School of Computer Science (BICA)
Monash University

Literature Review, 2017

Review of optimal multi-agent Pathfinding algorithms and usage in warehouse automation

Phillip Wong

Supervisors: Daniel Harabor,
Pierre Le Bodic

Contents

1 Introduction

In our multi-agent pathfinding problem, we have an environment containing a set of k agents on a 4 directional grid-map. Each agent aims to find a path to their goal such that no agent will collide with another agent at any time.

Hence we have a centralized agent coordinator which aims to resolve path collisions.

1.1 Problem Overview

Inputs:

- Grid map
- Set of agents with
  - Start location
  - Goal location

Outputs:

- Collision free path from the start to goal location for each each agent

INSERT IMAGE OF EXAMPLE PROBLEM

1.2 Use of Mixed Integer Programming

Branch and bound
Column generation
Branch and price

2 Related Work

CBS
ICTS
Centralised A*
That Network Flow paper

3 Algorithm

Each agent knows of a number of paths to the goal. Using a Mixed Integer Program (MIP) we are able to solve the assignment problem of choosing a combination of paths such that no assigned two paths has a collision. Hence the two main parts of the algorithm:

Any specifics about the MIP I should mention here?

1. **Path Assignment** The Mixed Integer Program, Responsible for choosing a assigning paths to agents, while finding a conflict-free solution.
2. **Path Generation:** Responsible for generating alternative paths for agents whose current paths are in collision.

3.1 High Level Description

At the high-level we describe the cycle of path generation and assignment.
Initializing: Starting on line **??**, we find the shortest path to each agent's goal and add it to their path bank. A path bank is simply a pool of valid paths to each agent's goal, these paths can be non-optimal.
Calling the MIP: Starting the loop, we call AssignPathsMIP. Here we create and run the Mixed Integer Program. This method will choose suitable paths from each agent's path pool. Importantly we have not yet described the second parameter, pathCollisions. Hence we do not know if the assignment is collision-free.
Checking collisions: So we must check if the solution the MIP gave us is valid.
No collision?: If there was no collision, then the solution the MIP assigned us was valid and we can terminate the algorithm.
Collision occured: If there was a collision, we must perform extra steps. The goal here is to penalize this collision and generate new alternative paths. Hence we add this collision to the pathCollisions (used in the AssignPathMIP method), this later creates a hard constraint saying path i for agent n and path j for agent m cannot exist at the same time. Hence the MIP will not be able to choose these two in combination.
Choosing an agent: This method deals with choosing the next agent who we generate a path for. Here we choose the agent with the smallest Δ. This value describes the length between the shortest path and the current path. By choosing the agent with the smallest Δ we can ensure optimality as the makespan of the solution grows incrementally *IMPORTANT: describe this better.*
Penalties: Now we penalize the collision. We do this by looking at the perspective of the chosen agent. So for a collision between $a_n p_i$ and $a_m p_j$, say we have chosen a_n. Hence we are interested in penalizing against p_j. The penalty applies a negative value to any action that could lead to a collision against p_j **EXAMPLE HERE OR LINK TO EXAMPLE?**.
Looping: Now we repeat this iteration of assignment and applying penalties. By ensuring that the makespan of the solution space grows incrementally by 1. We can generate an optimal solution, as the MIP will find a possible combination of paths from each agent's path bank.

Procedure 1 high-level algorithm

Input: *agents* : list of agents
Output: an assignment of collision-free paths

```
find and store shortest path for each agent
loop
    run the MIP and apply the path assignment solution
    check the path assignment for collisions
    if there are no collisions then
        return current path assignments
    else
        get the path collision with the lowest Δ
        for the two agents in this collision: find and store an alternative path
        create path constraints from this path collision and apply to the MIP
```

Procedure 2 high-level algorithm

Input: *agents* : list of agents

Output: *void*

```
pathCollisions ← {}
penalties ← Map < Agent, Penalties > ()

for agent : agents do
    agent.pathBank.add(ShortestPath(agent))

loop
    assignedPaths ← AssignPathsMIP(agents, pathCollisions)
    collision ← CheckForACollision(assignedPaths)
    if collision is not none then
        pathCollisions.add(collision)
        chosenAgent ← ChooseAgent(collision)
        UpdatePenalties(chosenAgent)
        chosenAgent.pathBank.add(GenerateNextBestPath(agent, penalties))
    else
        break
```

3.2 Low Level Description

Here we overview the specifics of our algorithms and possible alternatives.

