Multi Agent path planning using SIPP

FINAL PROJECT

Of Planning Algorithms in Artificial Intelligence

G2
Ngoc Bich Uyen Vo
Tigran Ramazyan
Xavier Aramayo Carrasco
Alexander Lepinskikh

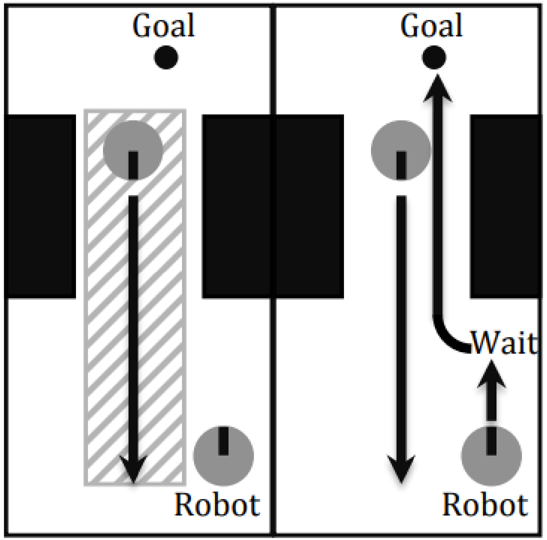
Overview

Introduction

- Multi agent problems can be approached from a dynamic obstacle approach, where the second agent needs to be avoided as all the other obstacles.

- Due to the nature of the problem, most of the solutions are not efficient.

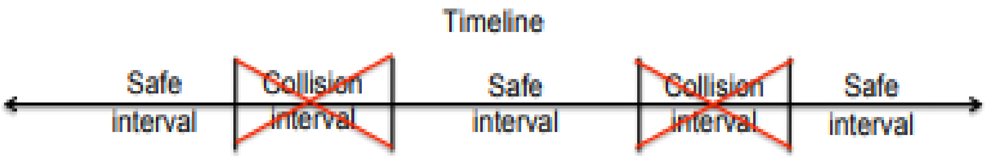
- SIPP (Safe Interval Path Planning for Dynamic Environments) solves the problem by proposing an approach based on safe intervals with no collisions.



Introduction

Each spatial configuration has a timeline that follows one rule:

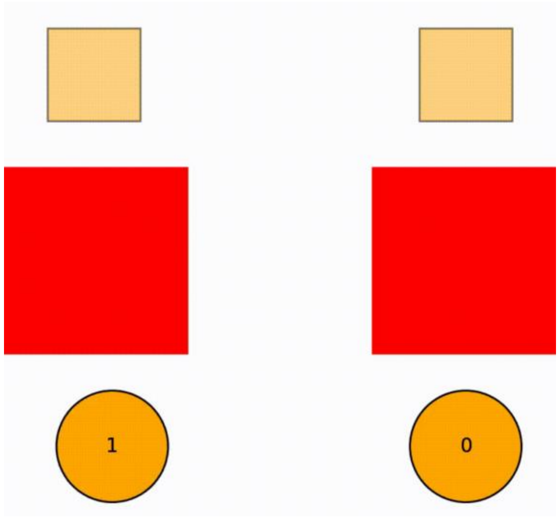
a safe interval is a contiguous period for a configuration during which there is no collision, and it is in collision one timestep prior and one timestep after the period.



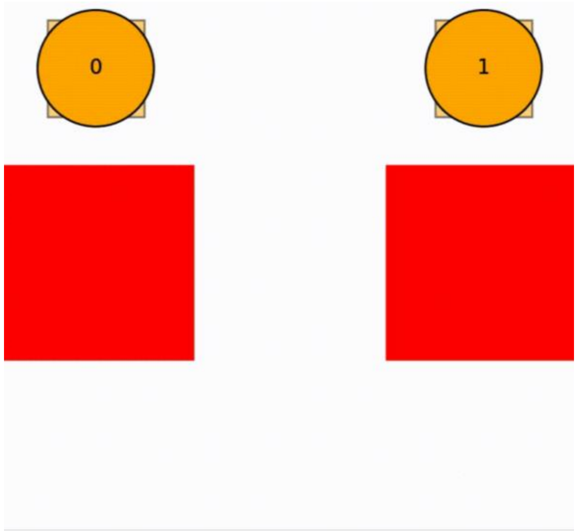
The same rule can be adopted for the **collision intervals**.

