The B. M. S. Team!

The Shortest Path
to
Springfield!

Team Members

Mary Botros
Bilal Qureshi
Samuel Puchinskiy

Dr. Trana

Artificial Intelligence CS 335-1
Final Project
11/09/2019

Table of Contents

Abstract

This project allows a user to find the most efficient way to travel to Springfield, Illinois (Goal State) from different cities. Using A* search algorithm, we want to minimize distance/cost to travel. For example, we want to minimize the distance from Chicago to Springfield. Our project is trying to implement A* algorithm in a real-life problem such as going from select cities in Illinois to Springfield, Illinois. This project uses real numbers that represents mileage between cities. We are using 15 different cities as examples. We investigated many heuristics to compare which one will give us the shortest path. We used the Straight Line Distance from all other cities to Springfield, traffic, the population of each city, and area in square miles of each city as different heuristics.

Introduction

We chose to find the shortest path because it is so useful in our daily life. Every day, we use maps in almost everything in our lives. By understanding A* algorithm, we also understand that short roads are not always the best way, because other things such as traffic could make shorter paths, take a longer time. Therefore, we had to come up with logical heuristics, such as traffic, population, area of each city, and straight-line distance, that would be useful.

Creating the map wasn't an easy task because Illinois has 100 cities. Creating a map with one-hundred cities would be time consuming for this project, so we decided to choose Springfield as our goal state and use different cities from different directions that give us reasonable results. We ended up with 15 cities. We used Google Maps to draw our graph(map) using approximately the same location of each city on the map.

To be more efficient, we used four different heuristics, According to khanacademy.org: heuristic is : "a technique that guides an algorithm to find good choices. When an algorithm uses a heuristic, it no longer needs to exhaustively search every possible solution, so it can find approximate solutions more quickly". "Heuristic" is like a shortcut that leads us to efficiency, accuracy, and comprehensiveness.

We used different heuristics because we don't only want to find the shortest distance, we also want to take into consideration the travel time. To get an accurate result, we had to come up with accurate data. We thought about what the most realistic inputs for heuristics could be, so our outputs would be also realistic. We decided that traveling from place to place doesn't only depends on mileage, but also depends on traffic: The more traffic we have, the more time it takes to go from one place to another. Also, travel time depends on population. A high percentage of population means more cars on the road. Also, population means too many people walking, too many accidents and etc. In addition, it depends on the area of each city. Small cities would have smaller paths. Finally, an optimal heuristic would be Straight-Line Distance because “The shortest distance between two points is a straight line.” (Archimedes Quotes.)

Background & Related Work

We decided to use A* Search Algorithm because it is the most popular choice for finding the shortest path in less time between a start node to a goal node on a weighted graph. According to our textbook, the A* search algorithm, was developed by Peter Hart, Nils Nilsson and Bertram Raphae in 1968.

A * search algorithm is similar to Dijkstra's algorithm, we learned in our Introduction of Algorithms class. However, A* search algorithm combines Dijkstra's algorithm and Greedy Best-First Search. A* is an informed search. A* search has 2 main parts, g(n) and h(n). g(n) represents the cost from the start node to node n, and the heuristic, h(n) is the estimated cost of the cheapest path from node n to the goal. $f(n) = g(n) + h(n)$

We want to find the most efficient way to travel to Springfield using the A* Algorithm. We will use the nearby cities as our nodes and miles in distance as our edge values. The edges between all nodes represent the miles. we used Google Maps to get real miles between all the cities that we are using in our map.

We had to draw many maps for many reasons. The first map we made caused the code to loop and we are not handling loops our code. Also, arrow direction didn't make sense in some maps that we have, so we updated them. Also, some cities have a close number of miles that make results the same, and it wasn't clear which path is the shortest. Also, the number of cities mattered. The more cities we have, the more realistic is the results.

For traffic heuristic we used traffic based on commuter Traffic. Using population for heuristic was based on 2010 Census data. Using Area based on indexmundi.com data. For straight line, we used mapdevelopers.comtool and also we confirm straight line using google map.

Admissibility of the final Graph

Using this formula h(n) <= h*(n), we can see that using Traffic and Straight Line as heuristic make the Graph admissible, using Population and Area as heuristic make our graph Not Admissible.

City Name	Heuristic (population)	Heuristic (area)	Heuristic (traffic)	Heuristic (straight Line)	Distance to Goal
Bloomington	76,610	27.22	12	57.86	64
Carbondale	25,902	17.09	4	144.97	167
Champaign	81,055	22.43	7	77.35	86
Chester	8,599	5.81	5	130.62	164
Chicago	2,695,598	227.63	45	178.12	222
Effingham	12,328	9.86	3	75.14	88
Freeport	25,638	11.78	17	172.58	261
Kewanee	12,916	6.71	26	101.03	123
Mt. Vernon	15,277	13.07	18	109.82	143
Peoria	115,007	48.01	12	41.95	172
Quince	40,633	15.91	2	94.12	111
Rockfield	152,871	61.08	39	173.21	133
St. Louis	318,069	66	40	86.28	98
Vandalia	6,758	8.1	6	64.96	76

Consistency for final Graph

h(n) <= c(n, a, n`) + h(n`) ⇒ for every node

Using abbreviation for cities such as Bloomington: B, Carbondale: C, Champaign: CHA, Chicago: CHI, Chester: CHE, Kewanee: K, Mt. Vernon: MT, Peoria: P, Rockford: R, SpringField: SF, St. Louis: ST, Quincy:Q, Vandalia: V, Effingham: E

We can see that using Traffic and Straight line as heuristic, make the graph consistent.

n-n'	H(n) Straight Line	H(n') Straight Line	H(n) Traffic	H(n')Traffic	c(n-n')	H(n') + c(n-n') Traffic	H(n') + c(n-n') Straight Line
B-CHA	57.85	77.35	12	7	50	57	127.35
B-CHI	57.85	178.12	12	45	134	179	312.12
B-R	57.85	173.21	12	39	133	172	306.21
B-SF	57.85	0	12	0	64	64	64
B-P	57.85	61.9	12	12	38	50	99.9
C-ST	144.97	86.28	4	40	104	144	190.28
C-V	144.97	109.82	4	18	91	109	200.82
CHA-E	77.35	75.14	7	3	78	81	153.14
CHA-MT	77.35	109.82	7	18	147	165	256.82
CHA-SF	77.35	0	7	0	86	86	86
CHE-C	130.62	144.97	5	4	38	42	182.97
CHE-ST	130.62	86.28	5	40	66	106	152.28
CHI-CHA	178.12	77.35	45	7	136	143	213.35
E-SF	75.14	0	3	0	88	88	88
F-R	172.58	173.21	17	39	27	66	200.21
F-P	172.58	61.9	17	12	118	130	179.9

