MATHAND Final Project Write-up

Hillel Hochsztein and Erica Tsai

May 6, 2020

Introduction

Problem Statement: optimal manufacturing is highly dependent on maintaining and distributing proper inventory of each necessary component. A machine is only as fast as the supply of materials it has available. Warehouses are often set up with storage areas, where raw materials are kept, and manufacturing machines, where those materials are turned into usable products. It becomes critical to move supplies from storage to the machines in an optimal way, to prevent downtime.

We imagine a warehouse with four machines, each of which requires its own unique supply material. Material is stored in bins located randomly throughout the warehouse. The machines deplete their supply of material at a constant rate, and the bins replenish their supplies at a slower rate. A robot, which can only carry one type of material at a time (and no more than the maximum amount of the bins) is tasked with transferring the material from supply bins to the machines.

Objective: To build an integrated planner that generates optimal plans for moving material through the warehouse. This planner must identify an order of tasks to complete, and then complete them in the shortest amount of time (i.e. taking the route of the shortest distance).

Approach

Formulation: The robot exists in a 2-D gridworld, looking down at the warehouse. There are static obstacles (the shape of the warehouse) and static goal points (the supply bins' and machines' locations). The task planner starts at state: S_{robot} = (x, y, empty), meaning the current pose of the robot, without any material on it, and attempts to find a path to a goal state: S_{robot} = ($x_{machine}$, $y_{machine}$, $full_{machine's\ material}$). Note, that there are multiple goal states (in our simulation all 4 machines, each with its own material). The travel planner plans from S_{robot}= (x_s, y_s) to S_{robot} = (x_g, y_g) for any two positions *s* and *g*.

Architecture: We structured our task planner as an implicit graph planner (using A*) that searches over the space of possible actions. Actions here, refer to travelling from a specific supply bin to a specific machine or vice versa. The costs in this space are the travel costs calculated using the low level travel planner, and the heuristic is a function that takes into account the remaining quantities of materials in each of the supply bins or machines and distance cost to waypoints. This heuristic is admissible and consistent, which is proven in the Additional Details section.

The goal for the task planner is a warehouse and machine waypoint combination such that distance travelled over benefit (max supply level) is minimized. The heuristic of the task

planner is calculated as $\frac{L}{S}$, where L is the total distance travelled so far and S is the maximum amount of benefit (or supply) the action (waypoint) will generate. Specifically, S is equal to MAX($S_{warehouse}$, $1 - S_{machine}$), where $S_{warehouse}$ is the supply level of the warehouse waypoint and $S_{machine}$ is the supply level of the machine waypoint. Therefore, this heuristic takes into account total distance travelled (which is calculated using a separate A* search) and changing benefit values (depending on what kind of bin we are at). Since the open list sorts by this heuristic, the first machine popped off the top of the list (or machine that the planner converges to) would be the one that minimizes distance travelled over benefit.

A lower level travel planner actually plans the specific path of the robot at any given point. This is a Weighted A* search over the explicit graph generated by an 8-connected grid with unit costs (1 in orthogonal directions, 1.4 in diagonal directions). The search uses a manhattan distance heuristic, and the weighting multiplier, ε, is changeable (part of the discussion in Results). The algorithm interprets the map as a series of binary positions (open or closed), which are checked upon reference to the corresponding position. A path is generated via backtracing, and the function outputs either the cost of a path, for use in the task planner, or the path itself, for use in the simulation.

Finally, the planner is executed in a simulated environment created in matlab. The map is a 1000X1000 grid, designed to have many rows, with random openings between rows (at least one opening is guaranteed for each wall). The 4 machines are placed at a small offset from each corner, and the supply bins' locations are generated randomly (but guaranteed to be in legal positions). An alternative map includes a central aisle through the walls, which is more accurate to typical warehouse layouts, and generally should be a somewhat easier problem for the planner to solve. The simulated environment opens the map and a set of initial conditions for the machines' and supply bins' levels. The simulation starts by calling the planner function and retrieving a path to a supply bin and then to a machine. The simulation steps through each point of the path, updating the levels (depleting machines and replenishing bins) and checking for collisions (confirming that the planner did not output an illegal move) and whether the robot has arrived at the supply bin or the machine. The simulation also draws the map and positions, and updates the colors of each machine and supply bin to match its supply level.

Results

The description for each of the columns in the following tables are as follows:

- ε_{cost} is the multiplier of the heuristic for the cost A* search
- ε_{path} is the multiplier of the heuristic for the actual path A* search
- Task Planner Time is the time it took from the beginning of the entire planner to the output of the task planner over 10 iterations
- Task Planner Nodes Opened is the average number of nodes opened over a 10 iteration run of the planner.
- Path Planner Time is the average time it took from the end of the task planner until the x,y coordinate output of the path planner over 10 iterations
- Total Planner Time is average of runtimes over 10 iterations

Each of the values in the table below is the average result after 10 iterations. Each iteration includes one run of the task planner (including symbolic planner and A* cost heuristic calculations, i.e. the path planner without the backtrace) and two runs of the path planner (to plan between waypoints).

