CT. It takes as input an MAMP instance and a suboptimality bound $w \ge 1$. ECBS-CT generates a solution that has a cost of no more than w times the optimal cost. Thus, for $w = 1$, ECBS-CT is optimal and essentially generalizes CBS for the MAMP problem.

On lines 1-2, the high-level root node is initialized using a low-level search for each agent. In ECBS-CT, the low-level search uses SCIPP. The main loop on lines 3-12 performs a focal search. On lines 11-12, FOCAL is appropriately maintained to include all relevant nodes from OPEN if $f_{\min}$ changes. The secondary heuristic value of a generated high-level node is defined to be the total duration in which two or more agents collide. As discussed in the next subsection, it can be efficiently computed while the reservation table is updated on line 10.

On line 6, a collision at cell c starting at time lb and ending at time ub, $(c, [lb, ub])$, between two agents j and k is identified. We focus on resolving the earliest collision among all agents in high-level node N. As discussed in the next subsection, this earliest collision can be efficiently identified using the reservation table. Resolving the earliest collision is known to be beneficial in the CBS framework (Sharon *et al.* 2015). The collision is resolved on line 8 by posting a constraint on cell c at a *timepoint* $\tau \in [lb, ub]$. This constraint (c, τ) prohibits the corresponding agent from being at cell c at timepoint τ. Two questions that arise in this context are: (1) "Why is a timepoint used instead of a timeinterval?" and (2) "What should the value of τ be?"

The answer to the first question relates to the requirements of CBS (ECBS) to guarantee optimality (w-suboptimality).

[2] Waiting can be conditioned on the velocity being zero.

[3] CT stands for continuous time.

Algorithm 1: ECBS-CT (High-Level Search)

Input: MAMP instance, $w \geq 1$.
Output: A w-suboptimal solution.

```
Initialize root node with a plan for each agent using SCIPP.
Insert the root node into OPEN and FOCAL.
while FOCAL ≠ ∅ do
    N ← Pop(FOCAL).
    if N is a solution then return N.
    Identify a collision (c, [lb, ub]) between agents j and k
      at cell c ∈ C during timeinterval [lb, ub].
    Identify a timepoint τ ∈ [lb, ub].
    Generate two successor nodes, N^j and N^k for agents j
      and k, each imposing the additional constraint (c, τ).
    Replan using SCIPP for agents j and k in N^j and N^k.
    Update the reservation tables in N^j and N^k.
    Insert N^j and N^k into OPEN and conditionally to
      FOCAL.
    Update FOCAL if necessary.
return no solution.
```

The proof of optimality (w-suboptimality) relies on the property that any solution which obeys the constraints of a high-level node N also obeys the constraints of at least one of its high-level successor nodes N^j or N^k. This is directly analogous to the proofs of Lemma 2 in (Sharon *et al.* 2015) and Theorem 1 in (Barer *et al.* 2014). However, if the constraint specifies a timeinterval $[lb', ub'] \subseteq [lb, ub]$ instead of a timepoint, this property no longer holds. In particular, if an optimal solution includes agent j being at c during timeinterval $[lb', (lb' + ub')/2)$ and agent k being at c during timeinterval $((lb' + ub')/2, ub']$, it is spuriously eliminated.

The answer to the second question relates to Zeno behaviors (Zhang *et al.* 2001). Zeno behavior is a phenomenon in hybrid (discrete-continuous) systems, where the system undergoes an infinite number of discrete transitions in a finite duration of time. Specifically in our case, if $\tau < ub$, agents j and k can satisfy their respective constraints by waiting for $\tau - lb$ time units before colliding again at c during the non-singleton timeinterval $(\tau, ub]$. Similarly, in a future iteration, they may collide yet again during the non-singleton timeinterval $(\tau', ub]$ for some $ub > \tau' > \tau$. Therefore, any strategy for choosing the value of τ other than setting it to the upper bound of the timeinterval (ub) results in Zeno behavior.

Reservation Table

In ECBS, a reservation table is simply a set of discrete timesteps specified for each cell in the environment during which this cell is occupied by some agent. In ECBS-CT, the discrete timesteps are replaced by timeintervals. A timeinterval $[lb, ub]$ in the reservation table for a cell c has an associated value v that indicates the number of agents occupying c during $[lb, ub]$. Figure 1 illustrates the reservation table for a cell occupied by three agents during overlapping timeintervals.

