Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

Optimal and Bounded-Suboptimal Multi-Agent Motion Planning

**Liron Cohen, Tansel Uras,
T. K. Satish Kumar, 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 conflict-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 Conflict 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 scale well with increasing suboptimality bounds.

Introduction

Multi-Agent Motion Planning (MAMP) is the task of finding conflict-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, Uras, and Koenig 2014) and unmanned aerial traffic management systems (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 with 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, Sturtevant, and Felner 2018) which is identical to the MAPF problem but allows positive non-uniform edge weights to represent different action durations. Still, in MAPF$_R$, vertices represent locations in metric space and movements are 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 time interval during which the agent executing the motion occupies it. As a consequence, in our formulation, an agent is allowed to take any geometric shape. This formulation also naturally lends itself to reasoning over state lattices (Pivtoraiko, Knepper, and Kelly 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 solvers that attempt to solve it are either incomplete or inefficient (Cirillo, Uras, and Koenig 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 Conflict 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* if $s_0^j = s^j$, 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 complete surveys.

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, Sturtevant, and Felner 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, Chan, and Sturtevant 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 if and only if 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 the 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) (Barer et al. 2014) 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 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. 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 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, Knepper, and Kelly 2009) are extensions of grids that are able to model motion constraints and are therefore well suited to planning for non-holonomic and highly constrained agents with limited maneuverability. 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 actions of the agent. A state in a state lattice is a tuple of the form $(x, y, z, \theta, v, \ldots)$, where x, y, 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 were successfully used for autonomous navigation in DARPA's urban challenge (Ferguson, Howard, and Likhachev 2008) and for quadrotors (Liu et al. 2018). A state lattice facilitates heuristic search algorithms that find optimal or bounded suboptimal paths[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 time intervals during which it does not collide with any dynamic obstacles. Using time intervals in the state-space representation allows 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 with an 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 take 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 list 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 time interval $[lb^e_i, ub^e_i]$ during which it is swept by e after the beginning of its execution. Thus $\min_{1 \le i \le m(e)} lb^e_i = 0$ and $\max_{1 \le i \le m(e)} ub^e_i = w(e)$.

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* if and only if $e^j_1 \ldots, e^j_{T_j}$ is an s^j-g^j path 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 time interval $[t^j_{i-1} + w(e^j_{i-1}), t^j_i]$.[2] We assume that agent j occupies $\{c^s_1, \ldots, c^s_{m(s)}\}$ for $s = g^j$ during the time interval $[t^j_{T_j} + w(e^j_{T_j}), \infty]$. 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 if the time intervals in which they sweep or occupy the same cell overlap. 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. We note that the MAMP problem can also be equivalently considered on PRMs and RRTs.

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 no more than w times the optimal cost. Thus, for $w = 1$, ECBS-CT is optimal and essentially generalizes CBS for the MAMP problem.

In 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 in lines 3-12 performs a focal search. In 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 of time intervals in which two or more agents collide. This can be efficiently computed while the reservation table is updated in line 10.

In line 6, a conflict $(c, [lb, ub])$ between two agents j and k is identified. We focus on resolving the earliest conflict among all agents in high-level node N; and this earliest conflict can be efficiently identified using the reservation table. Resolving the earliest conflict is known to be beneficial in the CBS framework (Sharon et al. 2015). The conflict is resolved in line 8 by posting a constraint on the 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 natural questions that arise in this context are: (1) "Why is a timepoint used instead of a time interval?" and (2) "What should be the value of τ?"

[1] optimality with respect to the state lattice discretization

[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.
$Push$ the root node to OPEN and FOCAL.
while *FOCAL* $\neq \emptyset$ **do**
  $N \leftarrow Pop$(FOCAL).
  if N *is a solution* **then return** N.
  Identify a conflict $(c, [lb, ub])$ between agents j and k at cell $c \in \mathcal{C}$ during the time interval $[lb, ub]$.
  Identify a time point $\tau \in [lb, ub]$.
  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 reservation tables in N^j and N^k.
  $Push$ N^j and N^k to OPEN and conditionally to FOCAL.
  $Update$ FOCAL if necessary.
return no solution.