K-SF	101.03	0	26	0	123	123	123
Mt-C	109.82	144.97	18	4	57	61	201.97
MT-E	109.82	75.14	18	3	69	72	144.14
Mt-SF	109.82	0	18	0	143	143	143
P-K	41.95	101.03	12	26	49	75	150.03
Q-P	94.12	61.9	2	12	127	139	188.9
Q-SF	94.12	0	2	0	111	111	111
R-CHI	173.21	178.20	39.00	45.00	192.00	237	370.20
R-K	173.21	101.03	39	26	111	137	212.03
R-P	173.21	61.9	39	12	135	147	196.9
ST-MT	86.28	109.82	40	18	80	98	189.82
St-SF	86.28	0	40	0	98	98	98
ST-Q	86.28	94.12	40	2	137	139	231.12
V-SF	64.96	0	6	0	76	76	76

Approach

In our approach, we will give a brief description of each class and the class Unfiled Modeling Language

The Node Class creates the city and the city's directions. The node is the city and the edges are cities that are connected.

Node
node: {CityName} edge: dict{CityName, Miles}
init(nodes, egdes): void

The Graph class creates each graph and imports the heapq from the Python Library. In addition, the Graph class contains the A* Search Algorithm, which updates the path using the heuristic and distance values from each city. We borrowed a few methods from Homework 4 for the Graph Class.

Graph
graph: dict{} heuristics: dict{}
init(): void add_node(u, edges): void add_node_heurstics(n, h): void a_star_seach(start, goals): Node initialize_queue(nodes, queue, start): void update_queue(queue, tail, top_node): void

The Add_Nodes Class adds more cities to the graph.

Add_Nodes
graph: object
add_nodes(graph): void

The add_heustrics adds four heuristics to the graph. Each method is a different heuristic, Population, Traffic, Area, and Straight-Line Distance. In addition, the fifth method prints out all four heuristics to give a simple way to view each heuristic.

Add_Heuristics
graph: object
add_heuristics_1(graph): void add_heuristics_2(graph): void add_heuristics_3(graph): void add_heuristics_4(graph): void add_heuristics_5(graph): void

The Springfield Class prints the shortest path from the user's heuristic and start city input.

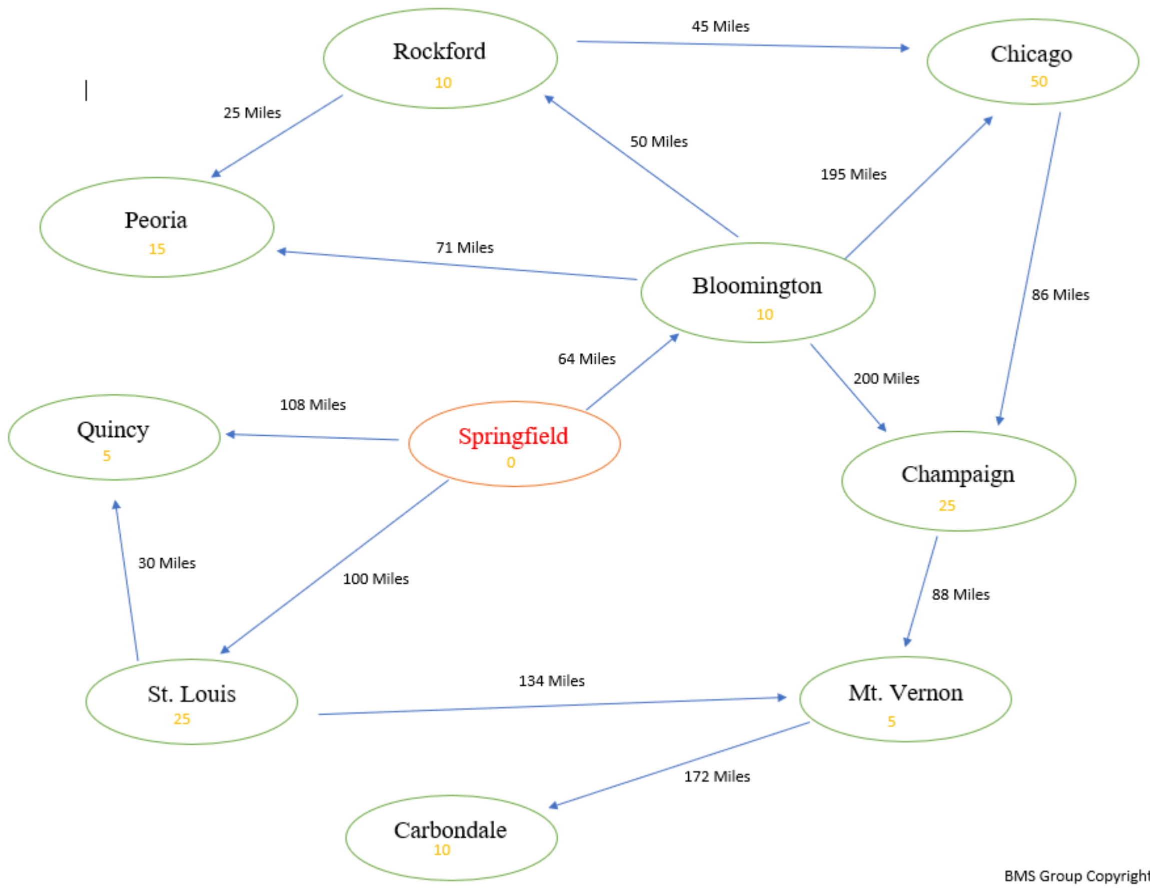
Springfield
graph: object
Springfield(answer): void

The main method contains a loop. The main method prompts the user for the start city and heuristic value. The main method error checks for invalid inputs.