The reservation table we propose maintains an *interval map* (Cormen *et al.* 2009) for each cell c in the environment. An interval map is a data structure that can efficiently manage intervals. It supports efficient insertion, deletion, search and aggregate-on-overlap operations. The insertion (deletion) operation can be done in logarithmic time unless the query interval overlaps with all existing intervals. Fortunately, this rarely happens in MAMP problems due to the locality of motion primitives. The aggregate-on-overlap operation combines (separates) the associated values of intersecting intervals on insertion (deletion) with the same complexity as the insertion (deletion) operation. Figure 1 also illustrates the aggregate-on-overlap operation.

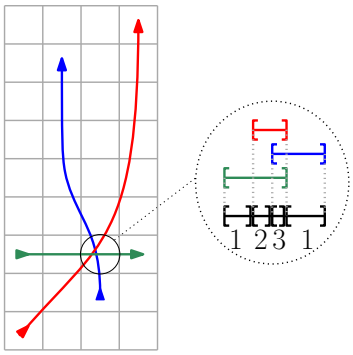


Figure 1: Shows three agents, red, blue and green, sweeping the encircled cell during overlapping timeintervals. The aggregate-on-overlap operation produces 4 timeintervals, each with an associated number of collisions, as indicated in black.

In addition to insertion, deletion, search and aggregate-on-overlap, the usage of interval maps for the reservation table may also facilitate an efficient detection of the earliest collision in the high-level search. Given the reservation table, one can simply iterate over all cells and extract the earliest collision (which is linear in the number of cells in the environment). This may be significantly faster than iterating over all pairs of plans (which is square of the number of agents times the maximal number of swept cells in any plan). Moreover, the usage of interval maps for the reservation table also facilitates an efficient computation of secondary heuristic values for focal search in the high-level search. Generating a high-level node involves replanning for one agent. The reservation table of the generated child node can be simply copied from its parent node and updated by: 1) deleting the timeintervals from the relevant cells according to the agent's previous plan; and 2) inserting the timeintervals to the relevant cells according to the agent's new plan. While updating the reservation table, we can also update the total duration in which two or more agents collide, which is used as the secondary heuristic value in the high-level search. Finally, the usage of interval maps for the reservation table also facilitates an efficient computation of secondary heuristic values for focal search in the low-level search. Generating a low-level node involves looking at all swept cells associated with the motion primitive that generated it and checking the number of collisions during different timeintervals. This process is explained in more detail in the next section.

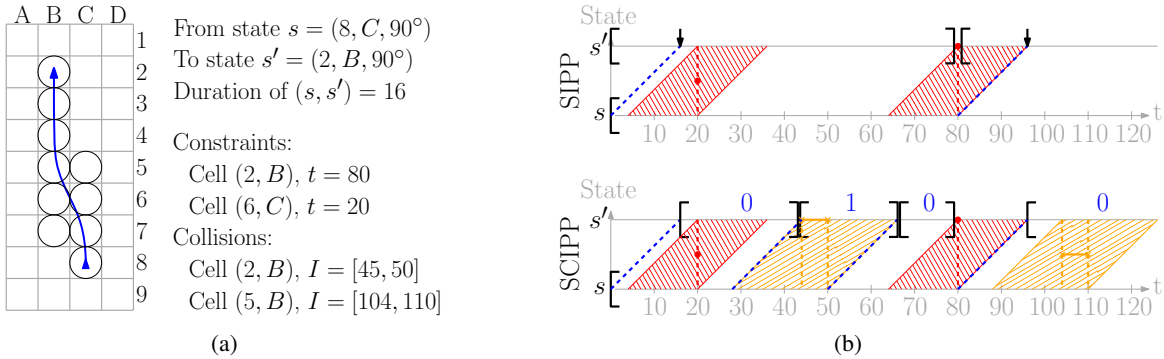


Figure 2: (a) shows a motion primitive (s, s') in blue. (b) illustrates the difference between SIPP and SCIPP for (a).

