Improving Multi-Agent Path Planning with Kinodynamic Constraints by Motion Primitives

Nikita Zavarzin[1] and Konstantin Yakovlev[2]

[1] Moscow Institute of Physics and Technology, Moscow, Russia
zavarzin.niu@phystech.edu
[2] Federal Research Center for Computer Science and Control of Russian Academy of Sciencies, Moscow, Russia
yakovlev@isa.ru

Abstract. Multi-agent path finding in realistic environments requires not only collision-free coordination but also adherence to agents' kinodynamic constraints, such as acceleration and turning limitations. In this paper, we focus on planning approach, where Priority-Based Search (PBS) serves as the high-level coordination strategy, and Safe Interval Path Planning with Interval Projection (SIPP-IP) is used as the low-level planner for individual agents. We extend this structure to handle a variety of motion primitives. Our evaluation focuses on how these primitives influence the quality, feasibility, and efficiency of generated multi-agent plans across diverse scenarios. The results show that integrating SIPP-IP into PBS supports effective planning under realistic motion constraints. Moreover, the selection of motion primitives significantly impacts the trade-off between computational cost and the quality of generated plans.

Keywords: MAPF · PBS · SIPP-IP · kinodynamic constraints

1 Introduction

The Multi-Agent Path Finding (MAPF) problem is the fundamental problem of planning paths for multiple agents, where the key constraint is that the agents will be able to follow these paths concurrently without colliding with each other. MAPF applications extend to a wide range of robotic applications[1], including automated warehouses[2] and autonomous vehicles[3]. Therefore, it is important to develop approaches to solving MAPF that take into account kinodynamic constraints.

One of the most common algorithms for solving the MAPF problem is Priority-based Search (PBS[4]). In it each agent is assigned a priority, and then only individual paths are searched, while all previously planned trajectories are considered immutable (dynamic obstacles for the next agent). At the same time one of the leading algorithm for trajectory planning for a single agent, taking into account kinodynamic constraints is Safe Interval Path Planning with Interval Projection (SIPP-IP[5]). Based on Safe Interval Path Planning (SIPP[6]) is a variant of A* that can find optimal paths in dynamic environments. SIPP-IP modifies it by moving away from the explicit assumption that an agent can

start/stop moving instantly, taking in turn the assumption (more realistic from the practical perspective) that decelerating to a full stop and, similarly, accelerating from the full stop takes time.

In this paper, we explore a hierarchical planning framework in which PBS is employed for high-level coordination among agents, while SIPP-IP is utilized as the low-level path planner for individual agents. We enhance this architecture by incorporating various motion primitives, enabling the planner to account for more realistic movement constraints.

This paper is structured as follows. Section 2 reviews related work. Section 3 presents the problem formulation. Section 4 describes the methods used in our study. Section 5 is dedicated to the experimental evaluation, and Section 6 summarizes and analyzes the obtained results.

2 Related Work

The most proven approaches to the MAPF problem with kinodynamic constraints can be divided into 4 groups: search-based, sample-based, optimization-based and learning-based.

Search-based methods conduct an systematic exploration of the configuration space after discretization to locate a solution. Theese methods perform well in environments with known obstacle paths in advance, but in cases where the agents needs to avoid obstacles that were not taken into account when designing the road map or drive through a section that is not covered by the graph of the road network, difficulties arise. Leading methods, such as Conflict-Based Search (CBS[7]) and Prioritized-Based Search[4], which was mentioned earlier, use single-agent solvers to plan paths for individual agents and address collisions among agents by introducing constraints to single-agent solvers.

Sampling-based algorithms gradually build a tree based on random samples from the search space, thereby sequentially filling it. At each iteration, an attempt is made to connect the existing tree with a final configurations set. This approach is characterized by high speed and adaptability to a changing environment, but at the same time requires post-processing in the form of smoothing solutions. A prominent representative is a multi-agent RRT*[8] derived from single-agent RRT*[9], combining the state space of individual agents into a collective joint space for RRT* planning. However, planning within the joint state space of agents presents scalability challenges, as the dimension of the joint state space increases exponentially in the number of agents.