3.2.1 Path Generation

This method aims to generate paths for an agent. The first iteration of the path generation will use A* to quickly find the shortest path to the goal. This path will likely be in conflict. Next we use Temporal A* which aims to avoid collisions by applying penalties to tiles which are in conflict. We increments the penalties by 1 and iteratively finds paths of the next cost.

A generated path may have a longer length than the optimal single-agent solution. Hence we describe this difference to the optimal path as (Δ). We do not generate a path for this agent until other agents in the deadlock have the same Δ. In this way we find an optimal solution.

However our implementation of Temporal A* is not complete. There are instances which it is unable to find a solution. If the optimal shortest path requires the agent to move past the goal tile. Then Temporal A* fails and will never expand past the goal. An example of this is can be seen in Figure 1. Hence in this algorithm we determine that agents are in **deadlock** when they have $n = 100$ number of collisions. If this occurs we start finding paths with Centralized A*.

$a1$		
$t2$		$t1$
		$a2$

Table 1: A deadlock occurs here. The optimal solution is
a1: d, r, r, u, d, w
a2: w, w, w, u, l, l
This deadlock occurs as the solution for a1 requires the algorithm to search past the goal.

Procedure 3 GenerateNextBestPath

Input: *agent*, *penalties* : tile penalties to be used for the TemporalAStar pathfinder.
Output: *path* : the generated path

$pathCollisions \leftarrow \{\}$
for agent a in $\vec{a}$ **do**
  $path \leftarrow \{\}$
  if $\exists$ agents in deadlock with a **then**
    $path \leftarrow Dijkstras(agents\ in\ deadlock)$
  else
    $path \leftarrow TemporalAStar(agent, penalties)$
  return *path*

3.2.2 Path Assignment

Here we are given paths The goal of this method is At the core, this method simply calls the Master problem (5). The objective here is to assign paths to agents which look suitable. This solution may be invalid but will be checked later. These agents are assigned the penalty variable by the mip and are returned as output of this method.

Procedure 4 AssignPaths

Input: $\vec{a}$: list of agents $\vec{c}$: list of collisions
Output: $\vec{p}$: list of agents who were assigned the penalty

$solution \leftarrow RunMasterProblem(\vec{a}, \vec{c})$
for a in *solution.assignedAgents* **do**
  assign collision-free path to a as described by *solution*
return *solution.penaltyAgents*

3.3 Alternative paths

Here we describe our method of generating the next-best alternative path. An agent has a bank of paths, we aim to generate an alternative path of equivalent length (if one exists), otherwise a path of 1 more in length.

Here we describe our Time-expanded A* implementation with the use of **tile penalties**. In the high-level description of the algorithm, we detect a path collision. At this stage we create penalties. A collision occurs between two paths p_1 and p_2. When generating an alternative for p_1 we penalize the entirety of the path p_2. For example, between these two paths:

- $p_1 : \{1, \mathbf{2}, 5, 8\}$
- $p_2 : \{5, \mathbf{2}, 1\}$

Here a collision occurs at time 2 between p_1 and p_2. Hence we generate a path for both p_1 and p_2. In order to do so, we apply penalties.

We do this applying a penalty to tiles when collisions occur. Collision detection is described in Section 3.2.1.