Algorithm 2: SCIPP

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

root nodes $\leftarrow InitializeNodes(s^j)$.
$Push$ root nodes to OPEN and conditionally to FOCAL.
while *FOCAL* $\neq \emptyset$ **do**
  $n \leftarrow Pop$(FOCAL).
  if n *is a goal node* **then return** plan.
  for *each* $n' \in GenerateSuccessorNodes(n)$ **do**
    $\mathcal{N} \leftarrow Merge\ n'$ into GENERATEDLIST.
    $Push/Update$ OPEN and FOCAL according to $\mathcal{N}$.
  $Update$ FOCAL if necessary.
return no solution.

The answer to the first question relates to the requirements of CBS (ECBS) to guarantee optimality (w-suboptimality). 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 time interval $[lb', ub'] \subseteq [lb, ub]$ instead of a timepoint, this property no longer holds. In particular, if an optimal solution includes agent j sweeping c during the time interval $[lb', (lb' + ub')/2)$ and agent k sweeping c during the time interval $((lb' + ub')/2, ub']$, it is spuriously eliminated.

The answer to the second question relates to Zeno behaviors (Zhang et al. 2001). If $\tau < ub$, agents j and k can satisfy their respective constraints by waiting for $\tau - lb$ time units before conflicting again at c during the non-singleton time interval $(\tau, ub]$. Similarly, in a future iteration, they may conflict yet again during the non-singleton time interval $(\tau', ub]$ for some $ub > \tau' > \tau$. Therefore, any strategy for choosing the value of τ other than setting it to the upper bound of the time interval (ub) results in a 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 time intervals. A time interval $[lb, ub]$ in the reservation table for a cell c has an associated value v that indicates the number of agents occupying c during $[lb, ub]$. In addition, c's time intervals are not necessarily disjoint. Thus, a suitable data structure is required to efficiently maintain the reservation table. For this purpose, we use the *interval map* (Cormen et al. 2009). It supports efficient insertion, deletion, search and aggregate-on-overlap operations. The aggregate-on-overlap operation combines (separates) the associated values of intersecting time intervals on insertion (deletion) with the same complexity as the insertion (deletion) operation. Figure 1(c) illustrates the aggregate-on-overlap operation. 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 usage of interval maps for the reservation table facilitates an efficient detection of the earliest conflict in the high-level search and an efficient computation of secondary heuristic values for focal search in the high and low-level searches.

SCIPP

Since focal search with discrete timesteps is vital for 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 hard constraints specified by the high-level node, it unfortunately cannot be used to reason about the soft conflicts specified in the reservation table of the high-level node. Therefore, SIPP is not suitable for focal search. Instead, we develop SCIPP to serve the purpose of the low-level focal search of ECBS-CT. SCIPP not only efficiently reasons about continuous time and hard constraints but also successfully reasons about soft conflicts and derives secondary heuristic values from them.

Given a start state s^j, a goal state g^j and a list of safe time intervals for each cell, SIPP finds a plan from s^j to g^j with minimum arrival time at g^j. This plan also guarantees that no cell is swept outside its safe time intervals. In SIPP, a node n represents an $(s, [lb, ub])$ pair where s is a state and $[lb, ub]$ is a safe time interval of s. The successor n' of n represents a valid 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 be $g(n) + d$ if $g(n') > g(n) + d$, where d is the duration of the motion.

Algorithm 2 presents SCIPP, a generalized version of SIPP suitable for focal search. It takes as input a start state

[4] A motion is legal if it sweeps each of its associated cells only during its safe time intervals.