Usually, optimization-based approaches initially have a global plan (for example, a potential distribution function in space or polylines connecting the start and end points, etc.) and obstacle trajectories. The point is to build a smooth solution by optimizing some functionality based on the original concept. The main disadvantages are the necessary preprocessing and convergence to local minima, while the benefits are in the continuity of the resulting controls and the quality of solutions. For instance, in [10], a vector field approach combined with model predictive control is used to steer agents in complex environments.

In [11], a mixed-integer linear program is utilized to search for timed waypoints that fulfill signal temporal logic constraints, which encapsulate the kinodynamic constraints.

Learning-based methods are very diverse in terms of neural network architecture, input data, and parameters. Reinforcement learning algorithms are often used, the essence of which is the interaction of agents with the environment and the dependence of rewards on the actions of agents. A separate class is the so-called end-to-end methods, which take observations of the environment as input data, and output trajectories for all agents. The disadvantages of such approaches are model training and difficulty in interpreting solutions, but they are able to quickly adapt to a new environment. One notable example is PRIMAL[12], a decentralized learning-based MAPF method that relies on local observations and combines imitation learning with reinforcement learning to train agents. It enables agents to navigate independently while promoting cooperative behavior through a reward structure that penalizes collisions and interference.

3 Problem Statement

We define our MAPF problem by a graph $G = (V, E)$ and a set of M agents $\mathcal{A} = \{a_1, \ldots, a_M\}$. Every agent has *start* and *goal* vertices: $start, goal \in V$. A vertex of the graph corresponds to the state of the agent, alternatively known as the confguration, in which the agents can reside without colliding with the static obstacles. Also we assume a discretized timeline $T = 0, 1, \ldots$

An edge $e = (v, u) \in E$ represents a *motion primitive*, a small kinodynamically feasible fragment of the agents' motion that transfers any agent from v to (distinct) u. We assume that for any $v \in V$, a finite set of such motions is available and each motion takes an integer number of timesteps. The duration of the motion defines the weight of the edge, $w(e) \in \mathbb{N}$.

Our approach assumes there is another system that tracks dynamic obstacles in the environment, predicts their future trajectories. We use them to search individual trajectories by safe intervals — contiguous periods of time for a configuration or edge, during which there is no collision and it is in collision one time step prior and one time step after the period.

A path for agent k is a sequence $\pi_k = (e_{k,1}, t_{k,1}), (e_{k,2}, t_{k,2}), \ldots, (e_{k,L}, t_{k,L})$, where $t_{k,i} \in T$ is the time moment the motion defined by the edge $e_{k,i}$ is started. The cost of the trajectory is the timestep when the agent reaches the final vertex, $cost(\pi_k) = t_{k,L} + w(e_{k,L})$. A trajectory is feasible if the target vertex of $e_{k,i}$ matches the source vertex of $e_{k,i+1}$ and $t_{k,i+1} \geq t_{k,i} + w(e_{k,i})$ (with $t_{k,0} = 0$). The cost of the solution will be defina as total cost of all agent trajectories.

A collision between the trajectory π_k and the dynamic obstacles occurs *iff* either of the conditions occur: i) the agent starts executing some motion primitive e at the timestep which is outside the safe interval corresponding to this edge; ii) according to π_k, the agent waits at a vertex v outside of safe interval corresponding to this vertex.

The problem now is to find a set of feasible collision-free trajectories $\pi =$

$(\pi_1, \ldots, \pi_M)$ that transfers the agents from their *start* to their *goal* positions on a given graph G (with the annotated safe intervals of vertices/edges).

4 Methods

Our approach to solving the MAPF problem with kinodynamic constraints employs a hierarchical framework. We utilize Priority-Based Search (PBS) for high-level multi-agent coordination and Safe Interval Path Planning with Interval Projection (SIPP-IP) as the underlying low-level planner for individual agents. This structure allows us to effectively incorporate and evaluate the impact of various motion primitives.

4.1 Safe Interval Path Planning with Interval Projection (SIPP-IP)

SIPP-IP [5] is specifically designed to address the limitations of the original SIPP algorithm [6] when agents face realistic kinodynamic constraints. While SIPP assumes agents can start and stop instantly (allowing waits at any configuration), SIPP-IP acknowledges that waiting is often only possible at specific states (e.g., zero velocity) and that transitions between velocity states take time and space.

