Solving MRTA Exploration Problem Using A*

Magus Verma
magus12141@iiitd.ac.in

Shubhorup Biswas
Shubhorup12103@iiitd.ac.in

I. Introduction

Given a graph G = (V,E), a set of starting locations of robots and a set of targets(both subsets of V), we need to plan a schedule for the robots such that each target vertex is visited by some robot at least once and no two robots are ever present on the same vertex at the same time. Movement model : Each robot, in one time step, can move to a vertex neighbouring that which it is currently on(we term the event of 2 robots being present on the same vertex at the same time a collision).

Objective :
We can construct different objective functions, such as

1) Minimise the time at which all targets are picked up
2) Minimise the sum of the distance travelled by all robots
3) Minimise the sum of the times at which the targets are picked up
4) Minimise the sum of the times at which all robots stop moving

3 is different from 4 as a robot need not stop moving after it has visited all its targets. It may, say, move to make way for a different robot.

II. Deconstruction

We can broadly dissect the problem into 3 levels :

1) Allocate targets to robots, i.e. which robot will pick up a certain target
2) Decide the order in which every robot will visit its allocated targets
3) Design a non-colliding schedule for the robots over each segment of their individual journeys.

A related research problem to MRTA is multi robot path planning(MRPP), in which every robot has a start position and a goal position, and they must move from start to goal without colliding. We can see that after we solve steps 1 and 2, we have current locations of robots and their immediate destinations. Thus if we had an MRPP solver then we only need to repeatedly call it after solving steps 1 and 2.

In our work, we only try to solve steps 1 and 2 efficiently since there already exists a large body of work on solving step 3, or the MRPP problem.

III. Method and Approach

We use an A* approach for finding an optimal solution. The different components of A* are:

1) state : A state consists of locations of all robots and locations of remaining targets. A valid state has no robots coinciding with each other and no targets coinciding with robots(if a robot lands on a target it is removed from list of remaining targets on expansion.)
2) goal state : A goal state has 0 remaining targets
3) Neighbouring states : A is a neighbour of B if there is a bijective mapping from robots in A to robots in B such that each preimage is at a vertex neighbouring the location of the image or at the images vertex.
4) Move cost : This depends on the objective function we are minimising over. For function 1) the move cost is always 1. For 2) cost = no. of robots that move. For 3) cost = no. of remaining targets.
5) Heuristic function : To underestimate the actual cost of optimal solution from a certain state, we consider all possible allocations of targets and then solve TSP for each robot individually, thus ignoring any collisions.

The A* approach thus underestimates the cost in case of collision avoidance. Thus the gap between the heuristic cost and actual cost depends on the number of collisions that are expected to occur on the plan computed by A*. If we are minimising objective function 1, then h=minimum tour cost over all robots and all allocations. If it is 2, then h= minimum sum of tour costs for every robot over all allocation. Such a decomposition is possible for all heuristic functions.

IV. Computational Complexity

r: number of robots
t: number of targets

Number of nodes expanded :
If we consider a search tree(worst case assumption) then depth of this tree = No. of states expanded. This will be upper bounded by the optimal solution cost, s.
Arity of tree = average no. of successor states to a certain state. This is upper bounded by

$$\delta_1 * \delta_2 * ... * \delta_r$$

where these are the degrees of the vertices the robots are currently at. Thus this is equal to δ^r where δ = geometric mean of degrees.
Thus the total number of nodes expanded under these worst case assumptions is δ^{rs}

Cost of node heuristic computation :
There are r^t possible allocations as every target can be allocated to any robot. For every allocation, we must compute the optimal TSP tour which costs $v * 2^v$ for v targets allocated, and is upper bounded by $t * 2^t$. Therefore a crude upper bound for cost of heuristic computation is $t * (2r)^t$.

Therefore time compexity = $O(\delta^{rs} * t * (2r)^t)$
And space complexity = O(δ^{rs}).

Quality of the heuristic and its closeness to actual cost affects space complexity as well as time complexity.
Cost of computing heuristic affects time complexity. Within the confines of our approach we can either 1)prune the number of allocations checked or 2)devise a method to quickly compute optimal TSP tour for every robot's allocation. Below we detail a provable approach for 2).

V. TSP-DP

We devised a way to optimise TSP computation cost so that with some precomputation and a DP table, querying the optimal TSP tour for 1 robot and v targets costs O(v) instead of $O(v * 2^v)$.

The DP solution of TSP stores, for each subset and starting city($n * 2^n$ entries), the optimal tour cost for that subset if you start at that city.

For our precomputation we compute the DP array for all targets. Thus we can now query in O(1) the cost of optimal tour of a subset of targets given the starting city.

When we have to compute optimal tour for a robot given its allocation(let's call this subset of targets S), we have to find, for all possible starting cities s∈S, the minimum of : ¡distance from robot to s¿+¡DP[S][s]
(DP[X][y], y∈X stores the optimal cost of traversing all cities in set X, starting from y).