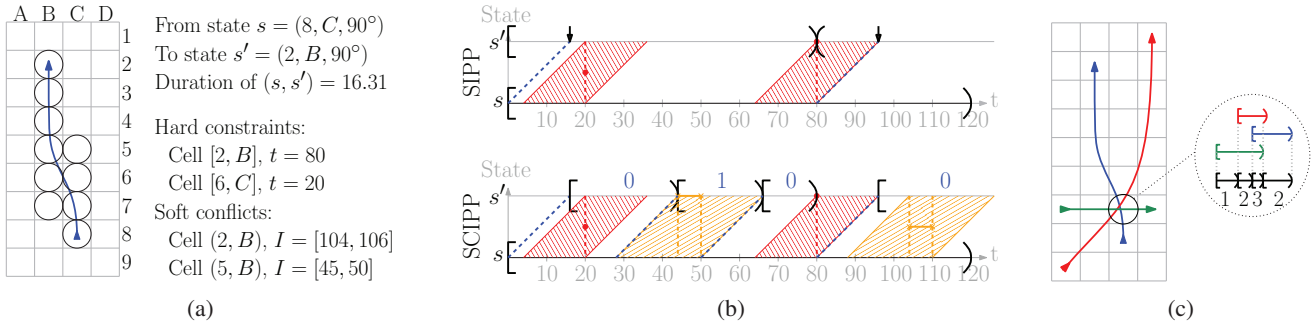


Figure 1: (a) shows an environment with 36 cells (grey squares) and a motion primitive (s, s') in blue from s to s'. At s and s', the agent occupies cells $[8, C]$ and $[2, B]$, respectively. Black circles depict swept cells throughout the motion. (b) illustrates the difference between SIPP and SCIPP. The x-axis represents time and the y-axis represents different states. A red dot between s and s' (on the horizontal line for s') represents a hard constraint at the time of its x-coordinate at an intermediate swept cell (destination cell) of (s, s'). In both SIPP and SCIPP, the safe time interval for s is $[0, 120)$. In SIPP, the safe time intervals for s' are $[0, 80)$ and $(80, \infty)$. The black markers indicate the earliest arrival times within these time intervals, serving as g-values. The red regions indicate blocked execution times; and the slope of the parallelograms indicate the duration of (s, s'). In SCIPP, a yellow horizontal line between s and s' (on the horizontal line for s') represents a soft conflict during the time interval between its end-point x-coordinates at an intermediate swept cell (destination cell) of (s, s'). The time intervals of s' are subdivided such that each has a constant number of soft conflicts, depicted with blue labels. (c) shows three agents sweeping the incircled cell during overlapping time intervals. The aggregate-on-overlap operation produces 4 time intervals, each with an associated number of soft conflicts, as indicated in black.

s^j, a goal state g^j, a list of hard constraints for each cell c and a reservation table, as specified in the high-level node. It outputs a w-suboptimal plan from s^j to g^j for the specified value of w without violating any hard constraints. 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 time intervals of all nodes in $L(s)$ are maintained to be disjoint. For each time interval $[lb, ub]$ of a node $n \in L(s)$, lb represents the earliest possible arrival time to n via n's predecessor without violating any hard constraint. Thus, $g(n) = lb$. $ub - lb$ represents the maximum wait duration at n while retaining the same number of soft conflicts and without violating any hard constraints. The secondary heuristic value of n equals its number of soft conflicts, 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.

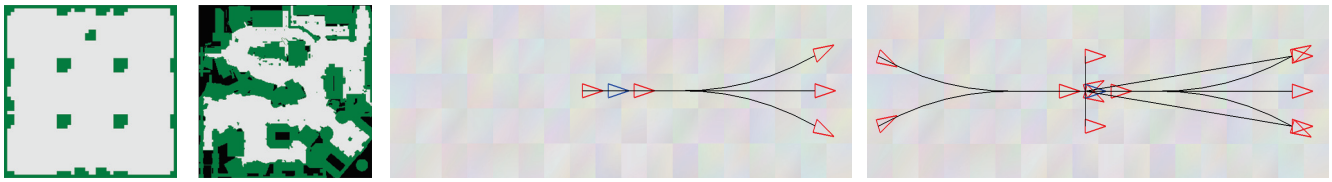
The main loop in lines 3-9 performs a focal search after initialization in lines 1-2. $InitializeNodes(s^j)$ in line 1 generates a list of nodes corresponding to disjoint time intervals between 0 and the earliest hard constraint imposed on any cell associated with s^j. Each time interval in this list has a constant number of soft conflicts. $GenerateSuccessorNodes(n)$ in line 6 generates all successors of n. Like in SIPP, n' represents a valid transition from s to s' via a legal motion. In SCIPP, the time interval of n' may have to be split up to disjoint time intervals so that the number of soft conflicts associated with each remains constant. Thus, more nodes may have to be created, one for each disjoint time interval. Figure 1(b) illustrates this process. Here, split time intervals are created by first generating "arrival" time intervals with respect to the hard constraints and soft conflicts on the swept cells of the motion, and then extending these arrival time intervals with respect to the hard constraints and soft conflicts on the cells associated with s' in order to reason about possible waiting at s'.

