29.10.2015
Overney Tristan
Pestalozzi Lukas

Intelligent Agents: Deliberative Agent

Task

Our task was to implement a deliberative agent able to find a optimal solution for the pickup delivery problem given a map and some tasks using a state search with the A* algorithm.

Implementation

Code

Structure

We implemented the standard 'textbook' A* algorithm and subclassed it to fit our needs.
In particular the following methods are abstract:

- *insertOpen(node)*: Determines how a new search node is inserted into the open list.
- *heuristic(node)*: Returns some heuristic for a given search node
- *children(node)*: Returns a list of child nodes for a given search node.
- *cost(nodeFrom, nodeTo)*: The cost to move from one search node to another.
- *isGoal(node)*: Determines if a search node is a goal node.

Overriding this methods allows us to achieve the desired behaviour.
Note that the value (f-value) of a search node is the sum of its h and g value.
The h value is the value of the heuristic for this node and the g value is the cost it takes to reach this node (from the initial node).

Pick up and delivery problem

The problem can be modeled by defining the methods in the following way:

- children(node): Adds all states that can be reached by a legal action (pickup, deliver or move). This can be made smarter (see "Interesting Points")
- cost(from, to): The cost for the vehicle to move between the cities.
- isGoal(node): If all packages are delivered then it is a goal node.

With this and the state representation (described later) we can also overwrite the missing methods to do BFS or BestFS on the resulting tree.

Best first Search

Best first search is A* with a sorted open-list and some heuristic.
Since we want to minimize the total cost it takes to reach a goal state the open list is sorted in ascending order (based on the f-value).

BFS

The BFS algorithm is just A* with a FIFO open-list and heuristic 0. Therefore we only override the 'insertOpen' and 'heuristic' methods in A* to achieve the BFS behaviour.
There is a little overhead compared to a 'clean' BFS implementation, but it is negligible.

29.10.2015
Overney Tristan
Pestalozzi Lukas

Interesting Points

Open List

For the open-list in the A* algorithm we decided to use a (double) linked-list for the BFS and a java priority queue for the bestFS. The reasons are following: BFS only removes nodes from the beginning and adds at the end of the open-list. Since this two operations take constant time with a linked-list this is optimal. BestFS removes nodes from the beginning and inserts nodes at the 'correct' place in the open-list. The java priority queue is implemented as min-heap, which offers constant time removal of the minimum node and O(log n) insertion (at the correct place) of a new node. Which fits pretty good for our needs.

State representation

We represent a state with 3 variables:

- the position of the vehicle
- the weight the vehicle can still carry
- an array with all the packages status (position and delivery city).

This allows us to generate successor states easily and we have all information need.
We can thus also use each state as a search node for the A* algorithm. This is also the reason why we use '(search)node' and 'state' interchangeably in this text.

Heuristic

After many iterations ranging from simple heuristic such as the number of package left to deliver to complicated ones such as the sum of all edges that have to be traveled through we settled for following one:
(The biggest distance from a not yet picked up task to its destination) + (The biggest distance from the vehicle to a picked up task's destination) - function(number delivered packages). Then take the max between this number and 0.
We use maxWaiting and maxCarried for the two first terms.
The intuition behind this heuristic is that the vehicle does have to travel at least the maxCarried distance to reach a goal state. The maxWaiting term is there to make a better prediction of the distance 'still waiting'. The function of 'number delivered packages' is a bonus guiding the heuristic to prefer states where more packages are delivered.
We found that a reasonable good value for the function is $4^{nbrDelivered}$.

Optimality

A heuristic in A* leads to an optimal solution if it underestimates the real cost to reach a goal state. This can be verified in every textbook describing A*.
The problem with our heuristic is that there can exist situations where the heuristic overestimates the real cost. But since, to reach a goal state, the vehicle has to pick up all tasks, the maxWaiting will be 0 and thus it will underestimate at that point (It may lead to some unnecessary state visiting, but we think that is ok). Furthermore the function helps to decrease the total heuristic. This are the reasons why our heuristic still gives us optimal solutions.

29.10.2015
Overney Tristan
Pestalozzi Lukas

Children states

There are 3 actions that can be taken in each state. Pickup(p), Deliver(p), Move(neigbor). We generate the successor (child) states according to following rules: If a package p can be delivered then this is the only successor state. If not, then all legal Pickups and Moves generate each one child state.
Then The idea is that if a delivery is possible then we definitively want to do it since we can avoid a lot of time by not computing (and inserting) transitions that will never be chosen. With the same idea in mind, If we have a pickup possible, we, in most situations, will want to pickup the package. As such we cas also avoid computing all moves if we do so there is some corner cases which may yield a suboptimal route; as such we implemented this shortcut in our A*-fast (agent deliberative-own-fast) only!

Results

Performance comparison between A*, A*-fast and BFS

In the following table we can see (with the default seed) all the informations relative to the plan computation depending on the agent behavior and the number of packages. The time is the time taken to compute the plan. The distance is the length of the used plan and the number of state is the total number of state visited to create the optimal plan.

	2 packages	6 packages	12 packages	15 packages	19 packages
A*	890 km 48 states 0.017s	1’380 km 1264 states 0.168s	1’820 km 1’105’901 states 48.40s	over a minute	over a minute
A*-fast	890 km 42 states 0.018s	1’380 km 256 states 0.048s	1’820 km 21’122 states 0.493s	1’820 km 53’761 states 1.554s	1’900 km 1’133’814 states 39.06s
BFS	990 km 82 states 0.016s	1’380 km 7982 states 0.216s	over a minute	over a minute	over a minute

Compare to 2 and 3 agents

As we can expect the gain for 2 or 3 agents is only that there are more vehicles to do the tasks and thus they take less time to complete them all.
The computational overhead for each agent to recalculate its plan is almost negligible since normally some tasks are already done and thus it has to take in acount fewer tasks. And since the search space grows exponentially for each added task the second computation will take exponentially less time. Of course the first plan has to be calculated 2 or 3 times (each agent oce).