Multi Agent Path Planning using SIPP

1st Xavier Aramayo Carrasco 2nd Tigran Ramazyan 3rd Ngoc Bich Uyen Vo 4th Alexander Lepinskikh

Abstract—**Single agent path planing in the environment with static obstacles is well known and deeply researched problem. There are a lot of different algorithm and methods which can solve this task with high accuracy and low computational time. However, planning with several agents or navigation in dynamically changes space is stall computationally challenging. In this paper we propose an implementation of advanced $A^{\star}$ algorithm which is using safe intervals - finite and small time periods. Safe interval can be described as a time period in which agents do not collide but if we expand configuration one timestep in either direction, it will be in collision. Planning algorithm exploits observation and construct a search-space with states which a described by their configuration and safe intervals. After that, results of the algorithm on 2 dimensional discrete plane are presented. We have made several experiments with different number of agents and environment configuration. In conclusion, we summarize the results and researching a future improvements of multi agent planning.**

I. Introduction

Planning algorithms are playing an important role in real life's problems. Autonomous driving and moving of house-keepers robots can be examples of performing planning algorithms in real life. In these examples robots should safely navigate throw some space with moving objects, such as people, pets, cars, other robots. To solve this planning problem robots should be able to predict the movements of the dynamic obstacles in the near future. What's more agents need to move by the shortest path to their goal state avoiding collision with moving obstacles. Robots should plan their path really quickly by the reason of if the obstacle trajectory don't coincide with the robot's prediction and the agent need to create new plan in limited time to avoid collision. The efficiency of plan making allows robots to increase their robustness in dynamic environment.

To effectively orientate in dynamic environment additional time dimension should be added in the state space. Adding an additional space noticeably increase the number of possible states, so the planning time grows up. Using the fact that the environment change in time, plans should be created as fast as it is possible or they will be out of date before the robot starts moving by this plan.

One of the possible way to solving this problem is to represent dynamic environment by the static one. It can be done with replacing all dynamic obstacles by the static one (also their trajectory can be predicted near further). But this method is usually sub optimally because if the obstacle and agent trajectories cross each other, robot can loose time waiting for obstacles movement finishing. Furthermore, this approach could not find the solution in the environment with the "doors", where obstacle and robot should go throw single space door.

In this paper we implement a method that using safe intervals to represent agent path planning in dynamically changing environment. Representing any agent state with state intervals we apply to new configuration space adopted $A^{\star}$ algorithm with modified successor function. Next, we perform $A^{\star}$ with safe intervals to two different 2 dimensional grids with different number of agents. With the simplest "door" environment we get two different results: successful solution in the simplest configuration and failure with collision with a more complicated one. On the second grid, we expand environment and number of agent and successfully test them reaching the goal state.

II. Related work

The most commons scenarios in path planning put an agent in middle of an environment with static obstacles surrounding it, the agent use to be positioned in a initial state and the objective is to move it from this location to a goal state, that is usually reachable by applying some actions to the agent. The type of actions allowed also defines the type of planning we will use, discrete or continuous, the key difference between these is the nature of the state space, if the state space is finite or countably finite, then the problem can be solved as discrete planning [1], due tot he nature and complexity of the multi agent problem, we will only focus on the discrete application of this solution.

Several methods have been developed to solve discrete planning problems, some examples are Dijkstra [2], A-star (A*) [3], value iteration [4] and more, all of these strategies focus on solving a single agent path planning, having multiple agents is requires a completely different approach, the type of solutions can be divided into centralised and decentralised solutions. A centralised approach ([5], [6]) has a single global decision maker for all agents, is theoretically optimal but, it does not scale up to many agents due to a prohibitive complexity. A decoupled (decentralized) approach decomposes the problem into several subproblems. The latter approach is faster but yields suboptimal solutions and loses the completeness. One such example is prioritised planning [7], which uses prioritisation to assign an order in which the objects move.

In Silver's work on [8], units perform windowed planning from start to goal in a gridbased world. A backwards A* search is initially run for each agent. The results of these searches (i.e., full open and closed lists) are cached and used as a heuristic guidance in subsequent windowed planning steps, so that each unit knows fully the routes planned by other units, this method is known as HCA* (Hierarchical Cooperative

A*), in the original paper for SIPP [9], there is a comparison between these two works and the main difference relies on efficiency since the search space in SIPP is smaller.

The approach in [10] is similar to SIPP since it recognizes that a state is only needed for the earliest arrival time in a safe interval. The major difference is how SIPP evaluates an edge from one interval to another. In [10], an "expand" is a partial move of a "probe" along an edge via a single timestep which then must be replaced back into the queue.