Problem statement

We propose some different scenarios to be solved by the algorithm, in these test cases the second agent is considered as a dynamic obstacle, we aim to go from state A (initial) to B (goal)



A



B

Methods

Skoltech
Skolkovo Institute of Science and Technology

- *Graph Construction.* When the planner is initialized, we create a timeline for each spatial configuration, using the predicted dynamic obstacle trajectories.
- *Graph Search.* After the initialization, we run the A* search.

```
g(s_start) = 0; OPEN = ∅;
insert s_start into OPEN with f(s_start) = h(s_start);
while(s_goal is not expanded)
  remove s with the smallest f-value from OPEN;
  successors = getSuccessors(s);
  for each s' in successors
    if s' was not visited before then
      f(s') = g(s') = ∞;
    if g(s') > g(s) + c(s, s')
      g(s') = g(s) + c(s, s');
      updateTime(s');
      f(s') = g(s') + h(s');
      insert s' into OPEN with f(s');
```

```
getSuccessors(s)
  successors = ∅;
  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;
```

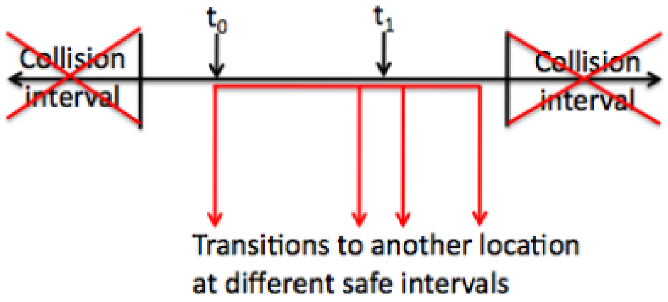
Methods

Skoltech
Skolkovo Institute of Science and Technology

- *A** works as usual but it implements a new get successor function.
- The function M(s) returns the motions that can be performed from state s.
- The *startTime(i)* returns the start time of safe interval *i* and *endTime(i)*, the end time of safe interval *i*.
- When a state s is expanded, we generate successors for it. For each of the motions that our robot can perform from s.
- Then for each of the time intervals in this new configuration, we generate a successor with the earliest possible arrival time that does not have any collisions along the motion.
- When generating successors, s uses “wait and move” actions.

Methods (notes and observations)

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



Since s exists in a safe interval, the robot can wait from any time in the interval to any later time in the interval, before moving.

Methods (notes and observations)

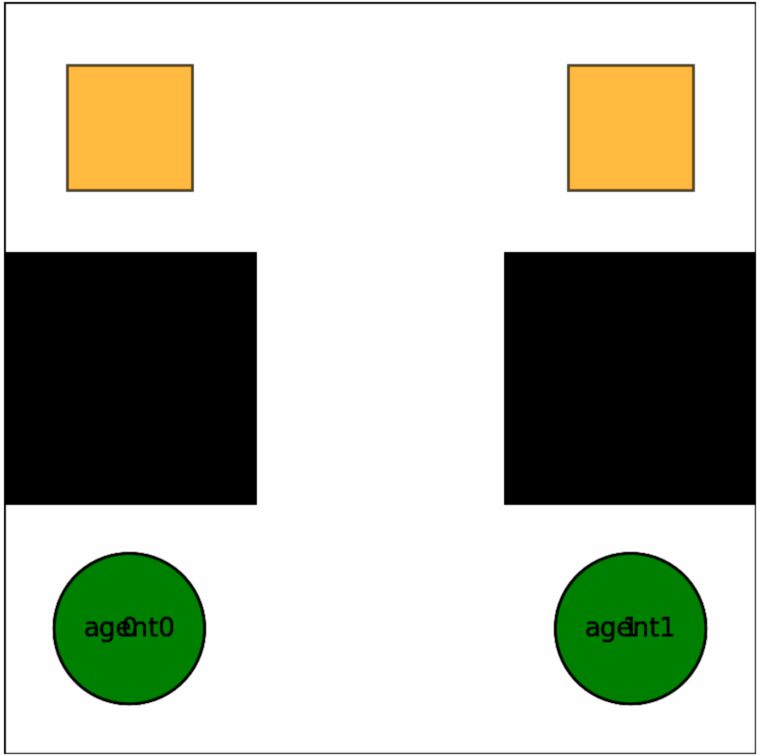
- **Theorem 2:** When the safe interval planner expands a state in the goal configuration, it has found a time-minimal, collision-free path to the goal.

Inherited from A*. In this planner, the cost on an edge is equal to the time it takes to execute that edge and whenever a g-value (cost of best-known path) is updated (from finding a shorter path), the time value is also updated to the earlier time.