Algorithm 2: SCIPP

Input: Start and goal states (s^j and g^j), constraints, reservation table, $w \geq 1$.
Output: A w-suboptimal feasible plan from s^j to g^j.

```
root nodes ← InitializeNodes(s^j).
Insert root nodes into OPEN and conditionally to FOCAL.
while FOCAL ≠ ∅ do
    n ← Pop(FOCAL).
    if n is a goal node then return plan.
    for each n' ∈ GenerateSuccessorNodes(n) do
        N ← Merge n' into GENERATEDLIST.
        Insert/Update OPEN and FOCAL according to N.
    Update FOCAL if necessary.
return no solution.
```

SCIPP

Since focal search with discrete timesteps is vital for bounded-suboptimal MAPF solvers, an efficient generalization of it to continuous time is required for ECBS-CT. While SIPP can be used to efficiently reason about continuous time and the constraints specified by a high-level node, it unfortunately cannot be used to reason about the collisions specified in the reservation table of the high-level node. Therefore, SIPP is unsuitable for focal search. Instead, we develop SCIPP for the low-level focal search of ECBS-CT. SCIPP not only efficiently reasons about continuous time and constraints but also about collisions and derives secondary heuristic values from them.

Given a start state s^j, a goal state g^j and a list of constraints for each cell c, SIPP finds a feasible plan from s^j to g^j with minimum arrival time at g^j. This plan also guarantees that no cell is swept outside of its safe timeintervals. Here, the dynamic obstacles are simply the constraints, that is, the safe timeintervals associated with c are all timeintervals other than the specified timepoints of the constraints for c. In SIPP, a node n represents an $(s, [lb, ub])$ pair, where s is a state and $[lb, ub]$ is a safe timeinterval of s. The successor n' of n represents a transition from s to s' via a legal motion.[4] $g(n)$ represents the earliest arrival time to s within $[lb, ub]$. Thus, when generating n', $g(n')$ is updated to $g(n) + d$ if $g(n') > g(n) + d$, where d is the duration of the legal motion.

The example in Figure 2(a) shows an environment with 36 cells (grey squares) and a motion primitive (s, s') in blue from s to s'. While waiting at s and s', the agent occupies cells $(8, C)$ and $(2, B)$, respectively. Black circles depict the swept cells of (s, s'). For the sake of simplicity, let us assume that the agent occupies all of the swept cells during the entire execution of (s, s'). The upper part of Figure 2(b) illustrates how SIPP works when generating safe timeintervals for (s, s'). The x-axis represents time, and the y-axis represents different states. A red dot between s and s' represents a constraint at the time of its x-coordinate at an intermediate swept cell of (s, s'), and a red dot on the horizontal line for s' represents a constraint at the time of its x-coordinate at a destination cell of (s, s'). The red regions indicate execution times of (s, s') that violate a constraint; and the slope of the parallelograms indicate the duration of (s, s'). The safe timeinterval for s is $[0, \infty]$ and, due to the constraint on s' at timepoint 80, the safe timeintervals for s' are $[0, 79]$ and $[81, \infty]$. The black arrows indicate the earliest arrival times within these timeintervals, serving as g-values. Note that the agent can arrive to s' as early as 16 and thus avoid violating the constraint on the intermediate cell at timepoint 20.

Algorithm 2 presents SCIPP, a generalized version of SIPP suitable for focal search. It takes as input a start state s^j, a goal state g^j, a list of constraints for each cell c and a reservation table, as specified in the high-level node. It outputs a w-suboptimal feasible plan from s^j to g^j for the specified value of w without violating any constraint. As in SIPP, each node represents an $(s, [lb, ub])$ pair. GENERATEDLIST is a hash table of all generated nodes. It maps a state s to a list $L(s)$ of all generated nodes having s as their state. The timeintervals of all nodes in $L(s)$ are maintained to be disjoint. For each timeinterval $[lb, ub]$ of a node $n \in L(s)$, lb represents the earliest possible arrival time to n via n's predecessor without violating any constraint and with a con-

[4] A motion is legal iff it sweeps each of its associated cells only during their safe timeintervals.

stant number of collisions. Thus, $g(n) = lb.\ ub - lb$ represents the maximum wait duration at n while retaining the same number of collisions and without violating any constraint. The secondary heuristic value of n is its number of collisions, defined to be the number of cells in which the plan from s^j to s collides with any other agent's plan, as specified in the reservation table.