III. Method

A. Safe intervals

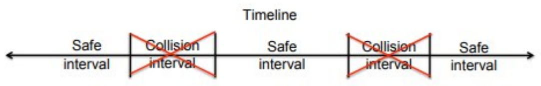

Fig. 1. A timeline describes contiguous safe and collision intervals

Each spatial configuration has a timeline (Figure 1), which is just an ordered sequence of intervals that alternate between safe and collision conditions. A safe interval is defined as a contiguous period of time for a configuration during which there is no collision and the configuration is in collision one timestep previously and one timestep after the period. The opposite of a safe interval is a collision interval. The use of safe intervals to express the temporal dimension of the search space is what makes our algorithm efficient, and it is the central concept of our method.

B. Dynamic obstacle representation

We assume that there is another system that tracks dynamic obstacles in the environment, predicts their future trajectories, and formats them into a general representation we define. We are given a list of dynamic obstacles, where each obstacle has a radius and a trajectory. For simplicity, we will assume each obstacle only has one trajectory. However, the algorithm described in this paper applies to multiple hypothesis trajectories.

C. Graph construction

When we initialize the planner, we generate a timetable for each spatial-configuration based on the projected dynamic obstacle trajectories. This is accomplished by iterating through each point along each dynamic obstacle's trajectory and updating the timelines for all configurations within collision distance of the point. This collision distance is the product of the radius of the obstacle and the radius of the robot.

D. Graph search

After the initialization, we execute the A^* algorithm with safe intervals and the successors function, as detailed in the Algorithm Section.

IV. Algorithm

A. A^* (A-star) algorithm

A^* is a computer algorithm used for path finding and graph traversal. The technique effectively constructs a feasible path between numerous graph nodes, or points. A^* operates by constructing the shortest path tree from the start node to the target node. What distinguishes A^* and makes it superior for many searches is that it utilizes a function $f(s)$ for each node to estimate the overall cost of a path involving that node. As a result, A* is a heuristic function, which varies from an algorithm in that a heuristic is more of a guess and is not always shown to be accurate. A^* expands paths that are already less expensive by using this function:

$$f(s) = g(s) + h(s)$$

where
$f(s) =$ total estimated cost of path through node s
$g(s) =$ cost so far to reach node s
$h(s) =$ estimated cost from s to goal. This is the heuristic part of the cost function, so it is like a guess.
Furthermore, the cost of a transition or edge from s to one of its successors s_0 is defined by $c(s, s_0)$.

Our approach assumes the following key assumptions:

- The heuristic function is consistent, which means that it never overestimates the cost to the objective and that it fulfills the triangle inequality.
- c(s, s0) = time to execute the operation from s to s0. In other words, the planner's purpose is to identify a time minimal trajectory.
- The robot is capable of waiting in place.
- There are no inertial limitations (acceleration/deceleration). The planner expects the robot can instantly stop and accelerate.

B. A^* algorithm with safe intervals and successors function

A^* algorithm with safe intervals and successor function (Figure 3) was proposed, which operates similarly to the standard A^* algorithm except for how it obtains a state's successors (Figure 2) and how it updates the time variable for states (line 11, Figure 3).
Successors function (Figure 2) was generated since we want to construct successors for state s when it is expanded. The function M(s) returns the motions that can be performed from state s. For each of the motions that our robot can perform from s, we compute the resultant configuration and the time of execution (line 3-5, Figure 2). Then, for each time interval in this new arrangement, we produce a successor with the earliest feasible arrival time and no collisions along the motion. The startTime(i) returns the start time of safe interval i and endTime(i), the end time of safe interval i.
When generating successors, we say that s uses "wait and move" actions. This means that for each safe interval in the new configuration, we wait the minimal amount of time possible, so that when the move to the new configuration is

```
getSuccessors($s$)
  $successors = \emptyset$;
  for each $m$ in $M(s)$
    $cfg$ =configuration of $m$ applied to $s$
    $m\_time$ =time to execute $m$
    $start\_t = time(s) + m\_time$
    $end\_t = endTime(interval(s)) + m\_time$
    for each safe interval $i$ in $cfg$
      if $startTime(i) > end\_t$ or $endTime(i) < start\_t$
        continue
      $t$ =earliest arrival time at $cfg$ during interval $i$ with no collisions
      if $t$ does not exist
        continue
      $s'$ = state of configuration $cfg$ with interval $i$ and time $t$
      insert $s'$ into $successors$
   return $successors$;
```