main
count: boolean
main(): void

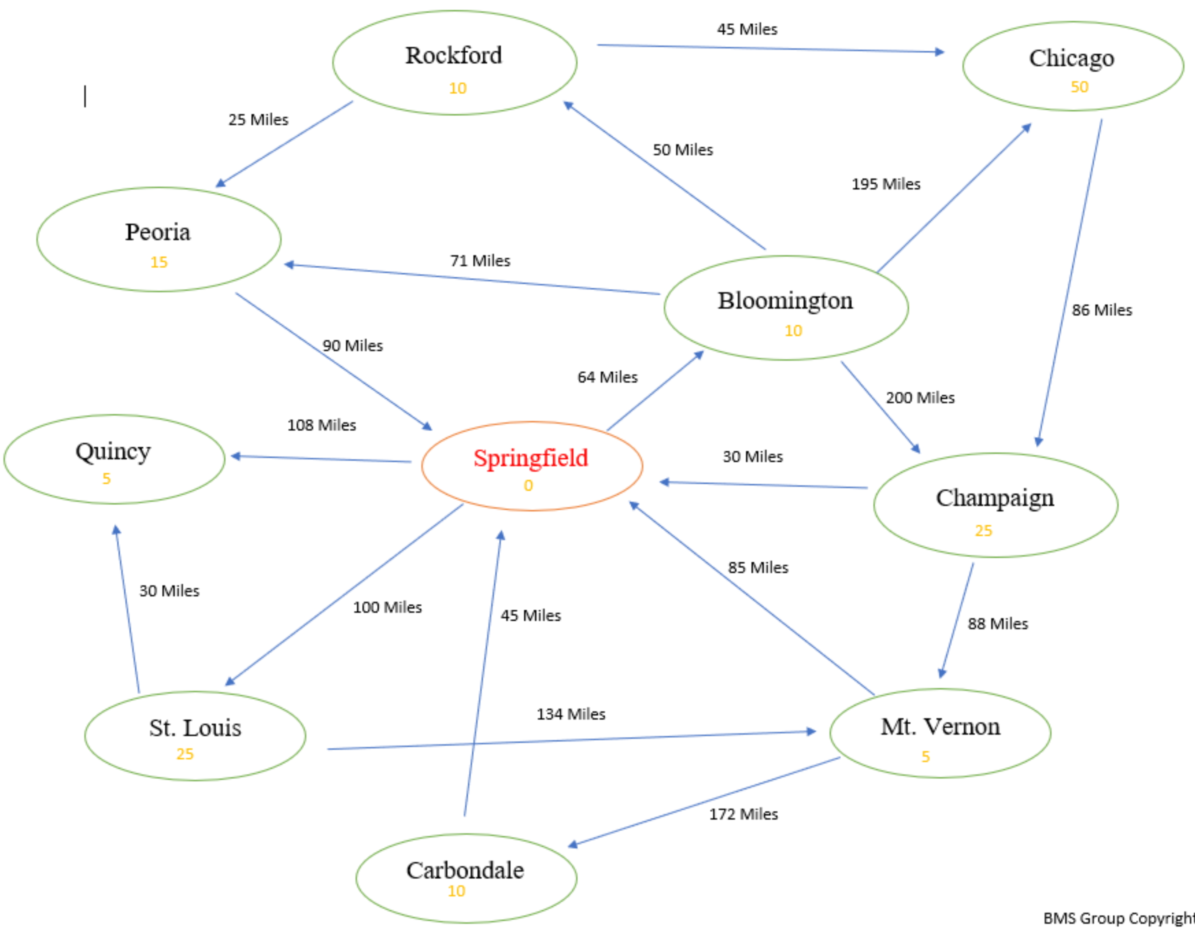
Experiment/Results

Step 1;
First map we created to make an experiment on it, had no path to the goal state, so we had to do a new graph.



Step 2:
We had to test our code results, so we created and tested our queue manually to compare the results. Finding the shortest Path from Chicago to Springfield,Give us:
Queue: Chicago → Champaign → Springfield: 116

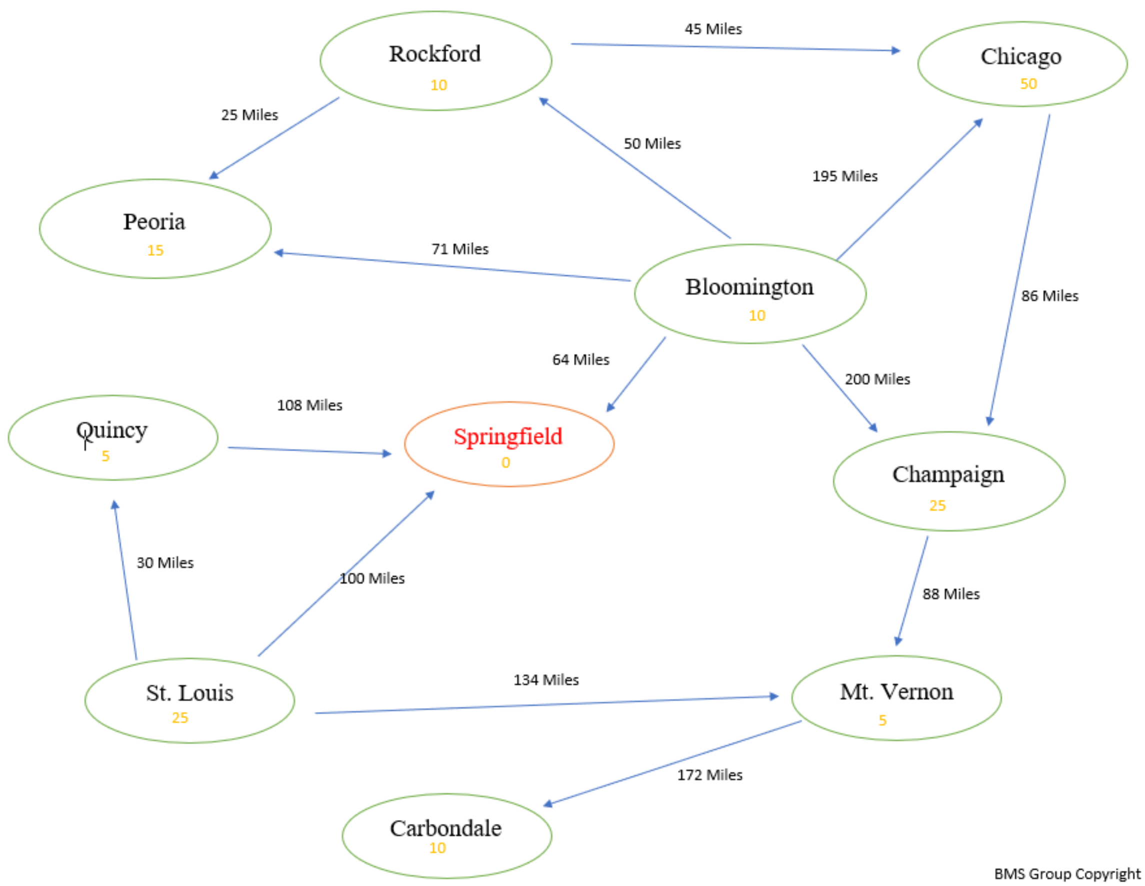
From State	To State	h(n)	g(n)	f(n)	Queue
Chicago	Champaign	25	86	111	CHI→ CHA: 111 CHI→ CHA → SpringField: 116 CHI→ CHA → Mt. Vernon: 179
Champaign	SpringField	0	116	116	CHI→ CHA → SpringField: 116 CHI→ CHA → Mt. Vernon: 179



