Multi-Agent Path Planning for Warehouse Butlers

Anchita Goel 2012019, Sarthak Ahuja 2012088

Abstract

Our problem environment resembles to that of a grid-map warehouse where in there are multiple agents (butlers) trying to transport the requested items from the conveyor belt to the requested positions. The items in the warehouse are spread over the entire area of the warehouse organised in racks. The communication among these butlers will be decentralised in the sense that each robot will try to optimise its own path to the goal state. The aim of each of the butlers will be to reach the belt and the requested positions without colliding with any of the other butlers in the shortest time/path possible.
Some of our assumptions in the beginning will be:

1. All butlers are identical and follow the same algorithm.
2. The warehouse is of the form of a grid-map.
3. Each item in the warehouse are capable of being carried by a single butler.

1. Motivation

The boom in the e-commerce industry has lead to cropping up of a large number of warehouses all over the globe. Automating the processes in these warehouses is a growing requirement to achieve a reduced cost in terms of manpower and increased efficiency in terms of time taken. The problem of multi-agent path planning can be used in many such areas and others like - computer games (like Pacman), transportation networks etc. Hence, we would like to work on a solution which partially tries to solve this problem.
Naive approaches such as a complete A* over all combination of butlers and targets will not work in this case due to the huge statespace being created. Neither will Local Repair A* where each butler will selfishly move towards the target and replan only on collision in a blindfolded fashion. This type of problem requires some more sophisticated approaches. On doing a literature survey we came across many popular approaches like RRT[1], WHCA*[2], FAR[3]. But none of these approaches were guaranteed completeness or explained situations in which they could. We found MAPP[4] as an interesting topic for this project due it's defined completeness over a certain set of maps and it's very intuitive algorithm. It is also an approach that came after WHCA* and FAR showing considerable improvements, which further motivated us to explore this algorithm.

2. State-of-the-Art

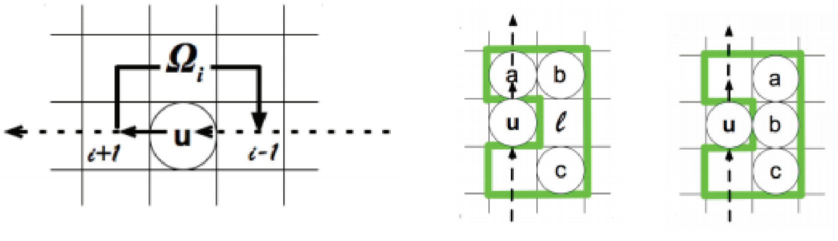
One of the state-of-the-art algorithms for multi-agent path planning is the solution proposed by Ko-Hsin Cindy Wang and Adi Botea.[4] The solution doesn't not guarantee any optimality but ensures completeness over a range of problems it calls SLIDABLE. A setting of a warehouse belongs to an instance of SLIDABLE problem if it satisfies the three constraints : target isolation, alternate connectivity and initial blank. This is the algorithm that we plan to implement to understand exactly what instances belong to the SLIDABLE class of problems. Details about this algorithm are given below.

3. Description of the Algorithm

The solution mentioned above needs butlers to be given priorities. Giving them priorities ensures that there are no cycles because a butler with low priority never interferes with the movements of a butler with high priority. In our implementation we have assigned the butlers priorities randomly. However, there is also a provision for assigning them priorities as per wish.
Let the path for a butler u be represented by $\pi(u) = (l_0^u, l_1^u, ..., l_{|\pi(u)|}^u)$ where l_0^u is the start position and the $l_{|\pi(u)|}^u$ is the destination position.
MAPP requires to calculate for each butler a path from their initial position to the designated position such that the path satisfies the following constraints:

1. **Initial Blank:**The first position after start position i.e. l_1^u is blank.
2. **Alternate Connectivity:** For $0 < i < |\pi(u)| - 1$, there must exist an alternate path Ω_i^u such that the alternate path doesn't go through l_i^u.
3. **Target Isolation:** Neither the $\pi(u)$ or Ω_i^u for any butler u must go through the destination position of another butler with higher priority.

If the above mentioned conditions satisfy, then the problem is SLIDABLE and MAPP can solve the problem. Finding paths and alternate paths is done through $A*$ search taking *Manhattan distance* as a heuristic. Half of the problem of finding paths is solved in trying to check whether the problem is SLIDABLE or not.