The mechanism of duplicate detection in focal search is implemented using $Merge$ in 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 time intervals overlap. Regardless of the different ways in which the two time intervals may overlap, the $Merge$ process restores the invariant that all nodes in $L(s')$ are disjoint and each has a constant number of soft conflicts. The $Merge$ process can either: (1) add the newly created nodes from $GenerateSuccessorNodes$ to GENERATEDLIST and push them to OPEN and conditionally to FOCAL; or (2) update the time interval of the existing nodes in GENERATEDLIST and thus also update their priorities in OPEN and FOCAL. For example, suppose n'' has a soft conflicts, n' has $b < a$ soft conflicts, $lb'' \in [lb', ub']$ and $ub' \in [lb'', ub'']$. After $Merge$, GENERATEDLIST has nodes n_1 and n_2 with state s' and time intervals $[lb', ub']$ and $[ub', ub'']$, respectively. The number of soft conflicts for n_1 and n_2 is b and a, respectively. Finally, n_1 is a new node pushed to OPEN and conditionally to FOCAL, while n_2 is the updated n'' with a larger g-value of ub'.

Experiments

We present experimental results for 2 different environment maps and 2 different motion primitives.[5] The environments are 2 benchmark maps from the Grid-Based Path Planning

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



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

Figure 2: (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 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.

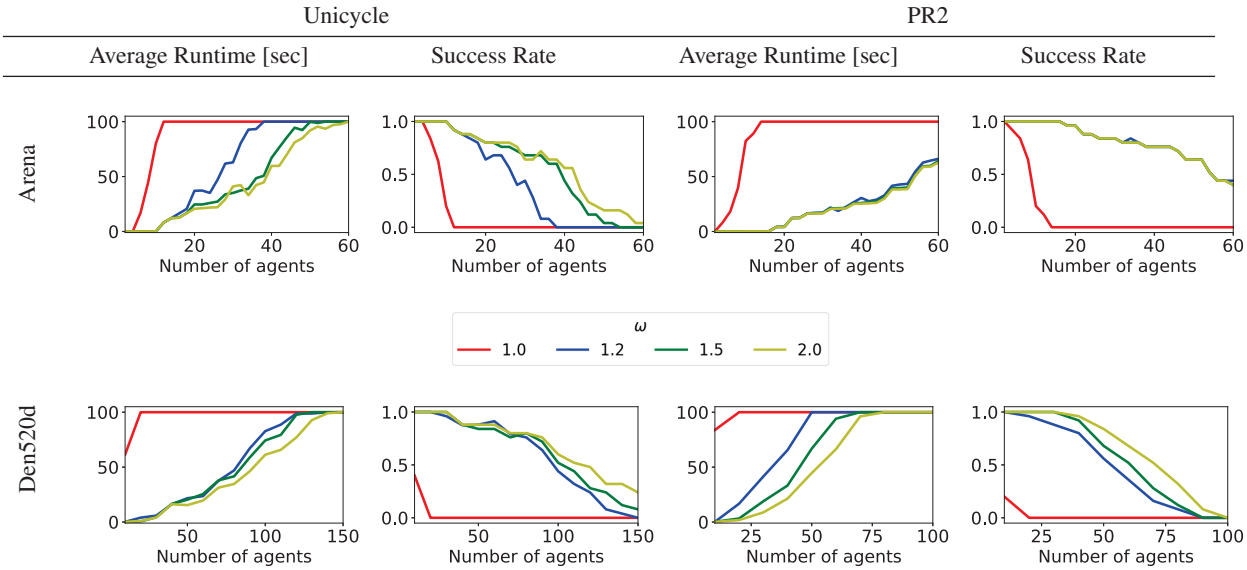


Table 1: shows a matrix of experimental results for the 2 different maps and the 2 different motion primitives from Figure 2. 4 different values of the suboptimality bound, w, are used, including $w = 1$ for the optimal MAMP solver.