Step 3:

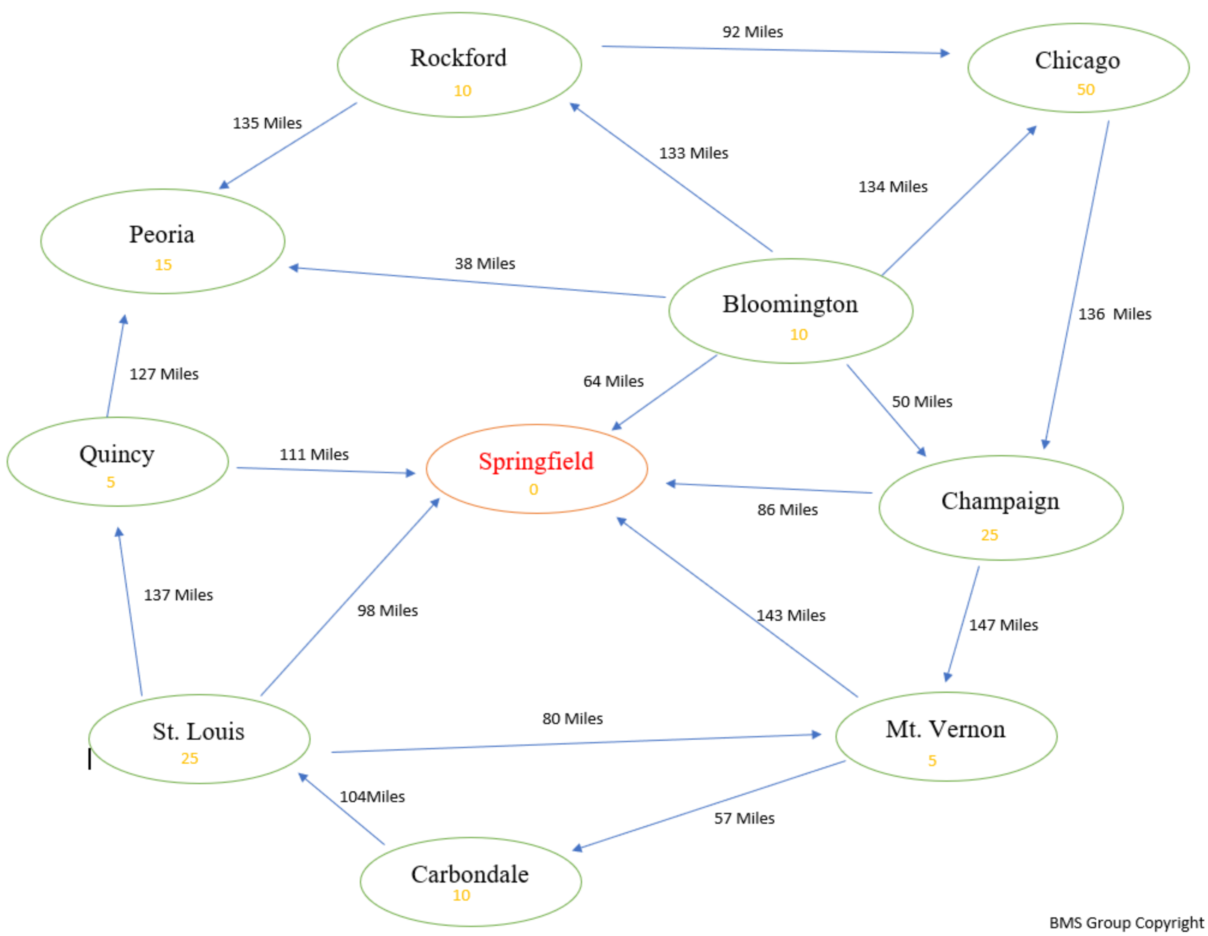
We removed loops by removing the directions that were leaving out of Springfield.
Using the third map going from Chicago to Springfield :

From State	To State	h(n)	g(n)	f(n)	Queue
Chicago	Champaign	25	86	111	CHI→ CHA: 111 CHI→ CHA → SpringField: 116 CHI→ CHA → Mt. Vernon: 179
Champaign	SpringField	0	116	116	CHI→ CHA → SpringField: 116 CHI→ CHA → Mt. Vernon: 179



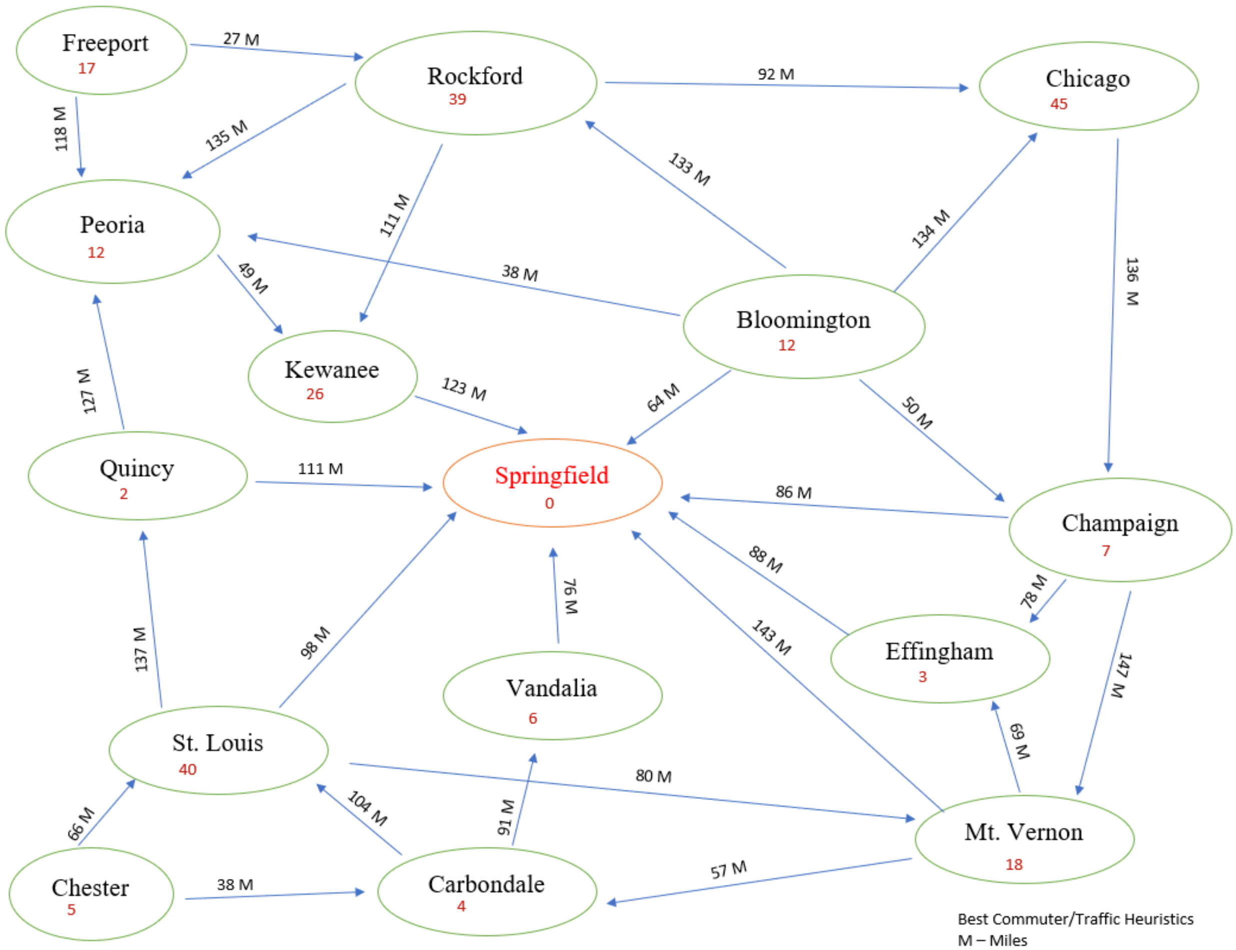
Step 4:
We added more directions coming to Springfield to get more different paths.
Using the Fourth map going from Chicago to SpringField :

