CMPT 317

Assignment 1:

February 8, 2016

Colby Lemieux - cjl671

Wyatt Grant - wyg065

Problem Description:

K = Amount of packages to be delivered.

N = Amount of vehicles in the city that are able to be pick up and drop off packages.

P = Amount of Packages carried per car

M = Amount of locations(nodes) on the graph, will be square with R x R dimensions.

For each package K, there is a destination on the graph M chosen at random. The packages are initially placed randomly on the graph. There is some number of cars N that are in the city and able to pick up P packages, per car. The cars (if there is more than one) start at the same location (a garage) which is randomly chosen on the map at the start. The goal is for the car(s) to take all packages to their destination and then return to the garage as efficiently as possible (travelling the shortest distance). With this knowledge we are to experiment what will happen to the efficiency of the algorithm and complexity of the problem by increasing and decreasing the problem variables and using a variety of different search methods.

Solution Description:

(libraries used: math, copy, random as rand, from random import choice, matlibplot, networknx)

Our search variant for this problem was A* with forced choices based off of our state space and N, K, P and M. Because of our choice of A* our complexity ended up increasing a lot when we initially implemented it but, our results were far more efficient based on distance and it was able to easily handle a high and variable amount of N K P. As a result of using A* we decided to optimize based off of total distance, because of this we did not consider cases where cars could be moving simultaneously resulting in starvation depending on the number of cars at the expense of a total shorter distance travelled. Because of the choice of A* we will be able to

further improve our solution and it's complexity by improving the heuristic function used by A* or adding others to help find better paths in certain situations.

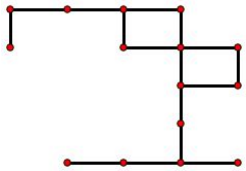
Implementation Description:

In our problem our state space was our current amount of cars, packages and the garage that were from our map of size M which are initially placed at random in our implementation, our cars begin at where the garage is initialized, the search space is all points in the map. Our successor function will check a given state and give us the our best decided move based off of our heuristics. For our heuristics we wanted to reduce the total cost travelled by the cars at end of the problem by comparing other possibilities in our problem space and picking an efficient solution without increasing the complexity too high. Firstly we want to compare our distances to packages and match up the most relatively close package and car and begin with that. From there we continue by either picking up another package (if our capacity allows for it) or drop off the nearest package, this is based off of distance travelled by the car. If dealing with multiple cars we must compare every car's best possible move and pick the best move from all of the cars, by increasing the amount of cars in the problem we increase the complexity by having to also check all other cars but, we potentially save time by overall travelling less between the two cars. Increasing the capacity of the cars actually seemed to give our cars more options. Being able to carry more packages meant that less backtracking had to be done if we have a high amount of packages and not many cars. If the packages happen to be clustered near one another some distance is saved by not having to go to a deliver a package if distance is saved by picking up another package. Once our cars had no packages left to deliver and our total packages on the map was zero we returned to the garage that the cars were initially place at.

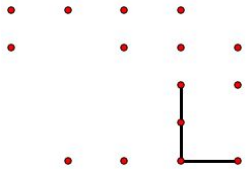
Results:

Our first Attempt, we used the variables K=1, N=1, P=1, M=5
We decided that we should first get the basics working, so in our initial result we simply got the car to find the package, go to the package, drop the package off, then go back to the garage. The Path to these nodes were found using A*. We also used a very small graph size to help simplify the problem. The time complexity of this is O(2^m).

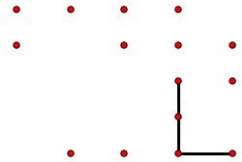
city



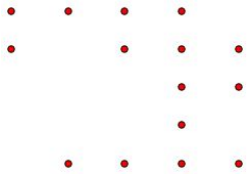
car leaves garage and gets package



car delivers package (package destination happens to be same location as garage here)



finally the car goes back to the garage (for this example his last destination was the garage so he doesn't have to go anywhere)

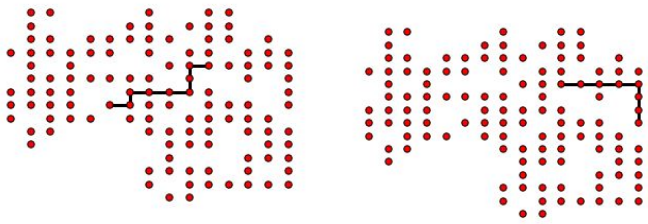


Our next result used the variables K=3, N=1, P=1, M=10
Here we increased the complexity of the problem by creating multiple packages with multiple destinations that have to be picked up one at a time. there was still only one car on the graph, that can hold only 1 package and the path to these nodes was still found using A*. We increased the graph size, due to now having more packages, and to make the problem less trivial. The time complexity that resulted from this was O(2^m).

Our next result used the variables K=3, N=1, P=3, M=10
Here we increased the complexity by adding the ability for the car to hold more than one package so that we could hopefully reduce some of the travel distance for our car, and make his route more efficient, the path to these nodes was still found using A*.The car will now be able to have up to 3 packages in his vehicle. The graph sized remained at 10. This change resulted in an increased time complexity of O(2^m).