The main loop on lines 3-9 performs a focal search after the initialization on lines 1-2. $InitializeNodes(s^j)$ on line 1 generates a list of nodes corresponding to disjoint timeintervals between 0 and the earliest constraint imposed on any cell associated with s^j. Each timeinterval in this list has a constant number of collisions. $GenerateSuccessorNodes(n)$ on line 6 generates all successors of n. Like in SIPP, n' represents a transition from s to s' via a legal motion. In SCIPP, the timeinterval of n' may have to be split into disjoint timeintervals so that the number of collisions associated with each timeinterval remains constant. Thus, more nodes may have to be created, one for each disjoint timeinterval. The disjoint timeintervals are created by first generating "arrival" timeintervals with respect to the constraints and collisions on the swept cells of the motion, and then extending these arrival timeintervals with respect to the constraints and collisions on the cells associated with s' in order to reason about possibly waiting at s'. The upper part of Figure 2(b) illustrates how SCIPP works when generating disjoint timeintervals for (s, s'). Like in SIPP, constraints are represented in red. In addition, a yellow horizontal line between s and s' represents a collision during the timeinterval between its end-point x-coordinates at an intermediate swept cell of (s, s'), and a yellow horizontal line on the horizontal line for s' represents a collision during the timeinterval between its end-point x-coordinates at a destination cell of (s, s'). Note that the disjoint timeintervals of s' has a constant number of collisions, depicted with blue labels.

The mechanism of duplicate detection in focal search is implemented using $Merge$ on lines 7-8. The subtlety for continuous time is the possibility of a generated node n' having the same state s' as a different node n'' already in GENERATEDLIST such that their timeintervals overlap. Regardless of the different ways in which the two timeintervals may overlap, the $Merge$ process restores the invariant that all nodes in $L(s')$ are disjoint and each has a constant number of collisions. The $Merge$ process can either: (1) add the newly created nodes from $GenerateSuccessorNodes$ to GENERATEDLIST and insert them into OPEN and conditionally into FOCAL; or (2) update the timeinterval of the existing nodes in GENERATEDLIST and thus also update their priorities in OPEN and FOCAL. An example of this process is illustrated in Figure 3. Suppose that n'' is already in $L(S')$ with timeinterval $[lb'', ub'']$ and a collisions (shown in Figure 3(a)), and we now generate n' with timeinterval $[lb', ub']$ and $b < a$ collisions (shown in Figure 3(b)). Suppose that, $lb'' \in [lb', ub']$ and $ub' \in [lb'', ub'']$ ((shown in Figure 3(c))). After $Merge$, GENERATEDLIST has nodes n_1 and n_2 with state s' and timeintervals $[lb', ub']$ and $[ub', ub'']$, respectively. The number of collisions for n_1 and n_2 is b and a, respectively (shown in Figure 3(d)). Finally, n_1 is a new node inserted into OPEN and conditionally into FOCAL, while n_2 is the updated n'' with a larger g-value of ub'.

We now show that ECBS-CT using SCIPP for the low-level search is w-suboptimal.

Lemma 1. *Whenever a node n is exapnded by focal search, its g-value is bounded by $w(g^*(n)+\frac{w-1}{w}h(n))$, where $g^*(n)$ is the optimal g-value of n.*

Proof. Denote by:

1. n, the node being expanded (that is, n is the head of FOCAL).
2. n_{start}, the start node.
3. n_{min}, the head of OPEN (that is, the node with minimal f-value in the open list).
4. n', the first (/shallowest) node on an optimal plan from n_{start} to n which is in OPEN and has optimal g-value (that is, $g(n') = g^*(n')$). Such n' always exist (can be shown inductively).

From FOCAL's definition,

$$f(n) \leq wf(n_{min}) \tag{1}$$

By definition of n_{min},

$$f(n_{min}) \leq f(n') \tag{2}$$

Since $f(n') = g^*(n') + h(n')$ and $f(n) = g(n) + h(n)$,

$$g(n) + h(n) \leq w(g^*(n') + h(n')) \tag{3}$$

Since h is consistent, f is monotonically non-decreasing along an optimal plan to the goal (that is, $g^*(n') + h(n') \leq g^*(n) + h(n)$). Thus,

$$\begin{aligned} g(n) &\leq w(g^*(n) + h(n)) - h(n) \\ &\leq wg^*(n) + (w-1)h(n) \\ &\leq w(g^*(n) + \frac{w-1}{w}h(n)) \end{aligned} \tag{4}$$

□

Assumption 1. *There exist an $\epsilon > 0$ for which any action (that is, motion or wait) duration is an integer multiple of ϵ.*

Assumption 1 simply states that an action duration is a contiguous period of time (that is, a timeinterval) represented with some finite-precision.