From State	To State	h(n)	g(n)	f(n)	Queue
Chicago	Champaign	25	135	150	CHI→ CHA: 135 CHI→ CHA → SpringField: 221 CHI→ CHA → Mt. Vernon: 282
Champaign	SpringField	0	221	221	CHI→ CHA → SpringField: 221 CHI→ CHA → Mt. Vernon: 282



Step 5

We added more cities to get more realistic results. By adding more cities, we would get more paths from our code.



Step 6:
We added four different heuristic values to see how our results would differ. We want to consider how the Population, Area, Traffic, and the Straight-Line Distance would affect our path to find the shortest path to Springfield.

<table>
<tr><th colspan="2">Heuristic Values</th></tr>
<tr><th>Population of each city</th><th>Area in square miles of each city:</th></tr>
<tr><td>

Chicago: 2,695,598
Champaign: 81,055
Bloomington: 76,610
Rockford: 152,871
Peoria: 115,007
Quincy: 40,633
Carbondale: 25,902
Mt. Vernon: 15,277
St. Louis: 318,069
Freeport: 25,638
Kewanee: 12,916
Chester: 8,599
Vandalia: 6,758
Effingham:12,328

</td><td>

Chicago: 227.63
Carbondale: 17.09
St. Louis: 66
Champaign: 22.43
Bloomington: 27.22
Rockford: 61.08
Peoria: 48.01
Quincy: 15.91
Mt. Vernon: 13.07
Freeport: 11.78
Kewanee: 6.71
Chester: 5.81
Vandalia: 8.1
Effingham: 9.86

</td></tr>
<tr><th>Best commuter cities (aka lowest in traffic):</th><th>Straight line distance in miles from city to Springfield:</th></tr>
<tr><td>

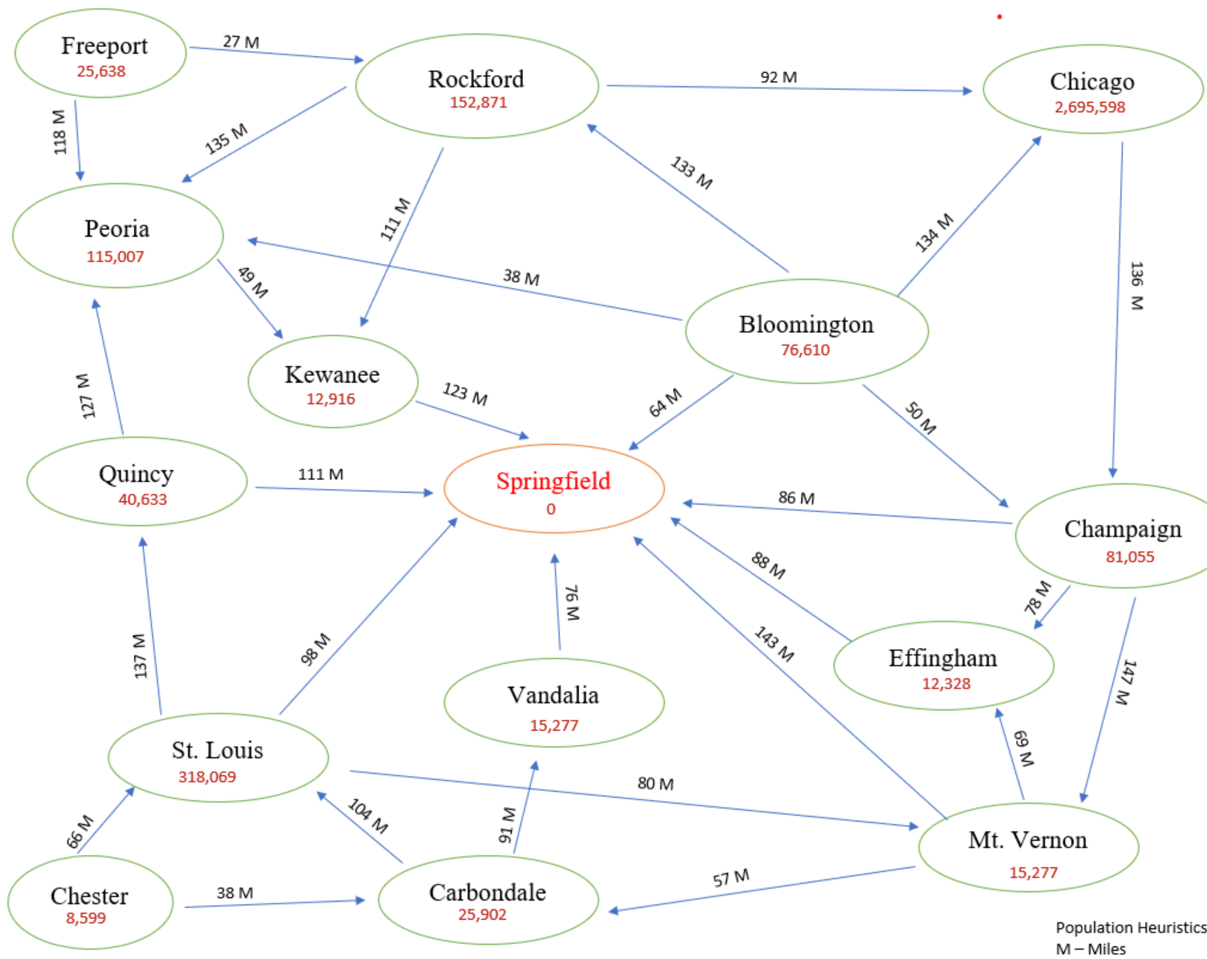
Chicago: 45
Carbondale: 4
St. Louis: 40
Champaign: 7
Bloomington: 12
Rockford: 39
Peoria: 12
Quincy: 2
Mt. Vernon: 18
Freeport: 17
Kewanee: 26
Chester: 5
Vandalia: 6
Effingham: 3