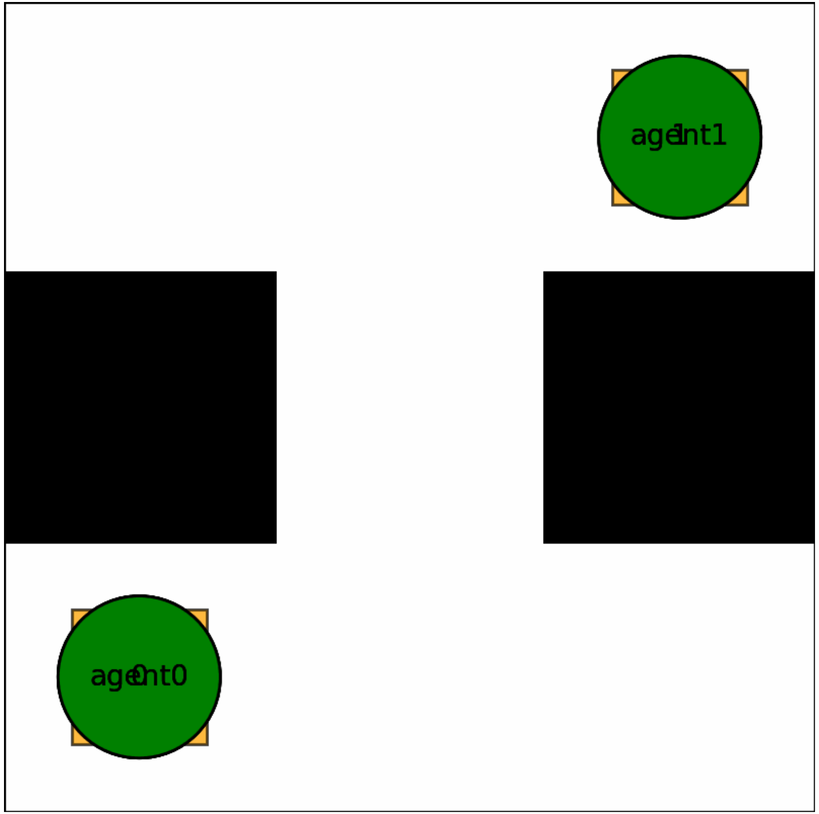
- **Theorem 3:** If the configuration with the most dynamic obstacles passing through it has n such occurrences, then each configuration can have at most n + 1 safe intervals.

Experiments - 1

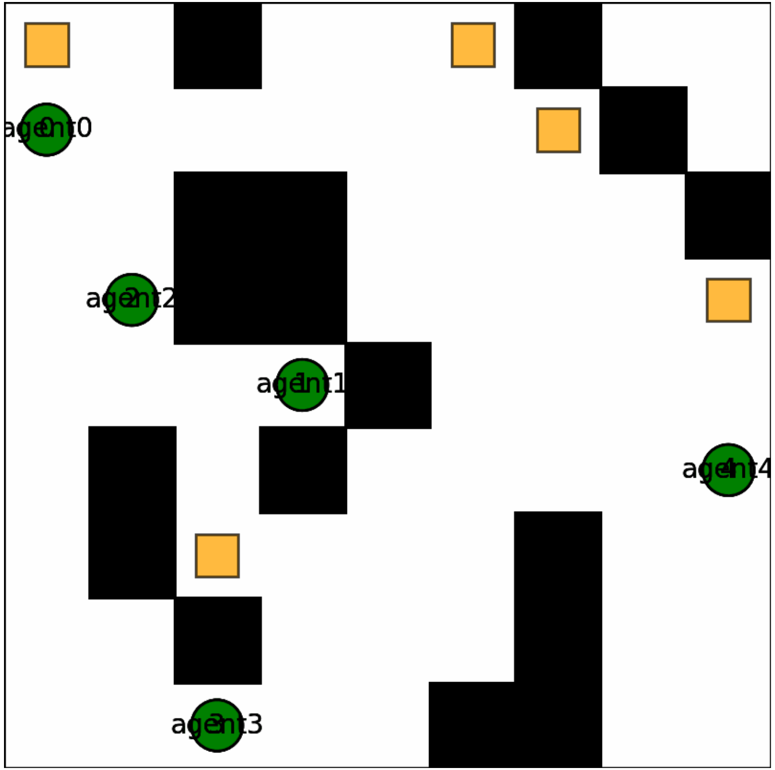
Skoltech
Skolkovo Institute of Science and Technology



Experiments - 2



Experiments - 3



Conclusions

- Using SIPP for solving multi-agent problems is a good choice since it is able to deal with more than a couple of agents and in different environments.

- During the experiments, we could notice that the algorithm fails sometimes, this happens since it fails to predict a future collision with the other agent and both reach to an unscabaple state.

- The model is efficient, it takes some seconds to find the desired path.

Thank You!