Theorem 2. *When assumption 1 holds, ECBS returns a w-suboptimal plan if one exists.*

Proof. This is the original proof of ECBS. □

Theorem 3. *Let ECBS' be an algorithm similar to ECBS except for the following differences in the low-level search:*

1. *Before expanding the start node $n_0 = (s_0, t = 0)$, recursively generate $(s_0, t + 1)$ as long as being in s_0 at time $t + 1$ does not violate any constraint.*
2. *For any generated node $n = (s, t)$, recursively generate $(s, t + 1)$ with parent n as long as being in s at time $t + 1$ does not violate any constraint.*

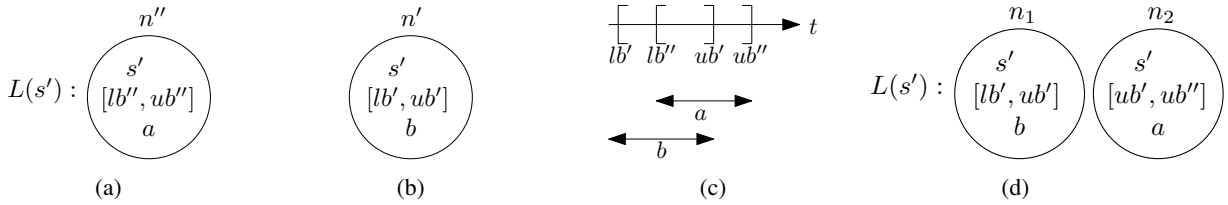


Figure 3: Duplicate detection example.

3. *For any node $n = (s, t)$ selected for expansion, recursively expand $(s, t + 1)$ if the node $(s, t + 1)$ is in OPEN (and have the same h_{FOCAL}).*

ECBS' returns a w-suboptimal plan if one exists.

Proof. The high-level of ECBS relies on the low-level focal search to return:

a) A w-suboptimal plan w.r.t. the constraints.
b) An f_{min} that is a lower bound on the optimal plan cost w.r.t. the constraints.

Focal search achieves (a) and (b) by maintaining the following invariant. For any node n, at least one of the following holds:

i) There is no plan from the start state s_0 to n.
ii) n has already been expanded with $g(n) \leq w(g^*(n) + \frac{w-1}{w}h(n))$.
iii) There exists n' with optimal g-value in OPEN that is on an optimal plan to n.

We now show that ECBS' does not violate the invariant. (1) and (2) do not violate the invariant because (1) and (2) do not remove nodes from open. (3) do not violate the invariant because, if it removes n', a successor of n' must be generated with an optimal g-value, be in OPEN and on an optimal plan to n.

Since the low-level of ECBS' also maintains this invariant, ECBS' returns a w-suboptimal plan if one exists. □

SCIPP is equivalent to the low-level search of ECBS' but, instead of generating separate nodes recursively, it generates them "in bulk" by using timeintervals. Thus, ECBS-CT using SCIPP for the low-level search is w-suboptimal.

Experiments

We present experimental results for two environment maps and two motion primitives.[5] The environments are two benchmark maps from the Grid-Based Path Planning Competition (GPPC). The two maps, Arena and Den520d, as shown in Figures 4(a)&(b), are representative of obstacle-rich environments. The motion primitives are taken from the Search-based Planning Laboratory (SBPL). The two primitives, Unicycle and PR2, as shown in Figures 4(c)&(d), are popularly studied. In the Unicycle (PR2) setup, there are 16 discrete orientations, each with 5 (13) primitives. Like in ECBS, we precompute the perfect single-agent heuristic in the environment. This is used for single-agent planning in SCIPP.

We generated MAMP problem instances with different numbers of agents. The numbers of agents in each setup, with increments of 2 (5) in the Arena (Den520d) map, are indicated in Table 1. For each number of agents, the reported results are averaged over 25 randomly generated instances. Each run is given a time limit of 100 seconds; and the runtime average uses 100 seconds for a timed-out run. We evaluated ECBS-CT with $w = 1, 1.2, 1.5$ and 2. All experiments used an i7-7700 CPU with 16GB RAM.