Competition (GPPC). The 2 maps, Arena and Den520d, as shown in Figures 2(a)&(b), are representative of obstacle-rich environments. The motion primitives are taken from the Search-based Planning Laboratory (SBPL). The 2 primitives, Unicycle and PR2, as shown in Figures 2(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 varying numbers of agents in each setup, with increments of 2 (5) in the Arena (Den520d) map, are indicated in Table 1. For each value of the 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 $w = 1, 1.2, 1.5, 2$. All experiments used a core i7-7700 CPU with 16GB RAM.

For all combinations of maps and motion primitives, we observe that the w-suboptimal 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 for the reason that w is small but also efficient in finding solutions. Moreover, for most instances, ECBS-CT with $w > 1$ produces a solution with a suboptimality guarantee that is significantly smaller than the suboptimality bound w.[6] For example, when $w = 2$, the average suboptimality guarantee 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 increasing numbers of agents, the number of conflicts that ECBS-CT needs to resolve increases more

[6]The suboptimality guarantee for each instance is the ratio of the solution cost to $f_{\min}$. This ratio is $\leq w$ by design.

rapidly in the Arena map due to a higher density of agents. The PR2 motion primitives are richer than the Unicycle motion primitives. Specifically, they include the possibility of turning in place and moving sideways. Therefore, the PR2 motion primitives have 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 conflicts, 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 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 soft conflicts and thereby enable the use of focal search. Experimental results demonstrate the promise of our approach. There are many avenues for future work, including heuristic guidance for high-level search, different heuristic designs for the low-level search and evaluating 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

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 7th Annual Symposium on Combinatorial Search*.

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

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2009. *Introduction to Algorithms, Third Edition*. The MIT Press.

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 10th International Symposium on Combinatorial Search*.

Ferguson, D.; Howard, T.; and Likhachev, M. 2008. Motion planning in urban environments. *Journal of Field Robotics* 25(11-12):939–960.

Hoenig, W.; Kumar, T. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic contraints. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*.

Hoenig, W.; Preiss, J.; Kumar, T.; Sukhatme, G.; and Ayanian, N. 2018. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics* 34(4):856–869.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4).

Kuffner, J., and LaValle, S. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Liu, S.; Mohta, K.; Atanasov, N.; and Kumar, V. 2018. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters* 3(3):2439–2446.

Ma, H.; Hoenig, W.; Kumar, T.; Ayanian, N.; and Koenig, S. 2019. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *Planning for Hybrid Systems, AAAI Workshop*.

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

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

Pivtoraiko, M.; Knepper, R. A.; and Kelly, A. 2009. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics* 26(3):308–333.

Prevot, T.; Rios, J.; Kopardekar, P.; Robinson III, J.; Johnson, M.; and Jung, J. 2016. UAS traffic management (UTM) concept of operations to safely enable low altitude flight operations. In *Proceedings of the 16th AIAA Aviation Technology, Integration, and Operations Conference*.

Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G.; and Seshia, S. 2016. Implan: Scalable incremental motion planning for multi-robot systems. In *ACM/IEEE 7th International Conference on Cyber-Physical Systems*.

Salvado, J.; Krug, R.; Mansouri, M.; and Pecora, F. 2018. Motion planning and goal assignment for robot fleets using trajectory optimization. In *IEEE International Conference on Intelligent Robots and Systems*.

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

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015.

Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*.

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 *Proceedings of the 22nd European Conference on Artificial Intelligence*.

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

Walker, T. T.; Chan, D.; and Sturtevant, N. R. 2017. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling*.

Walker, T.; Sturtevant, N. R.; and Felner, A. 2018. Extended increasing cost tree search for non-unit cost domains.

Yakovlev, K., and Andreychuk, A. 2017. Any-angle pathfinding for multiple agents based on SIPP algorithm. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling*.

Yu, J., and LaValle, S. 2013a. Planning optimal paths for multiple robots on graphs. In *IEEE International Conference on Robotics and Automation*.

Yu, J., and LaValle, S. M. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*.

Zhang, J.; Johansson, K. H.; Lygeros, J.; and Sastry, S. 2001. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* 11(5):435–451.