Multi-agent Path Planning with Non-constant Velocity Motion

Extended Abstract

Ngai Meng Kou, Cheng Peng, Xiaowei Yan, Zhiyuan Yang, Heng Liu, Kai Zhou, Haibing Zhao, Lijun Zhu, Yinghui Xu
Cainiao Smart Logistics Network
ngaimeng.knm,junpeng.pc,xiaowei.yanxw,shiyuan.yzy,hengsu.lh,jizhou.zk,haibing.zhb,yuanxiang.zlj,renji.xyh
@alibaba-inc.com

ABSTRACT

Multi-agent path planning has wide application in fields such as robotics, transportation, logistics, computer games, etc.. To formulate the multi-agent path finding as a concisely discretized problem, most of the previous works did not construct a detailed motion model of each agent. While many elegant algorithms were proposed in the literature, a method to efficiently plan the paths for multi agents with non-constant velocity is still lacking. In this paper, we propose two methods CRISE and COB to extend the existing algorithms for non-constant velocity motion path planning.

ACM Reference Format:
Ngai Meng Kou, Cheng Peng, Xiaowei Yan, Zhiyuan Yang, Heng Liu, Kai Zhou, Haibing Zhao, Lijun Zhu, Yinghui Xu. 2019. Multi-agent Path Planning with Non-constant Velocity Motion. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

To formulate the multi-agent path planning as a concise problem, traditional methods neglect the motion model of the agents and generate discretized path plans [2, 5], and is generally referred as *multi-agent path finding* (MAPF) problem. Therefore post-processing is necessary for these methods to avoid collisions between the result paths. Hönig et al. proposed an approach to convert imperfect discretized path plans into feasible plan-execution schedules [3]. However, it does not consider kinematic constraints in the process directly, which affects the planned path quality. In addition, it assumes an unrealistic dynamics of the agent with infinite acceleration (uniform velocity model), which is not suitable for environments where the grid length is small or comparable to the acceleration distance of the agent. Some other work formulates the problem as kinodynamic path planning and computes the path directly according to the agent actions [1, 9]. While these methods can return optimal paths, the extremely high computation cost limits the applicability in industry. Recently, [4] proposed modified path planning algorithms based on state-of-the-art MAPF algorithms with similar motivation. But it considered a different problem with discrete waiting time. Our work adopted different mechanisms in state expansion during the path planning search process, both for single and multi agent scenarios. Equipped with the new mechanisms, we can modify many previous MAPF algorithms to take into account agents' motion models.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada. © 2019 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

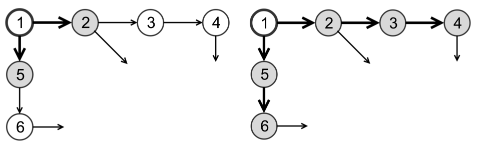
2 PRELIMINARY AND METHODOLOGY

We formulate *Non-constant Velocity Network* (NVN) to consider the physical movement into the graph structure. Let $\mathcal{G}(\mathcal{E}, \mathcal{U}, w, \theta)$ be an NVN with a set of edges $\mathcal{E} = \{e_1, \cdots, e_m\}$ and a set of vertices $\mathcal{U} = \{u_1, \cdots, u_n\}$. The weight of an edge e is $w(e)$ and the direction of e is a unit vector $\theta(e)$. Given a set of agents $\mathcal{A} = \{a_1, \cdots, a_k\}$, MAPF is to find a path $p_i = (s_i, \cdots, g_i)$ for each agent a_i from its start vertex s_i to its target vertex g_i without collision with other agents and minimize the *cumulative time cost* of the paths[1]. An agent moves on its path with a motion model and three types of actions:

(1) **Move:** For a straight path $p = (u_i, \cdots, u_j)$, $move(p)$ let an at rest agent move from u_i and stop at u_j with minimum time.
(2) **Turn:** An action $turn(\theta_i, \theta_j)$ let an agent do a spin turning on current vertex to change its heading from θ_i to θ_j.
(3) **Rest:** $\forall \tau_i \in \mathbb{R}^+$, an action $rest(\tau_i, \tau_j)$ let an agent stay at rest on current vertex for the time period $[\tau_i, \tau_j)$.

We introduce *cri*tical *s*tate *e*xpansion (CRISE) and *Co*ntinuous *b*ranching (COB) for single agent shortest path planning with static and dynamic obstacles, which can be extended to support MAPF.