(a) (b)

Figure 1: Part(a) depicts the existence condition for an alternate path while part(b) depicts how sliding along the alternate path happens

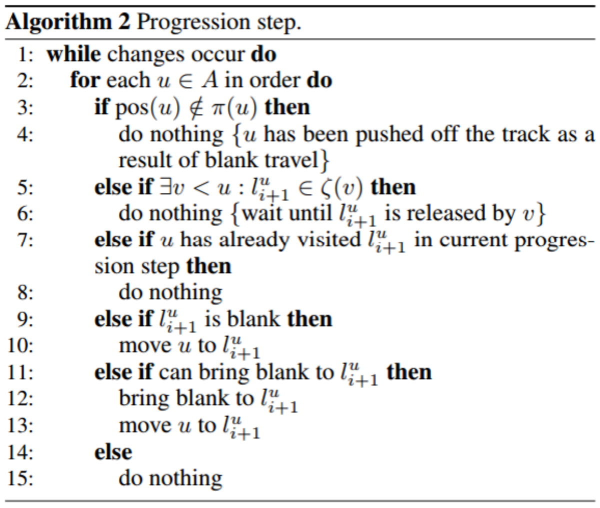
The next step involves a description of how the butlers would move along their $\pi(u)$ and in case of a possible collision, resolve the conflict.

Moving the butlers is divided into two parts. The below steps are carried out one after the anothe till no butler is active (i.e. all butlers have reached their destination).

3.1. *Part I: Progression*

This step maintains a list of all the butlers that are still active (i.e. haven't reached their target). For all active butlers it checks if the butler is not on its intended path, or if its next position will be the next position for any of the butlers with higher priority, the butler stays put. However, if its next position is occupied by a butler with lower priority, it forces the butler to move down its alternate path for the current location and itself moves to the blank position. Otherwise if it is on its intended path and can move to the next location, it moves ahead. The algorithm can be summarised as depicted in figure: 2.

Figure 2: Progression Algorithm



Algorithm 2 Progression step.

```
1: while changes occur do
2:   for each $u \in A$ in order do
3:     if pos$(u) \notin \pi(u)$ then
4:       do nothing {$u$ has been pushed off the track as a
         result of blank travel}
5:     else if $\exists v < u : l^u_{i+1} \in \zeta(v)$ then
6:       do nothing {wait until $l^u_{i+1}$ is released by $v$}
7:     else if $u$ has already visited $l^u_{i+1}$ in current progres-
       sion step then
8:       do nothing
9:     else if $l^u_{i+1}$ is blank then
10:      move $u$ to $l^u_{i+1}$
11:    else if can bring blank to $l^u_{i+1}$ then
12:      bring blank to $l^u_{i+1}$
13:      move $u$ to $l^u_{i+1}$
14:    else
15:      do nothing
```

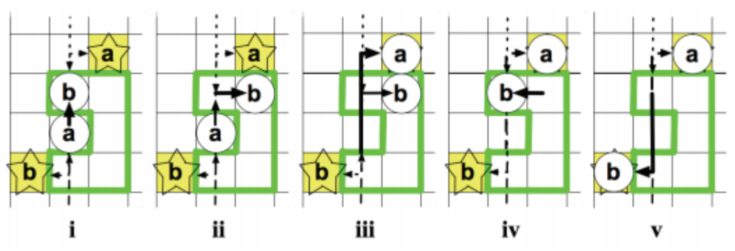
3.2. *Part II: Repositioning*

For all the butlers that are active till now and were moved because of butlers with higher priority, the butlers are moved back to the positions that they were earlier in. **??** depicts the algorithm in action.

4. Contribution

We have implemented the entire Basic MAPP algorithm and prepared a Simulation bench to run any placement of walls, butlers and items for further evaluation. [4] also suggests the algorithms for the *Progression* step when the two conditions *Target Isolation* and *Alternate Connectivity* are relaxed. Targets of other butlers almost act as wall for other butlers, thereby decreasing the space further that is available to move. Relaxing the target isolation condition as found out in [4] leads to an increase in the number of instances that can be solved. However, we have not implemented the relaxed problems. Please refer to the Implementation section for more details about the code.

Figure 3: The MAPP Algorithm in Action - A low priority butler is slided down the alternate path to create a blank space for the high priority butler



5. Evaluation

5.1. Completeness and Optimality

As stated before the algorithm is complete if the warehouse satisfies the *slidability* constraint though no guarantees can be given on optimality of the path. For each butler there is an alternate path to go to it's previous state which allows avoidance of collision every time.

5.2. Complexity

The algorithm runs in $O(n^2m^2)$ time where m is the number of available states and n is the number of butlers. The Space complexity is $O(nm^2)$.

5.3. Observations

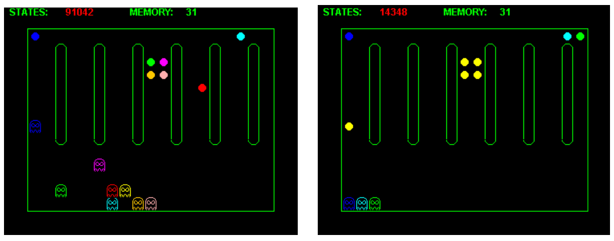
We have created many instances of the setting of the warehouse where MAPP can solve the problem and where it cannot. One can try out our implementation with different settings and number of butlers. We also show the number of states expanded in solving an instance and the memory used. The butlers are assumed to start from the bottom left corner of the warehouse. Based on our observations and experimentation, finding a solution with MAPP algorithm depends on the following factors:

5.3.1. Slidability

As discussed earlier the main criteria on deciding the solvability of a warehouse setting is equivalent to finding the slidability of the setting. Figure 4 depicts examples of slidable and non-slidable settings. One can observe that in part(b) of the figure due to no alternate path being found by the corner two item's and their assigned butlers.