Algorithm 1: SIPP-IP

```
OPEN ← ∅, CLOSED ← ∅
ti ← [t_start, t_start]
if v_start.vel = 0
    ti.t_u ← upper bound of SI(v_start, ti)
endif
n_start ← (v_start, ti)
f(n_start) ← t_start + h(v_start)
Add n_start to OPEN
while OPEN ≠ ∅ do
    n ← state from OPEN with minimal f-value
    Remove n from OPEN, Insert n to CLOSED
    if n.v = v_goal
        return π ← ReconstructPath(n, n_start)
    endif
    SUCC ← getSuccessors(n, G(V, E), SI)
    for n' in SUCC
        if n' in CLOSED or in OPEN
            continue
        endif
        f(n') ← n'.t_l + h(n'.v)
        Add n' to OPEN
    endfor
end
return ∅
```

Algorithm: getSuccessors

```
1 SUCC ← ∅
2 for e = (n.v, v') in available motions
3     intrvls = projectIntervals(n, e, SI)
4     if v'.vel = 0
5         for ti in intrvls
6             ti.t_u ← upper bound of SI(v', ti)
7     for ti in intrvls
8         Insert (v', ti) to SUCC
9 return SUCC
```

SIPP-IP performs an A*-based search over a state-space where each node $n = (v, [t_l, t_u])$ consists of a configuration v (position, orientation, and velocity) and a *waiting interval* $[t_l, t_u]$ denoting the admissible departure times from the predecessor node. This interval must be fully contained within a safe interval for v and implicitly captures possible waiting at the previous configuration.

The core feature of SIPP-IP is its *interval projection* method for successor generation. When expanding a node:

1. The interval $[t_l, t_u]$ is projected forward along each motion primitive $e = (v, v')$ with duration $w(e)$.
2. Arrival intervals at v' are computed such that traversals from $t \in [t_l, t_u]$ to $t + w(e)$ are collision-free.
3. Valid arrival times are grouped into maximal contiguous intervals $[t'_l, t'_u]$ within safe intervals of v'.
4. If waiting is allowed at v', t'_u is extended to the end of the containing safe interval.
5. Each valid interval produces a successor node $(v', [t'_l, t'_u])$, which is added to the OPEN list if not already visited with a better cost.

4.2 Priority-Based Search (PBS)

We employ Priority-Based Search (PBS) [4] as a high-level search coordinating multiple agents by resolving conflicts through relative priorities. PBS explores a Priority Tree, where each node N contains a set of agent paths ($N.plan$), a cost ($N.cost$), and a partial order $\prec_N$ representing relative agent priorities ($k \prec_N j$ means a_k has priority over a_j). The complete PBS process, including the path update procedure, is detailed in Algorithm 3.

The search begins at a root node, potentially with initial priorities $\prec_0$. An initial plan is computed by calling the `UpdatePlan` procedure (defined within Algorithm 3, lines 6-18) for all agents. The root is added to a priority queue `OPEN`, ordered by cost. The main loop (lines 25-42) extracts the lowest-cost node N from `OPEN`. If $N.plan$ is collision-free, it is returned as the solution.

Otherwise, the first collision C is identified. To resolve it, the search branches:

for each agent a_k involved in C, a new child node N' is created, adding constraints $a_l \prec a_k$ for all other agents a_l in C and updating the partial order $\prec_{N'}$. The `UpdatePlan` procedure is then called for agent a_k within node N' to replan its path respecting $\prec_{N'}$. If successful, N' is added to `OPEN`. If `OPEN` becomes empty, no solution is found.

The `UpdatePlan` procedure (lines 6-18) updates paths within a node N according to its partial order $\prec_N$. It identifies agents j needing replanning (those involved in the trigger or conflicting with higher-priority agents k where $k \prec_N j$). For each such agent j, it invokes the low-level planner, SIPP-IP, providing the paths of all agents a_k with $k \prec_N j$ as dynamic obstacles (line 11). If SIPP-IP finds a path for a_j, $N.plan$ is updated (line 15); otherwise, the procedure returns $false$, pruning this branch of the Priority Tree (line 13). If all updates succeed, it returns $true$ (line 17). This framework uses SIPP-IP to ensure kinodynamic feasibility while PBS manages inter-agent conflicts via priority assignments.