2.1 Critical State Expansion



(a) Neighborhood expansion (b) Line expansion

Figure 1: Difference between NE and LE. *The bold circle* u_1 *is the current vertex. The expansion go through the thick edges and pushes gray vertices into OPEN. The light edges are ignored.*

To finding the shortest path in NVN, the state of an agent can be represented by a three dimensional tuple $\langle u, \theta, v \rangle$ where u, θ and v are its current position, direction and velocity, respectively. This is an observation that the states with zero velocity are *critical* in the search. So we propose *line expansion* (LE) which can reduce the state space from three dimensions $\langle u, \theta, v \rangle$ to two dimensions $\langle u, \theta \rangle$ by only considering the critical states.

Definition 2.1 (Critical state). A state is critical if $v = 0$.

[1]A collision occurs if two agents meet in the same vertex or edge.

To pass a path p as soon as possible, the agent should decelerate as late as it should stop (as acceleration takes time). It is unnecessary to store all the possible state passing a vertex but the state stops on it. The states of intermediate vertices can be easily inferred accordingly. Only considering the zero-velocity states for every vertices that the agent stops on is sufficient to reform the shortest path.

CRISE is based on this intuition and generates the critical states by LE. The procedure of CRISE is similar to A*. In each iteration, CRISE pops a state $\langle u_i, \theta_i \rangle$ from OPEN with minimum cost τ_i for expansion until g_i is popped or OPEN is empty. Instead of the neighbors, LE considers every reachable vertex u_j of current vertex u_i by a turning followed by a straight line movement $(u_i, \cdots, u_j)$ (see Figure 1). For each vertex u_j, a successor state $\langle u_j, \theta_j \rangle$ with cost $\tau(u_j)$ is generated where θ_j is the arrival direction and $\tau(u_j) = \tau(u_i) + turn(\theta_i, \theta_j) + move((u_i, \cdots, u_j))$ is the arrival time. These states are inserted into OPEN. Then CRISE starts next iteration.

2.2 Continues Branching

We then consider finding the path in continuous time space with dynamic obstacles and arbitrary waiting time. To avoid collisions, the occupied intervals of vertices are stored in a *reservation table.*

Definition 2.2 (Reservation table (RT)). A RT is a set to maintain the reserved interval of each vertex. $\text{RT}(u) = \{[\tau_1, \tau_2), \cdots [\tau_i, \tau_{i+1})\}$ is the set of the reserved time intervals of vertex v.

Given an RT, the challenging is how to do branching in a continuous time-space and how long an agent should wait on its vertex. COB considers reachable vertices $\mathcal{U}'$ by LE of any state $\langle u, \theta \rangle$ in OPEN. For every $u' \in \mathcal{U}'$, it computes the feasible start time periods from u to u' such that for any vertex u_i in the straight line movement $p = \{u, \cdots, u'\}$, a does not occupy any time in $\text{RT}(u_i)$ when it passes u_i. This is done by two steps: *projection* and *branching*. We define four special time points, *Start time*, *Touch time*, *Departure time*, *Arrival time*, as shown in Figure 2, which are thoroughly used.

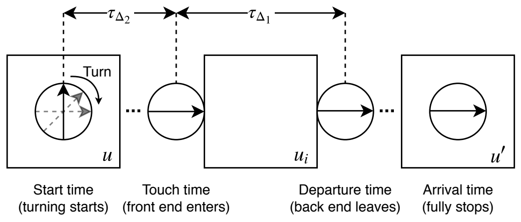

τ_{Δ_2}
τ_{Δ_1}
Turn
u
u_i
u'
Start time (turning starts)
Touch time (front end enters)
Departure time (back end leaves)
Arrival time (fully stops)

Figure 2: A movement from vertex u to another vertex u' in the same line, which contains two actions *turn* and *move*.

Projection: As shown in Figure 3, every reserved time period in $\text{RT}(u_i)$ will cause some start times to become unfeasible. *Projection* is to compute the corresponding unavailable start time period. Suppose $[\tau_l, \tau_r)$ is a reserved time period of u_i. Let the time difference between the touch time and the departure time of u_i be denoted by $\tau_{\Delta 1}$, which can be computed based on the shapes of a and u, and the motion model. Obviously, a cannot touch u_i in $[\tau_l - \tau_{\Delta 1}, \tau_r)$. Projection shifts this time period to the unavailable start time period of u. Let τ_i be the touch time of u_i and τ be its corresponding start time at u. The turning and moving time $\tau_{\Delta 2} = \tau_i - \tau$ from u to u_i can be calculated based on the motion model of a. The unavailable start time period of u is $\text{project}(u, u', u_i, a, [\tau_l, \tau_r)) = [\tau_l - \tau_{\Delta 1} - \tau_{\Delta 2}, \tau_r - \tau_{\Delta 2})$.