Fig. 2. Successors function

```
1  $g(s_{start}) = 0$; $OPEN = \emptyset$;
2  insert $s_{start}$ into $OPEN$ with $f(s_{start}) = h(s_{start})$;
3  while($s_{goal}$ is not expanded)
4    remove $s$ with the smallest $f$-value from $OPEN$;
5    $successors = getSuccessors(s)$;
6    for each $s'$ in $successors$
7      if $s'$ was not visited before then
8        $f(s') = g(s') = \infty$;
9      if $g(s') > g(s) + c(s, s')$
10       $g(s') = g(s) + c(s, s')$;
11       updateTime($s'$);
12       $f(s') = g(s') + h(s')$;
13       insert $s'$ into $OPEN$ with $f(s')$;
```

Fig. 3. A* algorithm with safe intervals and successors function

used, we arrive in the new safe interval as early as possible. The arrival time is stored with the state, but the state will not be identified by its time value, only by its other dimensions.

During A^*, whenever a shorter path to s_0 is found, the cost is replaced with the smaller one. Similarly, when this happens it corresponds to arriving at s_0 at an earlier time (since cost is equal to time) and so we also replace the time value stored in state s_0 with this new shorter time (line 10-11, Figure 3). We are guaranteed that when s_0 is expanded we have found the earliest time that we can arrive at s_0. This allows us to safely generate and set the time for the successors of s_0.

C. Theoretical investigation

In this section, we outline the proofs supporting our algorithm's completeness and optimality.

Theorem 1: Arriving at a state at the earliest possible time guarantees the maximum set of possible successors.

Theorem 1 is illustrated in Figure 4. Because s exists in a safe interval, the robot can wait from any point in the interval until any point in the interval before proceeding. Even if there is only one state per safe interval, we can still create all the successors that would be generated by a timestep-based algorithm by having that state be the earliest time feasible in the interval and employing "wait and move" operations. This theorem leads to the completeness proof.

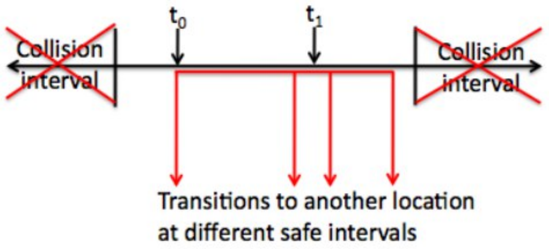

Fig. 4. The set of successors generated from t_0 is a superset of the set of successors generated from t_1

Theorem 2: When the safe interval planner expands a state in the goal configuration, it has found a time-minimal, collision-free path to the goal.
When a state is expanded in optimal A^*, its g-value is minimum. The cost of an edge in this planner is equal to the time required to execute that edge, and if a g-value is updated (due to finding a shorter path), the time value is likewise updated to the earlier time.

Theorem 3: If the configuration with the most dynamic obstacles passing through it has n such occurrences, then each configuration can have at most n + 1 safe intervals.
Because each collision period is followed by a safe interval, and the configuration has n collision intervals, the configuration already contains n safe intervals. If the configuration begins safe, there is one additional safe interval before the first collision interval, for a total of n+ 1 safe intervals.

V. Experiments and results

In order to test the techniques stated above we conducted several experiments exploring cases with variable environments and number of agents, some of the most relevant ones are shown in the figures below.

For experiment 1 (Figure 5) we took the simplest configuration to try the algorithm and it successfully completed the task which was to find an optimal path from the respective initial states of each agent to their goals, no collisions were detected.

In the case of experiment 2, the configuration was a little trickier since we decided to put the goal of each agent over the initial state of the other one, in this scenario, the algorithm failed and one collision was detected (Figure 6), this was mostly do to the timings in which both agents started, it was not possible to predict future collisions and when they reached a point close to collision, there was no chance to avoid it.

For the final experiment we explored the behavior of more than two agents added to a more complicated environment (Figure 7), under these conditions, several runs with collisions appeared sporadically, most of them were due to the same reason as the example above, the timing of execution of actions was not the ideal and it leaded to fail at some point, the algorithm is not able to solve these collision in a short time and it opens a way to improve it.