</td><td>

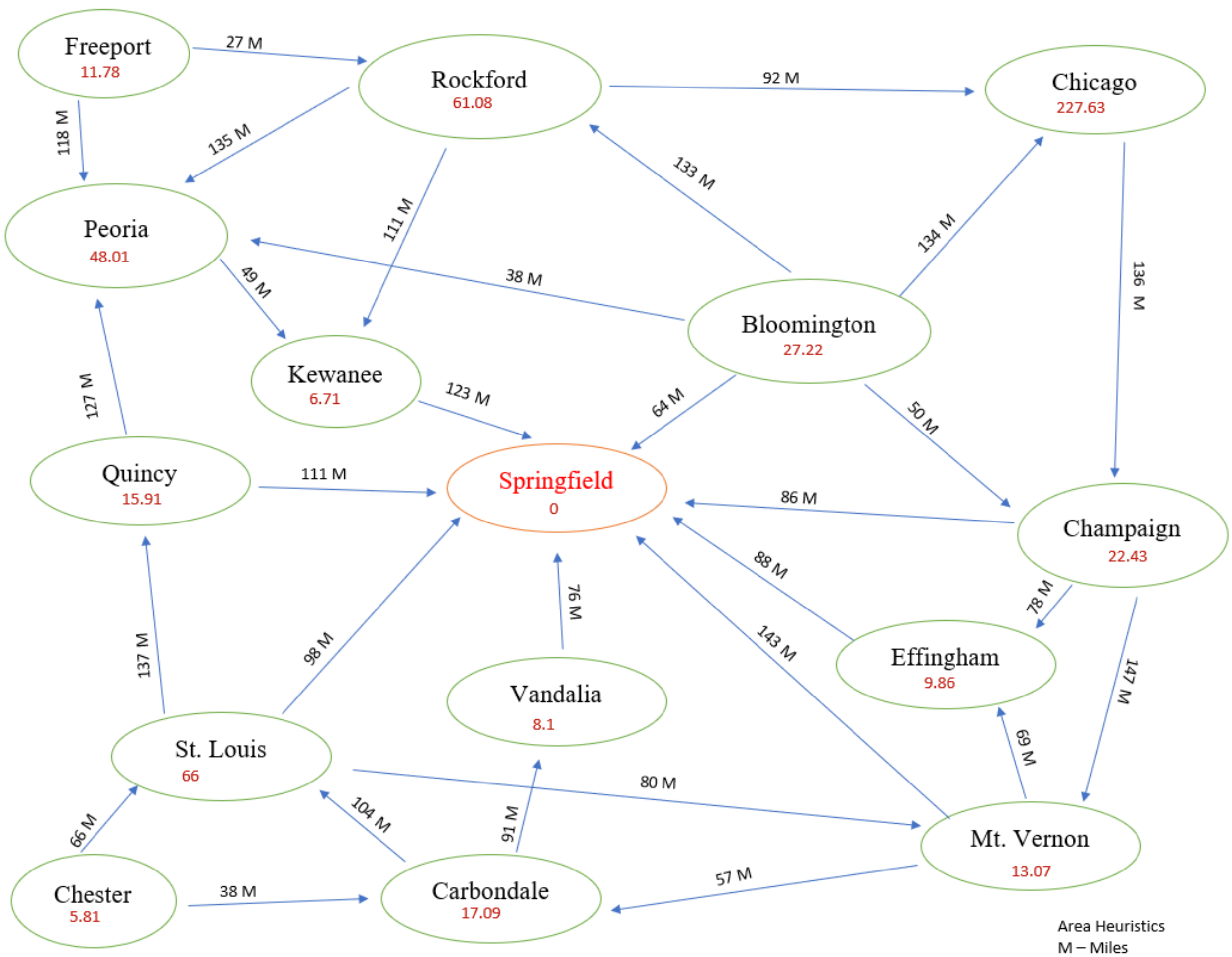
Chicago: 178.12
Carbondale: 144.97
St. Louis: 86.28
Champaign: 77.35
Bloomington: 57.86
Rockford: 173.21
Peoria: 41.95
Quincy: 94.12
Mt. Vernon: 109.82
Vandalia: 64.96
Kewanee: 101.03
Chester: 130.62
Effingham: 75.14
Freeport:172.58