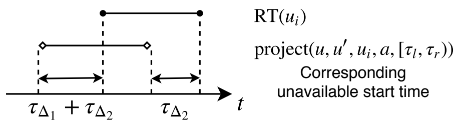

$\text{RT}(u_i)$
$\text{project}(u, u', u_i, a, [\tau_l, \tau_r))$
Corresponding unavailable start time
$\tau_{\Delta_1} + \tau_{\Delta_2}$
τ_{Δ_2}
t

Figure 3: Projection computes the infeasible start time intervals of vertex u by shifting the reserved time intervals of u_i.

Branching: For an agent at u, suppose the earliest and the latest start times are τ_1 and τ_2. The set of available start time periods is

$$\text{wait}(u, u', \tau_1, \tau_2) = [\tau_1, \tau_2) \setminus \left(\bigcup_{u_i \in p, [\tau_l, \tau_r) \in \text{RT}(u_i)} \text{project}\left(u, u', u_i, a, [\tau_l, \tau_r)\right) \right).$$

To search in the time-space, every available start time period should be considered as a state. COB branches the search into each disjoint period $[\tau_{j1}, \tau_{j2})$ of $\text{wait}(u, u', \tau_1, \tau_2)$.

The framework of COB is similar to A* algorithm. In each iteration, a state $\langle u, \theta, [\tau_l, \tau_r) \rangle$ is popped from OPEN list, a priority queue maintaining the minimum cost state. COB retrieves reachable vertices $\mathcal{U}'$ by line expansion. Consider one of the vertex $u' \in \mathcal{U}'$. A set of available start time periods from u to u' can be computed by projection and branching. For each available start time period $[\tau'_l, \tau'_r)$, COB generates a corresponding state $\langle u', \theta', [\tau'_l, \tau'_r) \rangle$ with cost $\tau'_l + h(u')$, where u' is the vertex, θ' is the arrival direction, and $h(u')$ is a consistent heuristic function returning an approximate cost from u' to target vertex. These states are inserted into OPEN list.

Based on CRISE and COB, LRA* [10] and WHCA* [7] can be directly extended to CRISE-LRA* and COB-WHCA* to support motion models and solve MAPF in NVN.

3 EXPERIMENTS

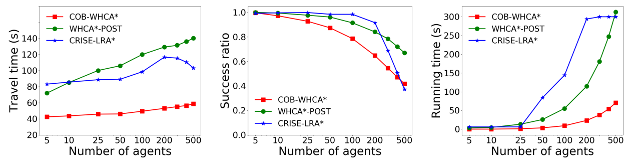


Figure 4: Evaluation results on DAO den900d (128×128)

We followed the setting of previous papers [3, 6, 7]. and evaluated the performance of the proposed methods in a benchmark map of the game *Dragon Age: Origin* (DAO) [8]. We also implemented the post-pocessing MAPF-POST [3] combining with WHCA* (WHCA*-POST in the figures). We report the average travel time of the agents (objective function), success ratio (the percentage of agents reached their targets) and running time (computation cost) of each method.

As shown in Figure 4, although we give an advantage to WHCA*-POST that agents move with maximum velocity, COB-WHCA* and CRISE-LRA* still have shorter travel time because they consider the information of NVN directly in the path planning. COB-WHCA* has lower success ratio because the effect of line expansion on success ratio is similar to a very large window size in WHCA*.

REFERENCES

[1] Marcello Cirillo, Tansel Uras, and Sven Koenig. 2014. A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *2014 IEEE/RSJ IROS 2014, Chicago, IL, USA, September 14-18, 2014*. 232–239. https://doi.org/10.1109/IROS.2014.6942566

[2] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan R. Sturtevant, Glenn Wagner, and Pavel Surynek. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Proceedings of SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA*. 29–37. https://aaai.org/ocs/index.php/SOCS/SOCS17/paper/view/15781

[3] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *ICAPS*. 477–485.

[4] Marius Merschformann, Lin Xie, and Daniel Erdmann. 2017. Path planning for Robotic Mobile Fulfillment Systems. *CoRR* abs/1706.09347 (2017). arXiv:1706.09347 http://arxiv.org/abs/1706.09347

[5] Mike Phillips and Maxim Likhachev. 2011. SIPP: Safe interval path planning for dynamic environments. In *ICRA 2011, Shanghai, China, 9-13 May 2011*. 5628–5635. https://doi.org/10.1109/ICRA.2011.5980306

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

[7] David Silver. 2005. Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Marina Del Rey, 117–122.

[8] N. Sturtevant. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 144 – 148. http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf

[9] Glenn Wagner, Howie Choset, and Avinash Siravuru. 2016. Multirobot sequential composition. In *IROS 2016, Daejeon, South Korea, October 9-14, 2016*. 2081–2088. https://doi.org/10.1109/IROS.2016.7759327

[10] Alexander Zelinsky. 1992. A mobile robot exploration algorithm. *IEEE Trans. Robotics and Automation* 8, 6 (1992), 707–717. https://doi.org/10.1109/70.182671