F29AI - Programming with A* Search

Introduction

Our task was to design a program that uses A* search to find a solution to the problem.

State Encoding

States are represented as a node object in our program. The node stores the number of small packages in the truck (S), the number or large packages in the truck (L), the number or medium packages in warehouse A(M_a) and the number of packages in warehouse B (M_b). Only the packages which are not in the correct place are represented, as those which are in the correct place will never be moved and will not factor into future decisions, i.e the ronbot is only concerned with packages in the wrong place.

The total path cost (from the root node to the current node), the heuristic value and the ronbot location is also stored. A pointer to a node's parent node is also stored, as this is essential for tracing a path once the goal state is found. Pointers to children nodes are not stored as they would go unused.

Transition Function

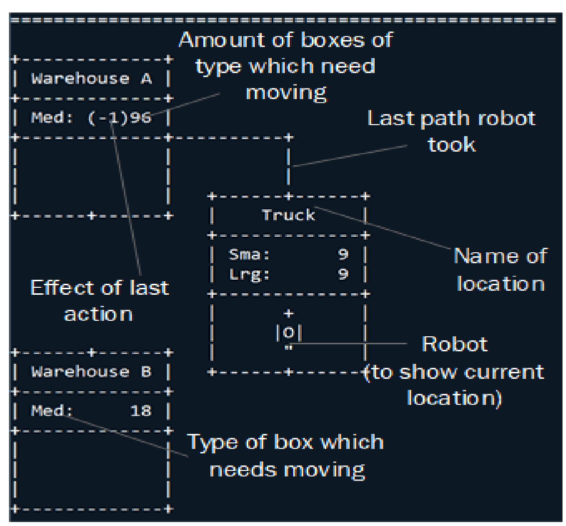
Our transition function generates all possible states that can be reached from the current state within one action. An action is the ronbot moving from one location to a connected location, either with or without moving a package. The ronbot cannot move a package to a location that that package cannot go, and the ronbot cannot move a package from where that package belongs.

The set of newly generated states are stored in a priority queue of other terminal nodes (i.e the nodes which have been generated, but have not themselves generated children yet), sorted by the value for path cost + heuristic cost. This is also known as the "open set"

Output

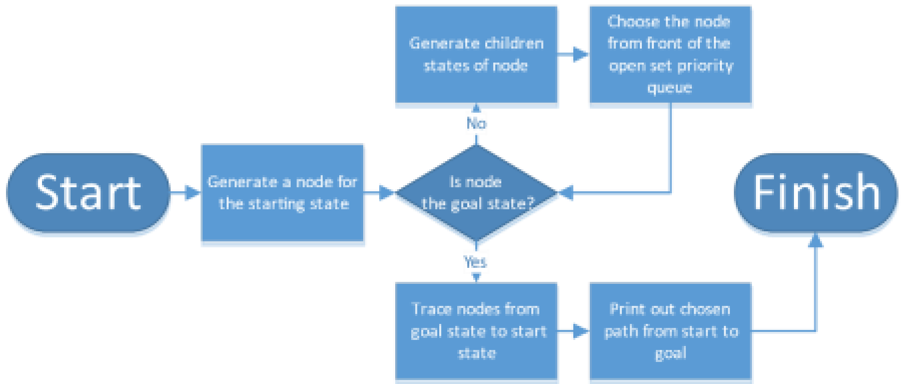
Once a goal state is found, we know that it is the best match as we have used an A* search algorithm. We put this goal state on a stack, follow it's pointer to its parent and put that on the stack, put its parent on the stack etc until we reach the start state (whose parent pointer is null). We can then pop each state off the stack and display it to show the solution.

We use an ascii diagram to visualize the output. Data from the previous state is used to show the effects that the last action had, such as where the ronbot moved from and what type of package had been moved.



Traversal

The front of the priority queue is removed and checked to see if it is at the goal state ($S = L = M_a = M_b = 0$). If the node is the goal state, then a plan has been found and is outputted. If it is not, then the children of the node are generated and the algorithm chooses the next node from the priority queue.



HEURISTICS

Manhattan

It is possible to map the state of the packages in a 3D space, with the x, y and z axis representing the amount of small, medium and large packages that are still to be moved. An action is a movement in one of three directions, x-1, y-1, z-1. The goal state is x = y = z =0, so our heuristic is the distance (or number of actions) from the current state to the goal. This has the potential to under-estimate, as it cannot take into account the situations where the ronbot must move without a package.

Refined

The refined heuristic proves significantly more efficient than the more rudimentary 'Manhattan' implementation due to the increased accuracy. In fact, empirical testing has shown the heuristic to never overestimate, and underestimate the true cost to goal by no more than 2 in the extreme. This accuracy comes from breaking the problem down into two distinct phases. Phase 1 always occurs the same regardless of state. In phase 1 we calculate the cost of moving as many parcels as possible whilst never moving hands-free; this is moving all applicable parcels between a given warehouse and truck until one of the two locations has no more - performed for each warehouse. This gives us `2(min(Ma, S))+2(min(Mb, L))`. Phase 2 can occur in two separate ways: the first simpler case is when the two remaining sets of parcels are S and L, or Ma and Mb. In this case the remaining parcels all have a cost of two - to move a parcel into location, and then to return for another. Since the final parcel will have no return cost, this gives us `Phase1+2(a+b)-1` as total cost to goal. Here `a = max(Ma-S, S-Ma)` and `B = max(Mb-L, L-Mb)`, which each represent one group of remaining parcels wherever they may be. The second case is a modification to the first, required when there remain parcels in one warehouse and the truck. This is because one set of the parcels have no return costs. Consider after phase 1, 5 parcels are in WarehouseA and 5 in the truck (which must be for WarehouseB after phase 1). Moving parcel from WarehouseA to truck incurs only one cost, and only the parcel from the truck has a return cost (returning either to WarehouseA of the Truck). Mathematically this is Phase1+2(a+b)-1 - min(a, b), with min(a, b) removing the extra return costs calculated. The copious use of min and max functions allow the heuristic to elegantly deal with differences between the number of parcels and abstract over their location. Naturally the exact cost to goal is also dependent on the Ronbot's location, which will not always perfectly map to the mathematical model. This is where the occasional underestimation occurs, with an additional hands-free move (or two).