Search

Author: Ruffin White

Import search functions

Setup environment

```
----- 6601 Grad AI: Seaching ATLANTA ------


NUMBER OF NODES:  12239
NUMBER OF EDGES:  13168
point A:          69121376
point B:          560602179
point C:          1418254999
```

Breadth First Search

```
...
```

1) Uniform Cost Search

Implement Uniform Cost Search. As a distance function use the straight line distance be- tween the adjacent nodes.

```
Start:  69121376
Path Found:  69121376  ->  560602179
 Path Cose:  1.77528663211

Shortest route:
Path :  69121376  ->  560602179
 Cost:  1.77528663211

Explored: 2142



Done!
```

Ruffin: It should be noted that different optimal routes can be observed from some unique points depending on the direction of approach. Perhaps this is due to rules of the road conditions that the Open Street Maps graph structure abides by, thus making it pertinent for real navigation systems.

2) Uniform Cost Search Three City Problem

Use Uniform cost search to solve three city problem. Describe how many searches were necessary. Shortly describe if Uniform Cost Search is complete and/or optimal.

```
Path Found:  69121376  ->  560602179
 Path Cose:  1.77528663211
Skipping  560602179
Path Found:  1418254999  ->  560602179
 Path Cose:  1.61560873679

Shortest route:
Path :  69121376  ->  560602179
 Cost:  1.77528663211
Path :  1418254999  ->  560602179
 Cost:  1.61560873679

Explored: 5718



Done!
```

Ruffin: The number of searches required for the three city problem using single directional uniform cost search is 2. To properly ensure that the absolute shortest path through all three cities is found requires that all three paths between all the cities are also found and compared in order to determine the shortest two paths that touch all three. The first search, say starting point 'A', is used to determine the path onward to 'B' or to 'C'. The first city to be found to be and explored node using uniform cost search, say 'B', is known to be on a shorter route from 'A' than the other point, 'C'. We can use the undiscovered point as a starting point for our second search that will attempt to find the path to 'B', as we can already deduct that we are in fact closer to point 'B', or else the uniform cost search starting from point 'A' would have explored 'C' first. This is an implementation of uniform, search, and is therefore complete in the regard that; if a solution exists, that the cost of each step remained positive, and that the graph is finite, it will find it. It is also optimal and that it will not stop nearly upon the first solution, but will continue to explore every available path untell found solution is known to bethe lowest-cost path among the frontier.

3) Bidirectional Search Three City Problem

Implement Bidirectional Search and solve three city problem. Describe how many searches were necessary, too. Shortly describe if Bidirectional Search is complete and/or optimal.

```
Path Found:  69121376 -> 560602179
 Path Cose:  1.72423120369
Path Found:  560602179 -> 1418254999
```

```
 Path Cose:  1.61560873679

Shortest route:
Path:  560602179 -> 1418254999
Cost:  1.61560873679
Path:  69121376 -> 560602179
Cost:  1.72423120369

Explored: 4892



Done!
```

Ruffin: by starting from two of the three points, and by having all nodes among the expanding frontier that ability to recall their starting origin, or seed, we can keep track of where the shortest intersection between the two expanding boundaries will occur as well as which of the two expanding boundaries, expanding at the same rate, encounters the third city first. It is also possible that the third city will be among the midpoint of the two expanding boundaries. This search method is also complete and optimal, but I however reduces the number of explored nodes by having the points expand towards each other simultaneously, essentially reducing the total radio area from the starting point required to be searched.

4) Tridirectional Search Three City Problem

Implement Tridirectional search. In Bidirectional search you start from the start and the goal state. In our case you start from all three cities. Implement all searches using Uniform Cost Search. Test your solution on the provided graph. Write a paragraph about your design choices. Also describe if Bidirectional Search is complete and/or optimal.

```
Path Found:  560602179  ->  1418254999
 Path Cose:  1.45792349594
Path Found:  69121376  ->  560602179
 Path Cose:  1.72423120369

Shortest route:
Path :  560602179 -> 1418254999
 Cost:  1.45792349594
Path :  69121376 -> 560602179
 Cost:  1.72423120369

Explored: 2685



Done!
```

Ruffin: My tri-directional search is implemented as such: all three points are initialized within the frontier from the beginning. These three frontier points will expand and explore, each keeping track of their origin and when they encounter a foreign frontier. Once a foreign frontiers encountered, this will be noted and checked to make sure that no foreign node has been explored before. Once the frontiers verified and that the foreign explored no is in fact the shortest distance to the foreign origin, the path is safed and the summation of the two distances from the midpoint with

respect to each origin is calculated. Once at least two paths that connect all three points have been verified, no more nodes expansion is required. This algorithm is complete and optimal, but also improves upon the time complexity, and space complexity due to the fewer number of nodes explored versus that of the previous two search algorithms.