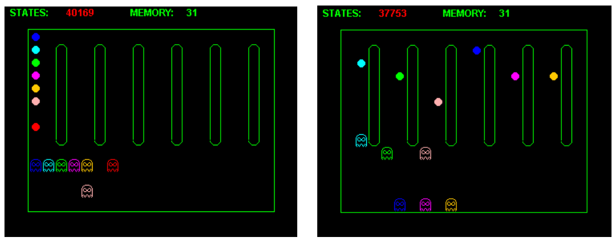
5.3.2. Proximity of Targets

Closely packed targets often lead to slow results as they require frequent repositioning - Low Priority butlers have to make way for high priority butlers every time. Figure 5 depicts the situation. Going for items separated into different sections of the warehouse leads to lesser repositioning situations.



(a) (b)

Figure 4: Examples of slidable and nonslidable instances

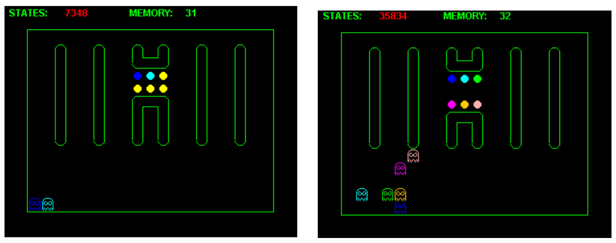


(a) (b)

Figure 5: Dependence on proximity of targets.

5.3.3. Channel Width

Channel width has a major impact on the success of the algorithm. Low channel width with high number of items often violates the slidability condition as depicted in part(a) of figure 6. Adding a channel between the items makes the situation slidable and allows the algorithm to proceed.

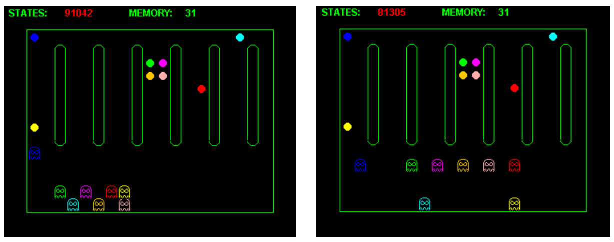


(a) (b)

Figure 6: Dependence on channel size

5.3.4. Placement of Work Stations

Having workstations placed further apart from each other improves the situation in terms of states expanded and time taken. This is due to the fact that initial collisions are avoided. Figure 7 depicts the same.



(a) (b)

Figure 7: Dependence of workstation spaciousness

6. Implementation

6.1. UML

We took apart only the UI component of a basic implementation of pacman available at [5] for the basic design of the underlying map and the agent. Major changes have been made to that as well and it comprises of only a minor section of our code. The submission along with this report consists of a png file depicting the UML of the code.

6.2. Code

You can find the code on the github repository - [6] The java code provided follows the above described design and provides the following features:

1. Easy addition of simulations: One can directly add a new warehouse - setting and position of goals and agents in a single file.
2. Calculation of memory and states expanded for each instance.
3. A robust implementation of the MAPP algorithm.
4. Eleven different warehouse settings.
5. Clean and Object Oriented class structure.
6. In our implementation we have used a 21x16 matrix as a model of our warehouse. This is completely customizable.

7. Deliverables

All the cited deliverables in the proposal were achieved in this project:

1. A working simulation of multiple butlers moving around in a warehouse following the constraints laid by us.
2. Provide customizable parameters to experiment and evaluate the results.
3. Provide a detailed log which reports usage of memory and other resources and states in each experimental setup.
4. An evaluation of cases where the algorithm fails to find a solution at all.

8. Future Work

As part of future work, we will look to evaluate the following:

1. Add relaxations for certain cases of *Target Isolation* and *Alternate Connectivity*
2. Add a dynamic model to plan the path back to the workstation. A static approach would allow all the butlers to reach their goals and then run the algorithm again with the initial goals as the new source positions and vice versa. But a more efficient approach would be to solve it in a dynamic fashion i.e. recompute the slidability when a butler reaches it's destination based on the current situation of the other butlers and re-plan it's way back to the conveyor belt.

References

[1] V. R. Desaraju, J. P. How, Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees.

[2] D. Silver, Cooperative pathfinding.

[3] K.-H. C. Wang, A. Botea, Far - flow annotation replanning.

[4] K.-H. C. Wang, A. Botea, Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees, Journal of Artificial Intelligence Research.

[5] RDKRAL, Capstone pacman.
URL `https://github.com/rdkral/Capstone-Pacman`

[6] S. Ahuja, A. Goel, Github repository.
URL `https://github.com/jokereactive/MAPP-Warehouse-Butlers`