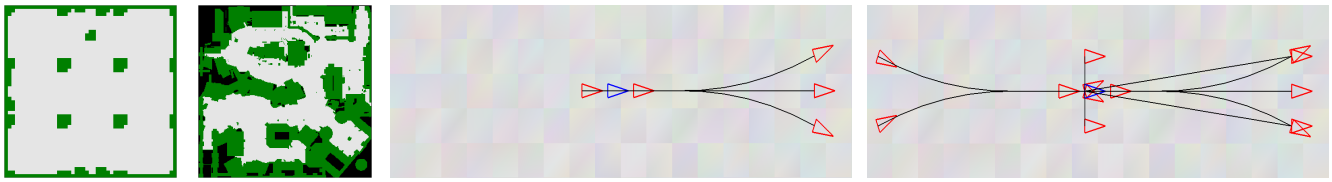
For all combinations of maps and motion primitives, we observe that the w-suboptimal MAMP solvers with $w > 1$ are significantly better than the optimal solver, both in terms of runtime and success rate. This observation shows the power of our bounded-suboptimality framework for MAMP. The w-suboptimal solvers also exhibit the "diminishing returns" property with increasing w. This means that ECBS-CT with $w = 1.2$ or 1.5 is not only effective (since w is small) but also efficient in finding solutions. Moreover, for most instances, ECBS-CT with $w > 1$ produces a solution with an experimentally determined suboptimality[6] that is significantly smaller than the suboptimality bound w. For example, when $w = 2$, the average experimentally determined suboptimality is 1.17 for 50 agents in the Arena map with the PR2 motion primitives.

The Arena map is much smaller than the Den520d map. Thus, single-agent plans in the Arena map tend to be shorter. However, with an increasing number of agents, the number of collisions that ECBS-CT needs to resolve increases more rapidly in the Arena map due to a higher density of agents. The PR2 motion primitives are richer than the Unicycle motion primitives since they include the possibility of turning in place and moving sideways. Therefore, using the PR2 motion primitives result in a larger branching factor but also allow for more flexibility. In the smaller Arena map, the flexibility in the PR2 motion primitives helps ECBS-CT resolve more collisions, thereby increasing the success rate for higher numbers of agents. In the larger Den520d map, the smaller branching factor of the Unicycle motion primitives helps SCIPP find longer plans more quickly, thereby increasing the success rate for higher numbers of agents.

Our results show that ECBS-CT is viable for solving realistic MAMP problems efficiently. Compared to other solvers, ECBS-CT not only solves a richer problem than

[5] taken from https://movingai.com/GPPC/ and http://sbpl.net/, respectively

[6] The experimentally determined suboptimality for each instance is the ratio of the solution cost to $f_{\min}$. This ratio is no larger than w by design.



(a) (b) (c) (d)

Figure 4: (a) shows the Arena map with 49×49 cells. (b) shows the Den520d map with 256×257 cells. (c) and (d) show the Unicycle and the PR2 motion primitives, respectively, for (x, y, θ) in 5×18 free cells for the start state depicted in blue. For a motion primitive, a black line represents the trajectory of the center of the agent's footprint and a red triangle at the end of it represents the orientation at the destination state.