5) Experiments

With your implementations compare the number of nodes touched when you perform three separate Uniform Cost Searches and the number of nodes expanded with Bidirectional Search and Tridirectional Search. Therefore, run a search for multiple starting conditions (pick three cities at random at least 100 times and for each run all your search methods). Count the number of nodes expanded for both. Write a paragraph about your results and explain your observations as well as your experimental setup. Also discuss the relation to the theoretical bounds using O notation. Report the number of nodes touched for Bidirectional and Tridirectional Search and report the difference to Uniform Cost Search. Graph the numbers on avg. saved.

```
Path Found:  551077003  ->  551073756
 Path Cost:  0.416106994452
Path Found:  551077429  ->  551073756
 Path Cost:  0.0984532733517
Skipping  551073756
Trial:  0
perimeter        :  0.607318413054
ucs3mono_explored:  489
Path Found:  551077429 -> 551073756
 Path Cost:  0.0984532733517
Path Found:  551077429 -> 551077003
 Path Cost:  0.454441435796
ucs3bi_explored  :  535
Path Found:  551077429  ->  551073756
 Path Cost:  0.0928909581357
Path Found:  551077003  ->  551073756
 Path Cost:  0.406182961094
ucs3tri_explored :  544
Path Found:  69189226  ->  553747986
 Path Cost:  0.519979538976
```

Ruffin:
Experimental Setup: Using a for loop, we continuously generating three random points among the graph. Due to computational limitations of my computer, I limit the distance of the surrounding perimeter of the three points to ensure that the points are not too far from each other such that the computational time is not insane. Another trick in order to limit computation time is to check that single directional uniform cost search does not require a expanding further searching for one point, further than roughly what would equate the entire perimeter of all three points. This is because even though three points might be relatively close to each other, because of the graph are using is not fully connected, it could result in that the points themselves never connect. And thus we would like to avoid having to expand the entire map to realize that three points less than a kilometer apart never connect. This however and effect works against being fully complete, but at the trade-off of improving my limited time complexity and space complexity requirements for this demonstration. As both bidirectional and tried correctional searches will only be compared against each other using points with existing path, this has little impact on the results other than that the number of expanded nodes are scaled-down as compared to what they could be if we let the random point generator have a field day.

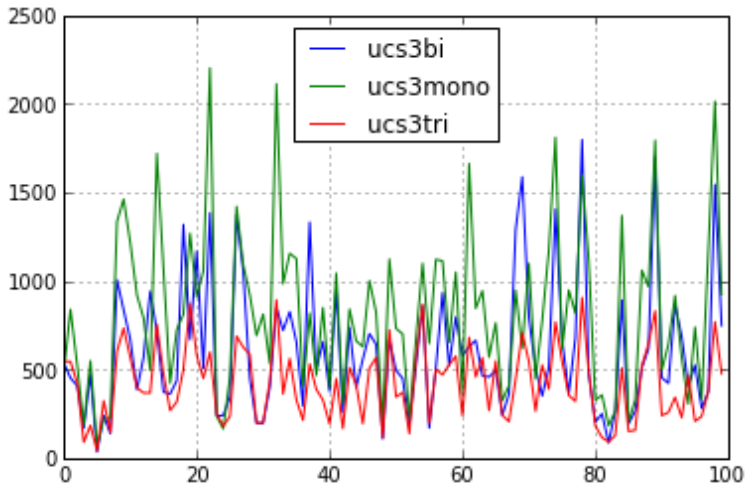
Results: As noted when describing each particular search algorithm, tri-directional search proves most efficient and resulting in the fewest number of nodes expanded on average: by taking advantage of reducing the area explored when searching for foreign frontiers and with each point contributing a smaller radius of expanded nodes.

Observations: As expected with a an even distribution of roadworks, such that an urban environment of a map of Atlanta would provide, the number of nodes necessary to expand is it that related to the total surrounding perimeter of the three points, as can be inferred from the scatter plot below.

Big O: As we know breadth first search can be described using big O notation as: $O(b^d)$. We can then see bidirectional search is say: $O(b^{d/2})$, so that the resulting total search is of a $O(b^{d/2} + b^{d/2})$. Uniform cost search however can be seen as having a big O of: $O(b^{1+C^*/\varepsilon})$ where C^* is the cost of the optimal solution and b is the branching factor and d was the depth of the shallowest solution. Bidirectional and Tri-directional uniform cost search can be seen as following a similar pattern such as $O(b^{\frac{1+C^*/\varepsilon}{2}} + b^{\frac{1+C^*/\varepsilon}{2}})$ and $O(b^{\frac{1+C^*/\varepsilon}{3}} + b^{\frac{1+C^*/\varepsilon}{3}} + b^{\frac{1+C^*/\varepsilon}{3}})$ respectively.