The temporal A* algorithm aims to find a path from start to goal. Additionally our variant takes in a set of collision penalties. A collision penalty describes that the agent should try to avoid moving from tile a to tile b at timestep t. The collision penalty is used to calculate the f value for each node and determines the priority for expansion.

struct {}

Procedure 5 Temporal A* Node

Input: *parent*, *tile*, *goal*, *penalty*
Output: *Node*: a new Temporal A* Node

$h \leftarrow Heuristic(tile, goal)$
if *parent* is *null* **then**
    $time \leftarrow 0$
    $g \leftarrow 0$
else
    $time \leftarrow parent.time + 1$
    $g \leftarrow parent.cost + 1$
$f \leftarrow g + h + penalty$
return *this*

Procedure 6 Temporal A* comparator

Input: node a, node b
Output: *bool*

if $a.f$ equals $b.f$ **then**
    if $a.penalty$ equals $b.penalty$ **then**
        return $a.g < b.g$
    else
        return $a.penalty < b.penalty$
else
    return $a.f < b.f$

Procedure 7 Temporal A* with collision penalties

Input: s : start, g : goal, $\vec{c}$: penalties, a vector of collision penalties
Output: p : path

$path \leftarrow \{\}$
if s or g is not valid or s equals g **then**
    return *path*
$open \leftarrow PriorityQueue < Node > ()$
push $Node(null, s, g, 0)$ into *open*
while *open* is not empty **do**
    $current \leftarrow open.pop()$
    for *action* in $\{up, down, left, right, wait\}$ **do**
        $nextTile \leftarrow$ perform *action* on *current*
        if $\exists node$ at time $current.time + 1$ on tile $nextTile$ **then**
            update *node* by comparing *node.parent* and *current*, taking the lowest f
        else
            penalty $\leftarrow penalties[current.time + 1][current.tile][nextTile]$
            $newNode \leftarrow Node(current, nextTile, g, penalty)$
while *current* is not *null* **do**
    add *current* to the front of *path*
    $current \leftarrow current.parent$
return *path*

4 Resolving conflicts

1. Given a set of paths, S which contains all agent's path, find a new path for each agent their goal and add it to S
2. Detect any path collision for each path
3. Convert the paths to MIP variables and path collisions to constraints
4. Repeat 1. if there is not a valid solution found i.e the optimal solution contains a path collision

5 Master problem formulation

- Section about where path constraints are generated and variables are added
- Bender's Decomposition
- Talk about the dummy variable

Each agent is given *one or more* paths to their goal. The master problem aims to assign one path to every agent while minimizing the path distance and avoiding path collisions.

- **Agents:** The MIP is given a set of agents, A
- **Path bank**: Every agent has a set of paths, P describing a unique path from an agent's position to their goal.
- **Dummy variable**: A dummy variable, q_i is added for every agent in the case that all the agent's paths are in collision. If the MIP chooses this penalty in the solution, then the MIP was unable to find an assignment of paths which resulted in a conflict-free solution. The cost of the penalty is set to be larger than the expected maximum path length (here it is 1000).
- **Path Constraints:** A set of path constraints, C is provided to the MIP. A constraint describes that two paths n and m are in collision and hence are not allowed to be assigned at the same time.

Line 1: First we define our minimization function. Here we have our path variables p_{ij} which is multiplied by the cost of the path d_{ij}. We sum these for each path in each agent's path bank. Additionally we define a dummy variable q_i for when the MIP fails to find a solution.
Line 2: Here we state that every agent must choose one of their paths p_{ij} or take the dummy variable, q_i.
Line 3: Here we state that every agent must choose one of their paths or take the dummy variable.
Line 4, 5: Here we state that all variables must be 0 or 1.

$$\min \sum_{i \in A} \sum_{j \in P_i} (d_{ij} * p_{ij}) + q_i \tag{1}$$

$$\text{subject to} \sum_{j \in P_i} (p_{ij}) + q_i = 1, \forall i \in A \tag{2}$$

$$n + m \leq 1, \forall (n, m) \in C \tag{3}$$

$$p_{ij} \in 0, 1, \forall j \in P_i \forall i \in A \tag{4}$$

$$q_i \in 0, 1, \forall i \in A \tag{5}$$

For example **?**, our generated variables are: $5a_0p_0 + 5a_0p_1 + 1000q_0 + 2a_1p_0 + 2a_1p_1 + 1000q_1$.