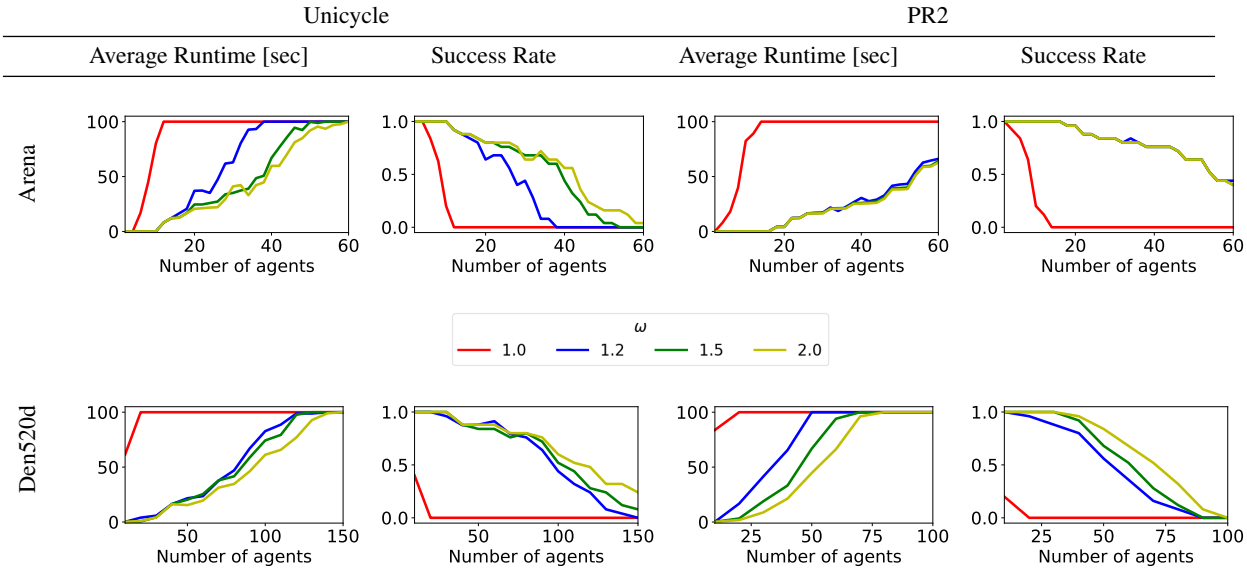


Table 1: shows a matrix of experimental results for the two maps and the two motion primitives from Figure 4. Four values are used for the suboptimality bound w, including $w = 1$ for the optimal MAMP solver.

MAPF$_R$ but also scales to larger numbers of agents in obstacle-rich maps.

Conclusions and Future Work

We introduced the MAMP problem, a generalization of the MAPF problem for kinodynamically constrained agents. We presented ECBS-CT, a generalization of ECBS that efficiently reasons with continuous time. In the high-level search, this requires a proper consideration of completeness, w-suboptimality and Zeno behaviors. In the low-level search, this requires efficient data structures for the reservation table and a generalization of SIPP to reason with collisions and thereby enable the use of focal search. Experimental results demonstrate the promise of our approach. There are many avenues for future work, including designing heuristic values for the high-level search, different heuristic values for the low-level search and motion primitives with velocity considerations, among others.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189 and 1837779 as well as a gift from Amazon.

References

Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the Annual Symposium on Combinatorial Search*, 2014.

Marcello Cirillo, Tansel Uras, and Sven Koenig. A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2014.

Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Bo-

yarski, Meir Goldenberg, Guni Sharon, Nathan R. Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Proceedings of the International Symposium on Combinatorial Search*, 2017.

Dave Ferguson, Thomas Howard, and Maxim Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11-12):939–960, 2008.

Wolfgang Hoenig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic contraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2016.

Wolfgang Hoenig, James Preiss, T. K. Satish Kumar, Gaurav Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34(4):856–869, 2018.

Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), 1996.

James Kuffner and Steven LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the International Conference on Robotics and Automation*, 2000.

Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in SE(3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018.

Hang Ma, Wolfgang Hoenig, T. K. Satish Kumar, Nora Ayanian, and Sven Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Robert Morris, Corina S. Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *Proceedings of the AAAI Workshop on Planning for Hybrid Systems*, 2016.

Judea Pearl and Jin Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:392 –399, 1982.

Mike Phillips and Maxim Likhachev. SIPP: Safe interval path planning for dynamic environments. In *Proceedings of the International Conference on Robotics and Automation*, 2011.

Mikhail Pivtoraiko, Ross Alan Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

Thomas Prevot, Joseph Rios, Parimal Kopardekar, John Robinson III, Marcus Johnson, and Jaewoo Jung. UAS traffic management (UTM) concept of operations to safely enable low altitude flight operations. In *Proceedings of the AIAA Aviation Technology, Integration, and Operations Conference*, 2016.

Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar, George Pappas, and Sanjit Seshia. Implan: Scalable incremental motion planning for multi-robot systems. In *Proceedings of the International Conference on Cyber-Physical Systems*, 2016.

Joao Salvado, Robert Krug, Masoumeh Mansouri, and Fedorico Pecora. Motion planning and goal assignment for robot fleets using trajectory optimization. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2018.

Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470 – 495, 2013.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

Trevor Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the European Conference on Artificial Intelligence*, 2016.

Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

Thayne T. Walker, David Chan, and Nathan R. Sturtevant. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2017.

Thayne Walker, Nathan R. Sturtevant, and Ariel Felner. Extended increasing cost tree search for non-unit cost domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018.

Konstantin Yakovlev and Anton Andreychuk. Any-angle pathfinding for multiple agents based on SIPP algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2017.

Jingjin Yu and Steven LaValle. Planning optimal paths for multiple robots on graphs. In *Proceedings of the International Conference on Robotics and Automation*, 2013.

Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2013.

Jun Zhang, Karl Henrik Johansson, John Lygeros, and Shankar Sastry. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 11(5):435–451, 2001.