Algorithm 3: Priority-Based Search (PBS)

Input : MAPF instance ($G(V,E)$, agents $\mathcal{A} = \{a_1..a_M\}$, start/goals, SI), initial partial order $\prec_0$ ($= \emptyset$ by default)
Output : Collision-free plan $N.plan$ or "No Solution"

Procedure UpdatePlan(N, a_i):
  LIST $\leftarrow$ topological sorting on $(\{a_j \mid a_i \prec_N a_j\} \cup \{a_i\}, \prec_N)$;
  for *each* $j \in$ LIST
    if $j = a_i$ *or* ($\exists a_k : k \prec_N j$ *and* a_j *collides with* a_k *in* $N.plan$)
      $dynamic_obstacles \leftarrow$ {path of $a_k \mid k \prec_N j$ in $N.plan$};
      $new_path \leftarrow$ SIPP-IP($a_j, G,$ SI $\cup$ constraints from $dynamic_obstacles$);
      if $new_path = \emptyset$
        return $false$;
      $N.plan[j] \leftarrow new_path$;
  return $true$;

$\prec_{Root} \leftarrow \prec_0$;
$Root.plan \leftarrow \emptyset$;
for $i \in [M]$
  $success \leftarrow$ UpdatePlan($Root, a_i$);
  if $\neg success$
    return "No Solution";
$Root.cost \leftarrow$ sum of arrival times in $Root.plan$;
OPEN $\leftarrow \{Root\}$;
while OPEN $\neq \emptyset$ **do**
  $N \leftarrow$ node from OPEN with minimum $N.cost$;
  OPEN $\leftarrow$ OPEN $\setminus \{N\}$;
  if $N.plan$ *has no collision*
    return $N.plan$;
  $C \leftarrow$ first collision $\langle a_i, a_j, \dots \rangle$ in $N.plan$;
  for *each* a_k *involved in* C
    $N' \leftarrow$ new node;
    $N'.plan \leftarrow N.plan$;
    N'.constraints $\leftarrow N$.constraints $\cup \{\langle a_l, a_k \rangle \mid a_l \in C, l \neq k\}$;
    $\prec_{N'} \leftarrow \prec_N \cup \{(a_l, a_k) \mid a_l \in C, l \neq k\}$;
    $success \leftarrow$ UpdatePlan(N', a_k);
    if $success$
      $N'.cost \leftarrow$ sum of arrival times in $N'.plan$;
      Insert N' into OPEN;
return "No Solution";

References

1. D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 3451–3458, 2022.
2. W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 1125–1131, 2019.
3. J. Li, T. A. Hoang, E. Lin, H. L. Vu, and S. Koenig, "Intersection coordination with priority-based search for autonomous vehicles," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, 2023, pp. 11578–11585.
4. H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in Proceedings of the AAAI conference on artificial intelligence, vol. 33, 2019, pp. 7643–7650.
5. Z. A. Ali and K. Yakovlev, "Safe interval path planning with kinodynamic constraints," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, 2023, pp. 12330–12337.
6. M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in Proceedings of the IEEE International Conference on Robotics and Automation, 2011, pp. 5628–5635.
7. G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," Artificial Intelligence, vol. 219, pp. 40–66, 2015.
8. M. Čáp, P. Novák, J. Vokrínek, and M. Pěchouček, "Multi-agent RRT: sampling-based cooperative pathfinding," in Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems, 2013, pp. 1263–1264.
9. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The international journal of robotics research, vol. 30, no. 7, pp. 846–894, 2011.
10. R. Hegde and D. Panagou, "Multi-agent motion planning and coordination in polygonal environments using vector fields and model predictive control," in Proceedings of the IEEE European Control Conference, 2016, pp. 1856–1861.
11. D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 3451–3458, 2022.
12. G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "PRIMAL: Pathfinding via reinforcement and imitation multiagent learning", IEEE Robotics and Automation Letters, vol. 4, no. 3, pp. 2378–2385, Jul. 2019