VI. Related result : Objective function 2 is unaffected by collision

If the function we have to minimise is the sum of distances traveled by all robots, then relaxing the condition of no collision will not lower the solution cost. In other words, if we are to optimise cost function no.2 then we do not need to consider collision of robots.

Sketch of proof

Let us consider a 'collision' caused by a schedule which only tries to seek the best task allocation and permutation of allocated tasks for each robot.

Let two robots r_1 and r_2 be present at vertices u_1 and u_2 respectively at some time t and contend for the occupation of vertex v at the next time step t+1.

There are 4 possibilities

1) At time t+2 r_1 and r_2 occupy u_2 and u_1 respectively according to original schedule
This means that being at u_2 takes r_1 closer to completing its goal than being at u_1 and similarly for the other robot. Since the robots are interchangeable, the allocations of the two robots can be exchanged at time t giving us a better solution than before and avoiding collision. This is a contradiction of the assumption that we were given a collision-oblivious optimal schedule to begin with. Therefore such a schedule cannot be optimal.
2) At time step t+2 r_1 and r_2 both occupy u_2 according to original schedule.
It is wasteful in this case for r_2 to travel to v and then go back again as any target here would be picked up by r_1. Thus this schedule cannot be optimal.
3) At time step t+2 r_1 occupies u_2 and r_2 occupies some vertex w according to original schedule, where w is not one of u_1,u_2 or v.
This can be fixed into a noncolliding schedule of the same cost by making r_1 wait at its current vertex for 1 time step and then continuing its schedule as before, delayed by 1 time step. Since it does not cover any distance while waiting, this fix does not add any cost to the collision oblivious solution.
4) At time t+2 r_1 and r_2 occupy w_1 and w_2 respectively according to original schedule, where these vertices are none of u_1,u_2 or v.
In this either vertex could be made to wait as a fix, as above.

Thus we see that in all possible collisions a fix is available that does not increase cost of collision-oblivious schedule.

VII. Experimental Results

We obtained results of running A* search on the given problem with 4 different heuristics. The grid size of environment was varied for this analysis, keeping robots=3 and targets=4 with blocked cell percentage=20. The implementations used for this lacked Dynamic Programming strategy mentioned above, this made the results lack the quality as the search couldn't happen beyond 1000 nodes.

Table 1 and Table 2 summarizes them for problems of last robot reaching time and total travelling/steps taken by robots respectively. The table gives out the state expansions that were performed for achieving the final result given in circular brackets. Many results in this analysis didn't terminate because of our weak/slow implementation, for them a ¿= remark has been added , in such cases the value till last state expansion has been used

VIII. Conclusion

We have developed a strong background of the problem during the project work till now. However, we need to optimize our implementation and add dynamic programming heuristic computation to further enhance our analysis. For doing a full justice to the problem, we need to make our implementations capable of achieving million state expansions in sufficient time. This is then we would be able to obtain better results for existing study presented also exploring the variations of robots and targets.

References

[1] Lagoudakis, M.G. ; Berhault, M. ; Koenig, S. ; Keskinocak, *Simple auctions with performance guarantees for multi-robot task allocation* IEEE , 2004.

[2] Gerkey, Brian P.., and Matari, Maja J. *A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems* Journal Article,The Intl. J. of Robotics Research , 2004.

TABLE I. STATES EXPANSION COMPARISON FOR MAX ROBOT TIME COST

Heuristic Grid Size	10X10	20X20	40X40	60X60	80X80	100X100
No Heuristic	>16309 (>=5)	(>=5)	NA	NA	NA	NA
Least Max TSP Cost Allocation (using manhattan path cost estimate)	15 (6)	505 (8)	>1057 (>27)	NA (>42)	NA(>61)	NA (>92)
Least Max TSP Cost Allocation (using precomputed path cost estimate)	14 (6)	71 (8)	>600 (>=33)	NA (>=58)	NA (>=85)	NA (>=117)
Least Max TSP Cost Allocation (using precomputed path cost estimate) (weight=2)	6	8 (8)	34 (34)	58 (58)	85 (85)	117 (117)

TABLE II. STATES EXPANSION COMPARISON FOR SUM OF ROBOT TRAVEL COST

H	10 X 10	20 X 20	40 X 40	60 X 60	80 X 80	100 X 100
No Heuristic	>7160 (>=8)	(>=5)	NA	NA	NA	NA
Least Sum TSP Cost Allocation (using manhattan path cost estimate)	16 (11)	>1381 (>=19)	(>=52)	NA (>=90)	NA(>=104)	NA (>=168)
Least Sum TSP Cost Allocation (using precomputed path cost estimate)	82 (11)	51 (20)	>1007 (>=60)	NA (>=89)	NA (>=101)	NA (>=167)
Least Sum TSP Cost Allocation (using precomputed path cost estimate) (weight=2)	6 (11)	14 (20)	34 (60)	89 (89)	101 (101)	150 (167)