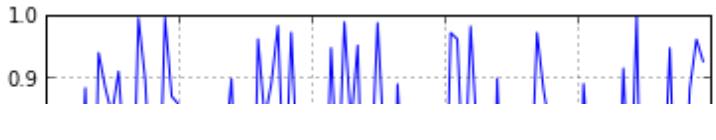
```
Out[11]: <class 'pandas.core.frame.DataFrame'>
         Int64Index: 100 entries, 0 to 99
         Data columns (total 5 columns):
         perimeter    100  non-null values
         points       100  non-null values
         ucs3bi       100  non-null values
         ucs3mono     100  non-null values
         ucs3tri      100  non-null values
         dtypes: float64(1), int64(3), object(1)
```

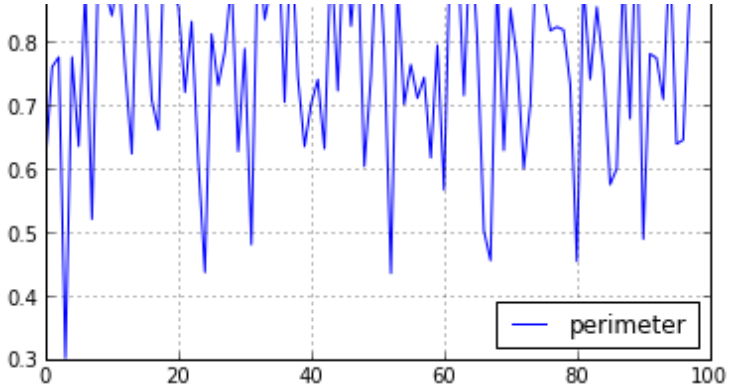
```
Out[100]: <matplotlib.legend.Legend at 0xf531250>
```



Comparing the number of nodes explored for all of the hundred random point combinations using each of the three search methods

```
Out[37]: <matplotlib.legend.Legend at 0xc478d10>
```





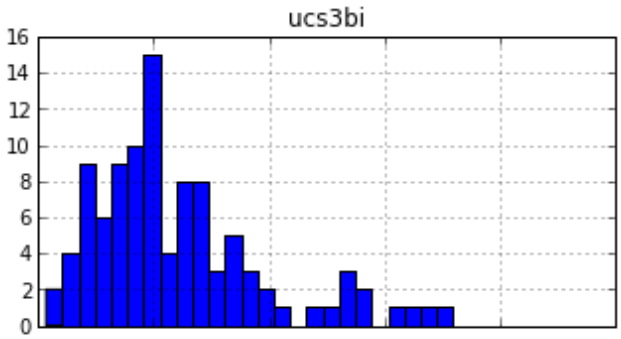
A complementary graph showing the distance of the perimeter with respect to the same 100 random sets of points

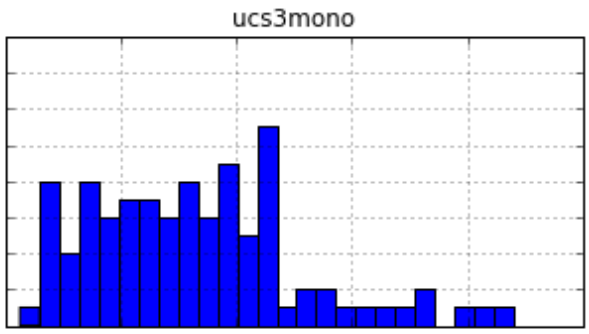
Out[38]:

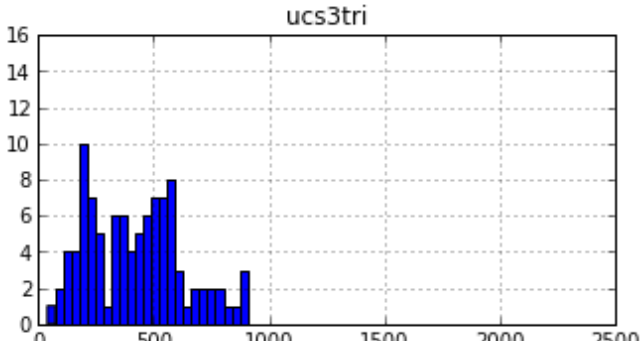
	ucs3bi	ucs3mono	ucs3tri
count	100.000000	100.000000	100.000000
mean	604.860000	814.840000	418.600000
std	374.650059	455.473729	207.731956
min	34.000000	64.000000	40.000000
25%	361.750000	464.500000	237.250000
50%	508.500000	800.500000	418.500000
75%	738.750000	1070.750000	550.500000
max	1796.000000	2201.000000	905.000000