Table 1: Map 1 (without aisle) Average Results after 10 Iterations

Cost A Star Heuristic Multiplier, ε_{cost}	Path A Star Heuristic Multiplier, ε_{path}	Task Planner Time (ms)	Task Planner Nodes Opened (#)	Path Planner Time (ms)	Total Planner Time (ms)
0	0	51.8715	11.2	0.1864	52.0579
0	1	49.9916	11.2	0.1826	50.1742
1	0	57.5744	11.2	0.1649	57.7394
1	1	57.7972	11.2	0.1833	57.9805
2	2	57.8885	10.9	0.1810	58.0696
10	2	45.5760	10.7	0.1070	45.6933
10	10	45.7886	10.9	0.1071	45.8958

Table 2: Map 2 (with aisle) Average Results after 10 Iterations

Cost A Star Heuristic Multiplier, ε_{cost}	Path A Star Heuristic Multiplier, ε_{path}	Task Planner Time (ms)	Task Planner Nodes Opened (#)	Path Planner Time (ms)	Total Planner Time (ms)
0	0	57.0696	13.3	0.2252	57.3248
0	1	58.5478	13.3	0.2658	58.8136
1	0	49.0824	13.3	0.2624	49.3448
1	1	43.0949	13.3	0.1018	43.1967
2	2	47.7112	12.8	0.1619	47.8731
10	2	45.8787	12.7	0.1258	46.0045
10	10	45.1252	12.3	0.0915	45.2167

Discussion: As we expect, the general trend is that an increase in weighting speeds up the search. Increasing weighting in the path A* search directly speeds up the path planner duration. Interestingly, increasing the weight for the cost path planner (that is the low level path planner used to calculate edgecosts for the high level planner) causes a slight decrease in the number of nodes expanded in the task planner, as well as an improvement in overall time taken by the task planner. We would expect the improvement in time, as weighting A* sacrifices optimality for focus on reaching the goal, however, the slight decrease in task planning nodes is a bit more complex. Since the increase in ε_{cost} makes a given edgecost in the task planner less optimal, (any where from 1 to ε_{cost} times less optimal) it creates a greater stratification of possibilities for the costs between any two states. Since these path costs are calculated as part of the heuristic in the task planner, the task planner becomes slightly weighted as well (somewhere between 1 and ε_{cost}, basically the average of the suboptimalities of each cost calculation). This explains why there is no improvement from $\varepsilon_{cost}=0$ to $\varepsilon_{cost}=1$ (going from uninformed A* to standard A* improves time without sacrificing optimality) but from $\varepsilon_{cost}=1$ to $\varepsilon_{cost}>1$ there is improvement.

Interestingly, we are unsure why the task planning timing did not follow the trend in map 1. Part of this may be as a result of memory issues slowing down the machine. We also saw an improvement from map to map. This is unsurprising because the central aisle in map 2 offers more optimal routes in pretty much every path, that are easier for weighted A* to find. Regardless we are able to plan paths for a highly complex 1000x1000 map in under a single millisecond, and accomplish task planning (albeit in an intentionally small search space) in under 60 ms.

Additional Details

Heuristic Proof

Cost is equal to $\frac{distance\ travelled}{supply\ refilled}$. Given that supply refilled can only be equal to or less than previous supply refilled and distance travelled can only increase from waypoint to waypoint. $h(s) \leq c^*(s, s_{goal})$ is true, and therefore the heuristic is admissible.

Variables:

- Cost from Robot Position to Warehouse Bin, L_1
- Cost from Warehouse Bin to Machine Bin, L_2
- Amount of Supply at Warehouse bin, S_1
- Amount of Supply at Machine bin, S_2

Given the variables above, the heuristic functions can be described as below:
$H_{RW} = \frac{L_1}{S_1}$, Heuristic from Robot to Warehouse, and
$H_{WM} = \frac{L_1 + L_2}{MIN(1 - S_2, S_1)}$, Heuristic from Warehouse to Machine

These heuristics take into account the possible benefit (or reward) of picking up from warehouse bins and the true benefit (or reward) of dropping off at specific machines (given the previous reward).

One of the properties of the heuristic functions is:
$h(s) = max(h_1(s), h_2(s))$ − *consistent* if h_1 and h_2 is consistent.

In this case, $h_1 = \frac{SUM(L)}{S}$ and $h_2 = \frac{L}{1 - S}$. With these heuristics, S will not change, but L (or cost/total distance travelled) will increase every time step. Therefore,
$h(s) \leq c(succ, (s)) + h(succ(s))$, and h_1 and h_2 are consistent.