</td></tr>
</table>

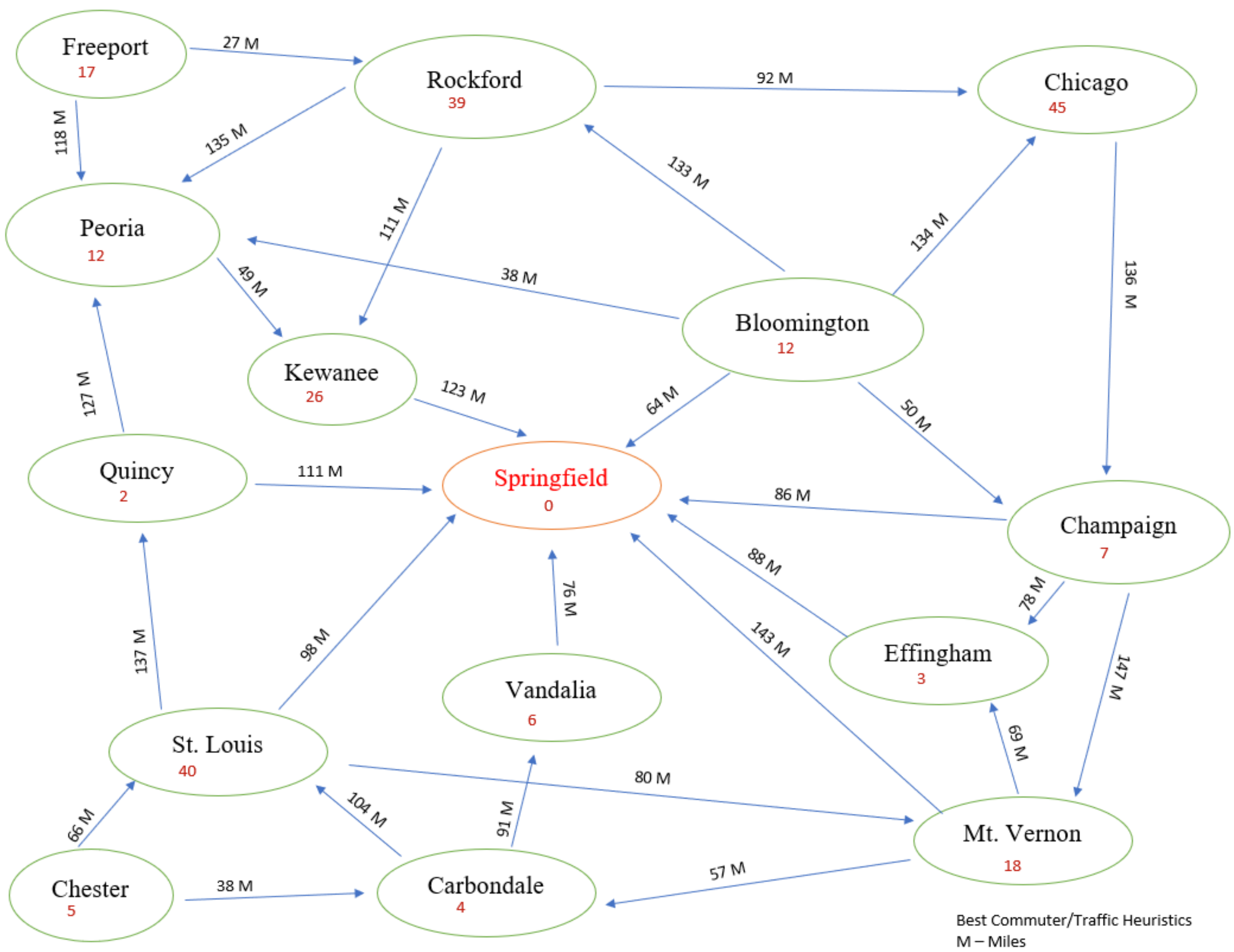
Population - Heuristic Graph



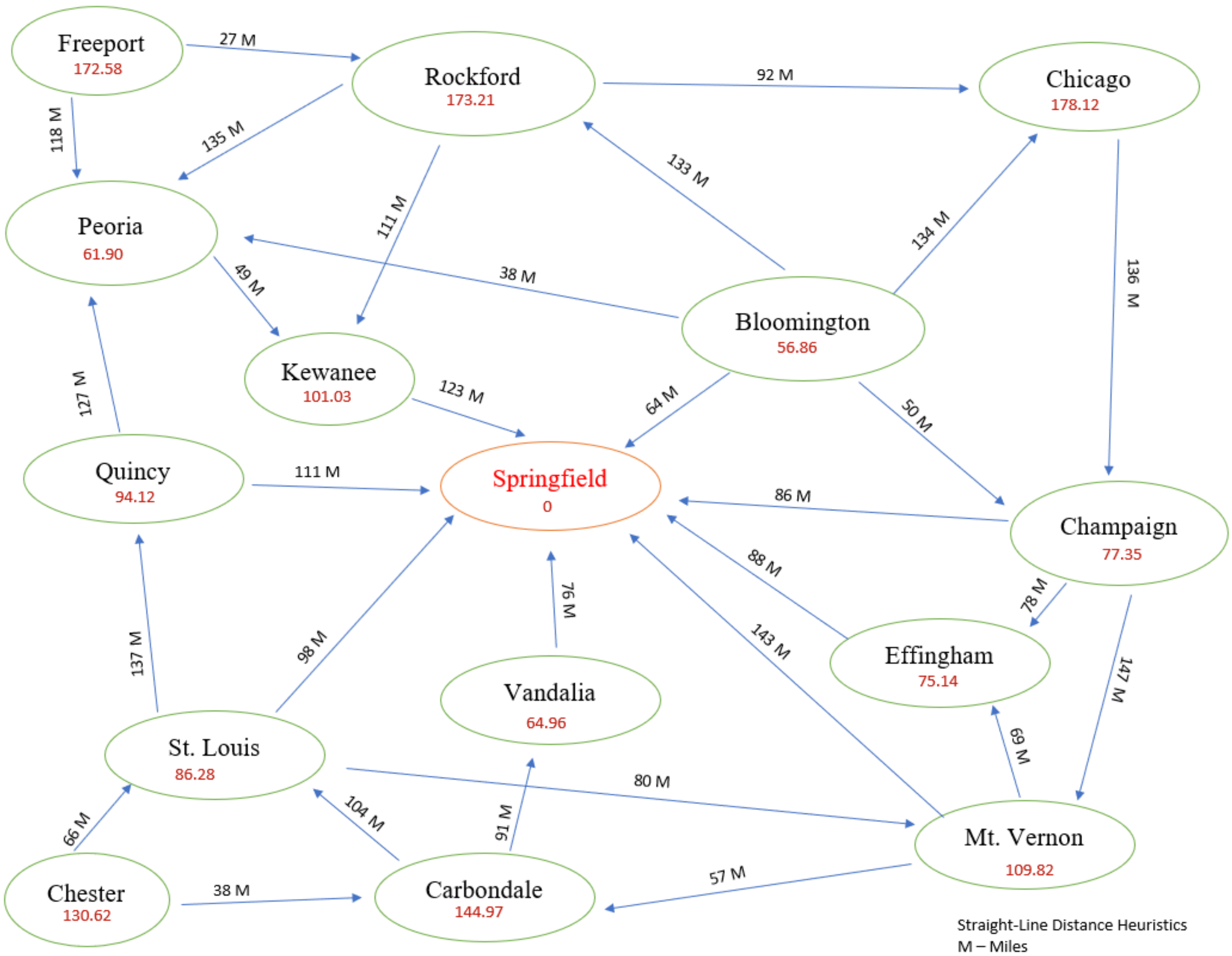
Area in Square Miles - Heuristic Graph



Best Commuter/Traffic- Heuristic Graph



Straight-Line Distance- Heuristic Graph



Conclusion

By implementing A* search algorithm in our python code, we discovered that with all the heuristics we used, they gave the same result with the exception of two starting cities using the population heuristic. When you start from Freeport to go to Springfield and use the population heuristic, the resulting path given from code is different from the other heuristics.The other heuristics start at Freeport then to Rockford, then to Kewanee, then to Springfield. But when you use population as the heuristic in the graph, then the path changes starting from Freeport then to Peoria, then to Kewanee and then to Springfield. The other city was Chester. If you use the population heuristic, and start from Chester, the path would take you to from Chester to Carbondale, then to Vandalia, and then to Springfield.