	1	2
3	4	5
	7	8

Table 2: A simple MAPF problem. $a_1 : (1, 5), a_2 : (3, 8)$

5.1 Simple examples

Simple example To get a better intuition of the algorithm, here we step through a simple example.

1. **Initializing:** Generate and store the shortest path in each agent's path bank for **all** agents.
   - $a_1p_1 : \{1, 4, 5\}$
   - $a_2p_1 : \{3, 4, 7, 8\}$
2. **Assign paths MIP:** Create a MIP from the agent's path bank and path constraints (currently none) and use the solution to assign paths to agents
   - $a_1assigned : a_1p_1 : \{1, 4, 5\}$
   - $a_2assigned : a_2p_1 : \{3, 4, 7, 8\}$
3. **Check for a collision:** Here we check the MIP path assignment by finding the first collision (if one exists). There is a path collision at time 1 where a_1 and a_2 are both on tile 4.
   - Collision(a_1p_1, a_2p_1, time: 1, tile: 4, Δ: 0)
4. **Choosing a collision:** We choose the collision with the lowest Δ. Currently there is only one collision to choose from which is:
   - Collision(a_1p_1, a_2p_1, time: 1, tile: 4, Δ: 0)
5. **Adding a constraint:** Since a collision exists, we now apply a constraint to the tell the MIP that a_1p_1 and a_2p_1 cannot be assigned at the same time.
   - Constraint(a_1p_1, a_2p_1)
6. **Penalties:** In order to generate the next best path, we apply penalties at the point of collision. Hence we penalize all **edges** that lead to tile 4 at time 1.
   - Edge ($1 \rightarrow 4$) has penalty 1 at time 1
   - Edge ($7 \rightarrow 4$) has penalty 1 at time 1
   - Edge ($4 \rightarrow 4$) has penalty 1 at time 1
   - Edge ($5 \rightarrow 4$) has penalty 1 at time 1
   - Edge ($3 \rightarrow 4$) has penalty 1 at time 1
7. **Generate next best path:** Here we generate and store the next best path. Unlike step 1, we now have penalties applied to certain tiles which lead our pathfinder to avoid collisions. We generate paths for the two agents involved in the lowest Δ collision. We keep the old paths in the agent's path bank as they are still potential collision-free options.
   - $a_1p_1 : \{1, 4, 5\}$
   - $a_1p_2 : \{1, 2, 5\} \leftarrow$ new path
   - $a_2p_1 : \{3, 4, 7, 8\}$
   - $a_2p_2 : \{3, 3, 4, 7, 8\} \leftarrow$ new path

8. **Assign Paths MIP:** Now that we have added additional paths to the agent's path bank. We now update our MIP, run it and assign solution to agents. Here the MIP has now assigned.
    - $a_1 assigned : a_1 p_2 : \{1, 2, 5\}$
    - $a_2 assigned : a_1 p_1 : \{3, 4, 7, 8\}$
9. **Check for a collision:** Here we check the MIP path assignment for any collisions. There are no collision so our loop terminates.