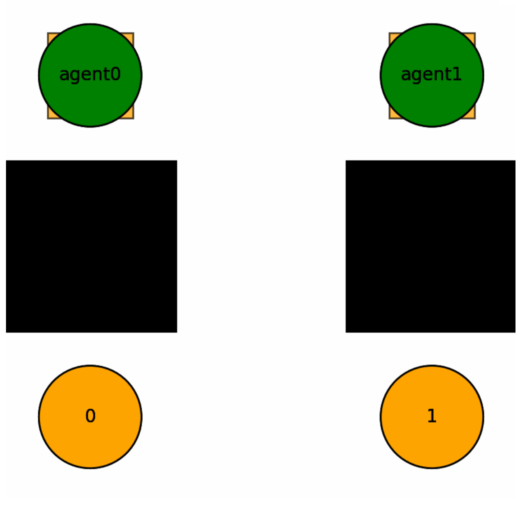

Fig. 5. Experiment 1: Two agents with different goals and initial states.

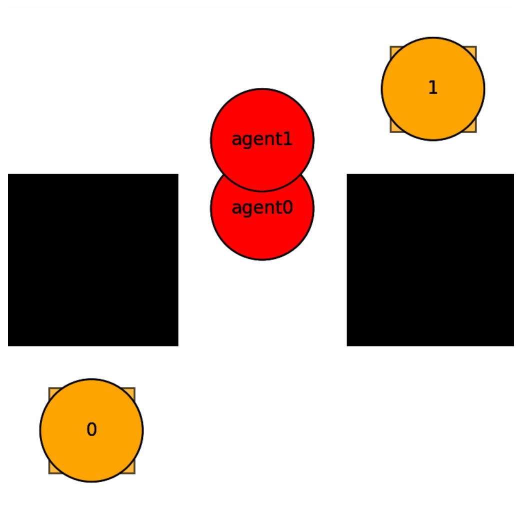

Fig. 6. Experiment 2: Collision detected for a new configuration.

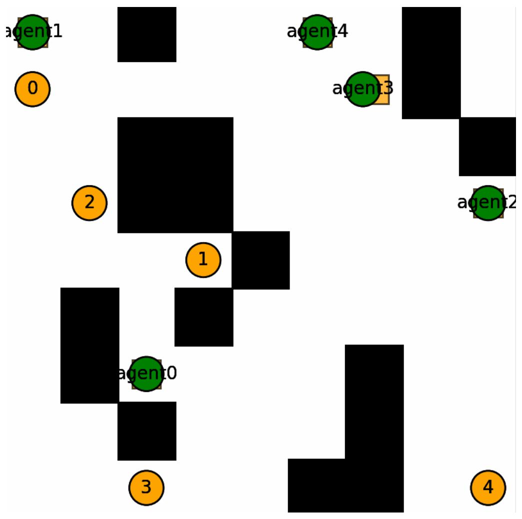

Fig. 7. Experiment 3: Several agents experiment

The results obtained by applying the SIPP algorithm are, overall, promising, it might require some modifications to avoid the beforehand mentioned collisions, but it is still an efficient solution to the multi agent path planning problem.

The code we used for the experiments is available at: SIPP

VI. Conclusions

- SIPP algorithm in an efficient approach to solve the multi agent path planning problem, the paths are commonly found in just seconds and they have a high successful rate.
- The algorithm will find an optimal path in spite of existing collisions, stopping conditions for collisions are not implement for now in the code, since they are useful for testing the situations in which the algorithm doesn't succeed.
- Collision happen since the algorithm is based in timings, these timings are equally distributed but they don't last the same, so, sometimes, timings with collisions will last longer than the collisions free ones, this gives the algorithm a small gap to change the path, but it is not enough in most of the cases.

References

[1] S. M. LaValle, *Planning Algorithms*. USA: Cambridge University Press, 2006.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[3] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/tssc.1968.300136

[4] E. Pashenkova, I. Rish, and R. Dechter, "Value iteration and policy iteration algorithms for markov decision problem," 01 1997.

[5] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991. [Online]. Available: https://doi.org/10.1177/027836499101000604

[6] S. M. LaValle and S. A. Hutchinson, "An objective-based framework for motion planning under sensing and control uncertainties," *The International Journal of Robotics Research*, vol. 17, no. 1, pp. 19–42, 1998. [Online]. Available: https://doi.org/10.1177/027836499801700104

[7] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1–4, p. 477–521, nov 1987. [Online]. Available: https://doi.org/10.1007/BF01840371

[8] D. Silver, "Cooperative pathfinding." 01 2005, pp. 117–122.

[9] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," 06 2011, pp. 5628 – 5635.

[10] J. van den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 2, 2004, pp. 1598–1605 vol.2.