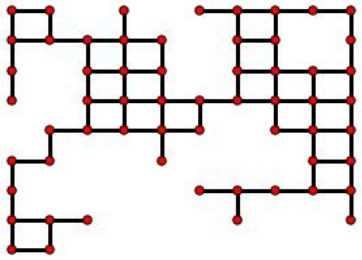
Our next result used the variables K=3, N=1, P=3, M=10
Here we used the exact same set of variables as above, but we changed the way the car decided which package to pick up and in what order. previously we were simply getting the packages in the order they were in the list given to the car. Now we decided to calculate the exact distance between nodes and choose the one closest to the car to pick up, then the next closest, and so forth. this worked better than going in order, but had potential to do very poorly depending on how the graph was laid out. certain gaps in the graph could lead the car to think a package was close, when in reality the car had much farther than it would have had it choose another package. The time complexity for this was O(p2^n).

Our next result used the variables K=3, N=1, P=3, M=10
Here we once again used the exact same set of variables. The above had potential to select unfavourable packages. for example



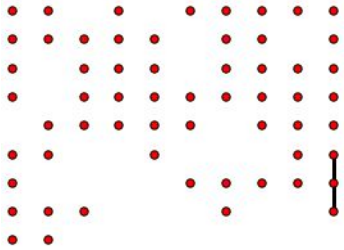
the graphs above represent to package locations. one is a distance of 9 units away from the car, the other is 8 units from the car. the algorithm based on exact differences selected the path of 9 units over the path of 8, even though it would have been the better choice. to combat this we decided to use A* to not only make our path to the package, but to also select which package should go first. this caused the time complexity to increase by a substantial amount, but we were able to get much more accurate results. the time complexity for this was O(kp(2^n)). below is a car picking up/dropping packages in an efficient manner. to keep track of cars we named them. the one in the below example is named “jimmy”.

city



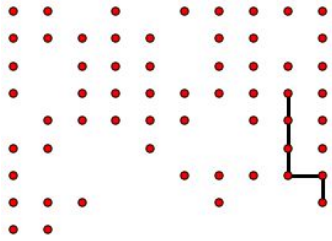
jimmy

Package Received



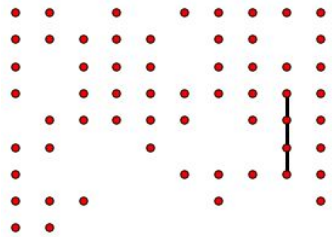
jimmy

Package Received



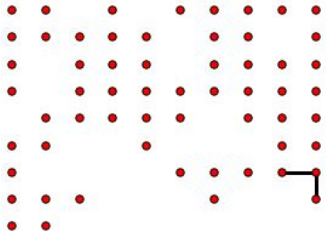
jimmy

Package Delivered



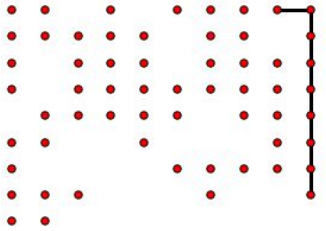
jimmy

Package Received



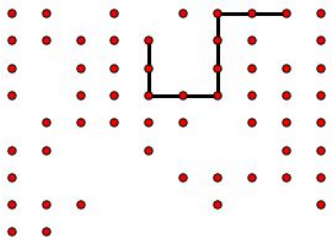
jimmy

Package Delivered

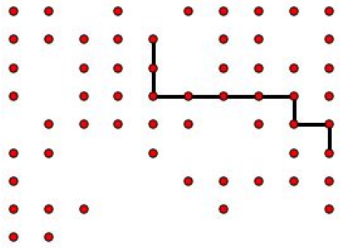


jimmy

Package Delivered



jimmy
returned to garage



Our next result used the variables K=3, N=3, P=3, M=10
In this result we finally add support for multiple cars. We used a similar approach to our previous result, because accuracy gained from more A* calls. here we checked the path of every car to every node and every package each cars to every destination and selected the most optimal path and best car to move. we keep checking for the most optimal path until there is no longer any packages to delivered or any packages are in the car. The time complexity that resulted from this was O(nkp(2^n))

Conclusions:
Based on the our results, we found that our solution offered an efficient way for cars to pick up and deliver packages while minimizing their total distance travelled as much as possible. In order to guarantee the best possible path of cars, we had to increase the time complexity by a large factor. We believe this trade off to be advantageous, because we were still able to compute the results in short amount of time for a graphs under 20x20. we could allow for large graphs by attempting to reduce A* usage, and rely on other things like the exact distance calculation we had used in a previous iteration of our solution. However, this would get worse results in terms of total distance travelled, which we were trying to optimize. We could reduce the total distance traveled by adding in a look ahead, so that it's not just finding the best package/delivery from where the car currently is, but where it ends after each time it moves. The farther we look ahead the worse our time complexity will get.

Implementation (in Python 3.5):
see the file wyg065cjl671.py included in the zip with this pdf.