5.1.1 Complex example

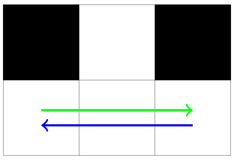
1	2	3
	4	

Table 3: A complex MAPF problem. $a_1 : (1, 3), a_2 : (3, 1)$

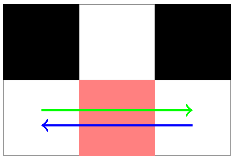
Here the optimal solution for 3 is:

- $a_1 assigned : \{0, 1, 4, 4, 1, 2\}$
- $a_2 assigned : \{2, 2, 1, 0\}$

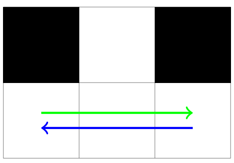
1. **Shortest path:**
   - $a_1 p_1 : \{1, 2, 3\}$
   - $a_2 p_1 : \{3, 2, 1\}$
2. **Path assignment:**



   - $a_1 assigned : \{1, 2, 3\}$
   - $a_2 assigned : \{3, 2, 1\}$
3. **Check for collisions:** Here we check the MIP path assignment by finding the first collision (if one exists).

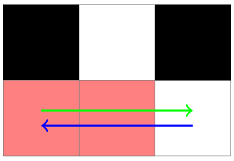


   - $a_1 assigned : \{1, \mathbf{2}, 3\}$
   - $a_2 assigned : \{3, \mathbf{2}, 1\}$
   - Collision at time 1 on tile (1, 0)
4. **Choose path collision with lowest Δ:**
   - Only one collision to choose from with Δ of 0: $a_1 p_1, a_2 p_2$
5. **Find alternative path for agents in collision:**
   - $a_1 p_2 : \{1, 1, 2, 3\}$
   - $a_2 p_2 : \{3, 3, 2, 1\}$
6. **Create path constraints and apply to the MIP**
   - $Not(a_1 p_1, a_2 p_2)$
7. **Path assignment:**



- $a_1 assigned : \{1, 1, 2, 3\}$
- $a_2 assigned : \{3, 2, 1\}$

8. **Check for collisions:**



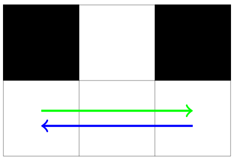
- $a_1 assigned : \{1, \mathbf{1}, \mathbf{2}, 3\}$
- $a_2 assigned : \{3, \mathbf{2}, \mathbf{1}\}$
- Collision at time 2 on tile (1, 0) and tile (0, 0)

9. **Choose path collision with lowest Δ:**
   - Only one collision to choose from with Δ of 1: $a_1 p_1, a_2 p_2$

10. **Find alternative path for agents in collision:**
    - $a_1 p_2 : \{1, 1, 2, 3\}$
    - $a_2 p_2 : \{3, 2, 2, 1\}$

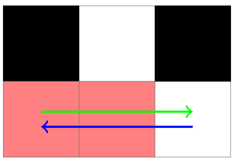
11. **Create path constraints and apply to the MIP**
    - $Not(a_1 p_1, a_2 p_2)$

12. **Path assignment:**



- $a_1 assigned : \{1, 2, 3\}$
- $a_2 assigned : \{3, 2, 2, 1\}$

13. **Check for collisions:**



- $a_1 assigned : \{1, \mathbf{2}, 3\}$
- $a_2 assigned : \{3, \mathbf{2}, 2, 1\}$
- Collision at time 1 on tile 2

14. **Choose path collision with lowest Δ:**
    - Only one collision to choose from with Δ of 1: $a_1 p_1, a_2 p_3$

15. **Find alternative path for agents in collision:**
    - $a_1 p_3 : \{1, 1, 1, 2, 3\}$
    - $a_2 p_3 : \{3, 2, 1, 1\}$

16. **Create path constraints and apply to the MIP**

- $Not(a_1p_1, a_2p_2)$

17. **What path / penalty would give the path $\{1, 2, 4, 2, 3\}$?**
    - Penalty on tile 3 timestep 2
    - Penalty on tile 2 timestep 3
    - Hence the path would look like: $\{3, 3, 3, 2, 1\}$
18. **What path / penalty would give the path $\{3, 3, 3, 2, 1\}$?**
    - Penalty on tile 2 timestep 2
    - Penalty on tile 2 timestep 3
    - Hence the path would look like: $\{1, 2, 2, 3\}$

References