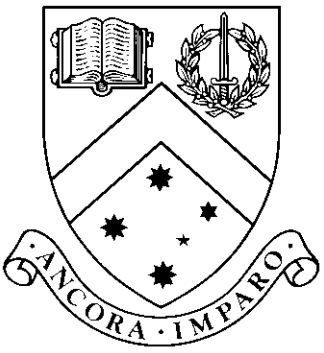
School of Computer Science (BICA)
Monash University




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 grid-map. Each agent aims to find a path to their goal without colliding with another agent (a path collision).

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

# 2 Related Work

**CBS**

**ICTS**

**Centralised A***

**That Network Flow paper**

# 3 Algorithm

The algorithm is based off branch and bound...¡Explain how?¿. Overview: Each agent knows of a number of paths to the goal. Using a Mixed Integer Program we are able to quickly find a combination of path assignments such that no assigned path has a collision.

# 4 Collision-free Paths

This section outlines the slave algorithm which is used to generate paths. Paths are then given to the Mixed Integer Program described in Section 6 which handles the assignment of paths to agents.

## 4.1 Agent Coordinator

The agent coordinator is the highest level method where paths are generated and assigned.

**Procedure 1** AgentCoordinator

**Input:** $\vec{a}$ : list of agents
**Output:** *void*

1: $collisions \leftarrow \{\}$
2: **do**
3: $\quad \vec{ap} \leftarrow GeneratePaths(\vec{a})$
4: $\quad AssignPaths(\vec{ap})$
5: $\quad failedAgents \leftarrow AssignPaths(paths,\ \vec{ap})$
6: **while** *collisions is not empty()*

## 4.2 Generating Paths

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. This method increments the penalties by 1 and iteratively finds paths of the next cost.

If an agent's new path has an increase in cost from the optimal path ($\Delta$). We do not generate a path for this agent until other agents in the deadlock have the same $\Delta$. In this way we find an optimal solution as the path increments slowly?

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 ¡HERE¿! 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*.

**Procedure 2** GeneratePaths

**Input:** $\vec{a}$ : list of penalty agents $c$: counts the collisions between agents
**Output:** $\vec{ap}$: list of paths which are in collision: a tuple of (agent, path)

```
ap⃗ ← {}
for agent a in a⃗ do
    path ← {}
    if agent.paths is empty then
        path ← AStar
    else if ∃ agents in deadlock with a then
        path ← CentralizedAStar(agents in deadlock)
    else
        path ← TemporalAStar
    path ← GeneratePath(a, firstRun)
    a.p⃗.append(path)
    ap⃗.append(a, CheckCollisions(path))
```

### 4.3 Path Assignment

At the core, this method simply calls the Master problem (6). The objective here is to detect if any agents were unable to be assigned paths (meaning there was no combination of paths which resulted in a collision-free solution). These agents are assigned the penalty variable by the mip and are returned as output of this method.

**Procedure 3** AssignPaths

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

```
solution ← RunMasterProblem(a⃗, c⃗)
for a in solution.assignedAgents do
    assign collision-free path to a as described by solution
return solution.penaltyAgents
```

## 5 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

## 6 Master problem formulation

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.

- **Potential paths**: A set of paths from an agent's position to their goal. We generate a variable for each path and the cost is set to the path length.
- **Penalty**: A penalty variable is added for every agent in the case that all the agent's paths are in collision. The cost of the penalty is set to be larger than the expected maximum path length (here it is 1000).

We specify an agent's path as $p_{ij}$. Penalty $q_i$. Path collision as $c_{nm}$.

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

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

$$\sum_{m \in S_n} (p_{nm}) \leq 1, \forall n \in C \tag{3}$$

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

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

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

Agents are assigned

## References