For both starting cities, Chester and Freeport, using the population heuristic the paths are longer in distance traveled. For Freeport, the distance traveled using the population heuristic is 290 miles, compared to 261 miles using the other heuristics For Chester, the distance traveled using the population heuristic is 205 miles, compared to 164 miles using the other heuristics.

By the end we have learned that heuristic affects our option in choosing the optimal way to go to any place. In our case, traffic would affect our decision in choosing the optimal way to go to Springfield.

The graph is optimal using Traffic and Straight line as heuristics because they are admissible and consistent.

A few things we could have done differently is we could add more goal states, and try different cities or we could have done our project using different states instead of cities. Also, we could have added a few more heuristics, to get different results.

Results

In our results, we only had different paths for Freeport and Chester. In both cases, the population affected the results. For the population heuristic, an additional city was added to the path to Springfield For instance, in our population heuristic, Chester went to Carbondale to Vandalia to Springfield, instead of going from Chester to St. Louis to Springfield. This was because St. Louis has a bigger population than Chester and Carbondale combined. .

#	City Name	Traffic Heuristic	Population Heuristic	Area of City Heuristic	Straight-Line Distance Heuristic	Results
1	Bloomington	Bloomington → Springfield (64M)	Bloomington → Springfield (64M)	Bloomington → Springfield (64M)	Bloomington → Springfield (64M)	N/A
2	Quincy	Quincy → Springfield (111M)	Quincy → Springfield (111M)	Quincy → Springfield (111M)	Quincy → Springfield (111M)	N/A
3	St. Louis	St. Louis → Springfield (98 M)	St. Louis → Springfield (98 M)	St. Louis → Springfield (98 M)	St. Louis → Springfield (98 M)	N/A
4	Champaign	Champaign → Springfield (86M)	Champaign → Springfield (86M)	Champaign → Springfield (86M)	Champaign → Springfield (86M)	N/A
5	Mt. Vernon	Mt. Vernon → Springfield (143M)	Mt. Vernon → Springfield (143M)	Mt. Vernon → Springfield (143M)	Mt. Vernon → Springfield (143M)	N/A
6	Carbondale	Carbondale → Vandalia → Springfield (167M)	Carbondale → Vandalia → Springfield (167M)	Carbondale → Vandalia → Springfield (167M)	Carbondale → Vandalia → Springfield (167M)	N/A

7	Peoria	Peoria → Kewanee → Springfield (172M)	Peoria → Kewanee → Springfield (172M)	Peoria → Kewanee → Springfield (172M)	Peoria → Kewanee → Springfield (172M)	N/A
8	Rockford	Rockford → Kewanee → Springfield (234M)	Rockford → Kewanee → Springfield (234M)	Rockford → Kewanee → Springfield (234M)	Rockford → Kewanee → Springfield (234M)	N/A
9	Chicago	Chicago → Champaign → Springfield (222M)	Chicago → Champaign → Springfield (222M)	Chicago → Champaign → Springfield (222M)	Chicago → Champaign → Springfield (222M)	N/A
10	Freeport	Freeport → Rockford → Kewanee → Springfield (261M)	Freeport → Peoria → Kewanee → Springfield (290M)	Freeport → Rockford → Kewanee → Springfield (261M)	Freeport → Rockford → Kewanee → Springfield (261M)	Population
11	Kewanee	Kewanee → Springfield (123M)	Kewanee → Springfield (123M)	Kewanee → Springfield (123M)	Kewanee → Springfield (123M)	N/A
12	Chester	Chester → St. Louis → Springfield (164M)	Chester → Carbondale → Vandalia → Springfield (205M)	Chester → St. Louis → Springfield (164M)	Chester → St. Louis → Springfield (164M)	Population

13	Vandalia	Vandalia → Springfield (76M)	Vandalia → Springfield (76M)	Vandalia → Springfield (76M)	Vandalia → Springfield (76M)	N/A
14	Effingham	Effingham → Springfield (88M)	Effingham → Springfield (88M)	Effingham → Springfield (88M)	Effingham → Springfield (88M)	N/A

References / Work Cited

Areavibes. "Chester, IL Demographics." *Chester, IL Population & Demographics*, www.areavibes.com/chester-il/demographics/.

Batoćanin, Vladimir. "Basic AI Concepts: A* Search Algorithm." *Stack Abuse*, Stack Abuse, 23 Mar. 2019, stackabuse.com/basic-ai-concepts-a-search-algorithm/.

Google Maps, Google, www.google.com/maps.

Illinois Land Area in Square Miles, 2010 by City, www.indexmundi.com/facts/united-states/quick-facts/illinois/land-area/cities#chart.

"Map of Illinois - About." *UIS*, www.uis.edu/about/visit/maps/illinoismap/.

Metterhausen, Fred. "Google Map Developers." *Distance Between 2 Addresses, Cities or Zipcodes*, www.mapdevelopers.com/distance_from_to.php.

Patel, Amit. "Introduction to A*." *Introduction to A**, theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.

"Populations of Illinois Cities." *Populations of Illinois (IL) Cities - Ranked by Population Size - Page 1*, www.togetherweteach.com/TWTIC/uscityinfo/13il/ilpopr/13ilpr.htm.

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: a Modern Approach*. 3rd ed., Pearson, 2016.

"Using Heuristics." *Khan Academy*, Khan Academy, www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/solving-hard-problems/a/using-heuristics.

Homework 4

```python
#Copied from Homework 4
def a_star_search(self, start, goals):
  if start in goals:
     return [start]

  nodes = (self.graph[start]).edges
  queue = []
  self.initialize_queue(nodes, queue, start)

  while queue:
     top_node = heapq.heappop(queue)
     tail = top_node[2][0]
     if tail in goals:
        return top_node[2][::-1]
     self.update_queue(queue, tail, top_node)
  return None

#Copied from Homework 4
def initialize_queue(self, nodes, queue, start):
  for node, cost in nodes.items():
     priority = self.heuristics[node] + cost
     node_info = (priority, cost, [node, start])
     queue.append(node_info)
  heapq.heapify(queue)

#Copied from Homework 4
def update_queue(self, queue, tail, top_node):
  nodes = (self.graph[tail]).edges
  for node, cost in nodes.items():
     new_cost = cost + top_node[1]
     priority = self.heuristics[node] + new_cost
     heapq.heappush(queue, (priority, new_cost, [node] + top_node[2]))
```