Hard numbers from the experiment

```
Number of Nodes Expanded: X axis
Count: Y axis
```

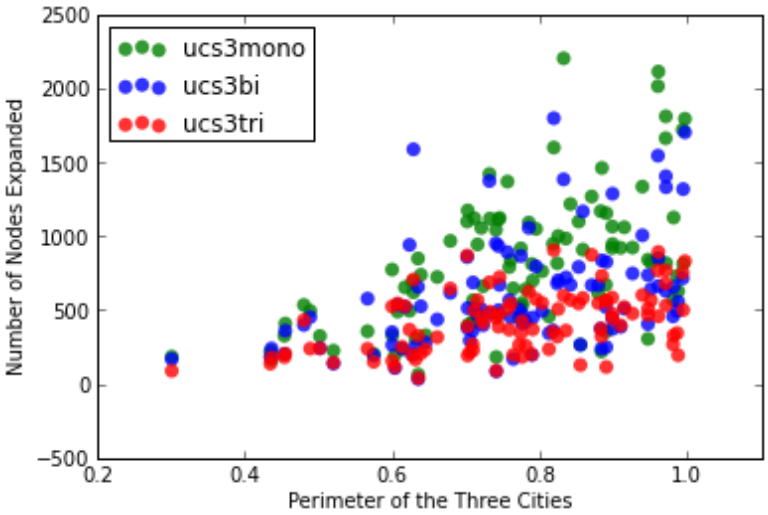




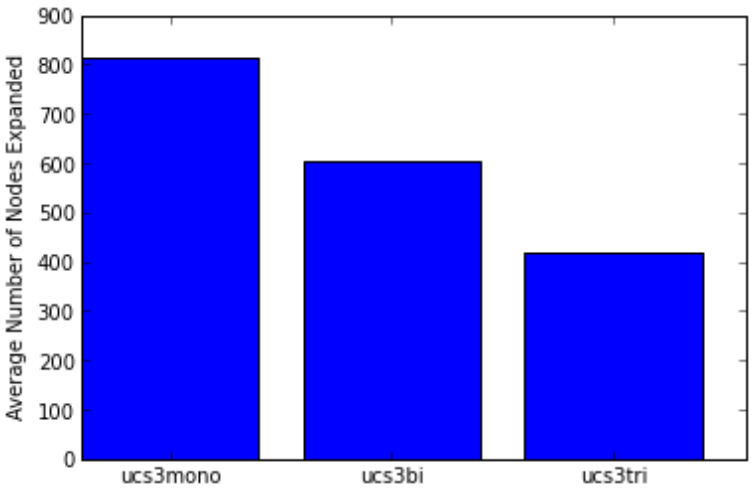


A histogram comparing the distribution of each algorithm with each other

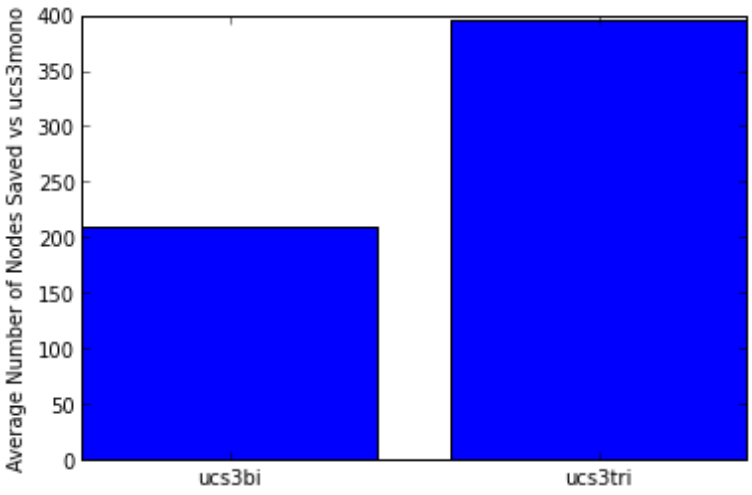
```
Out[95]: <matplotlib.legend.Legend at 0x11abc2d0>
```



The scatterplot, as mentioned above



Comparing the three algorithms average number of nodes expanded side-by-side



Showing the average number of nodes saved as compared to just a single directional uniform cost search