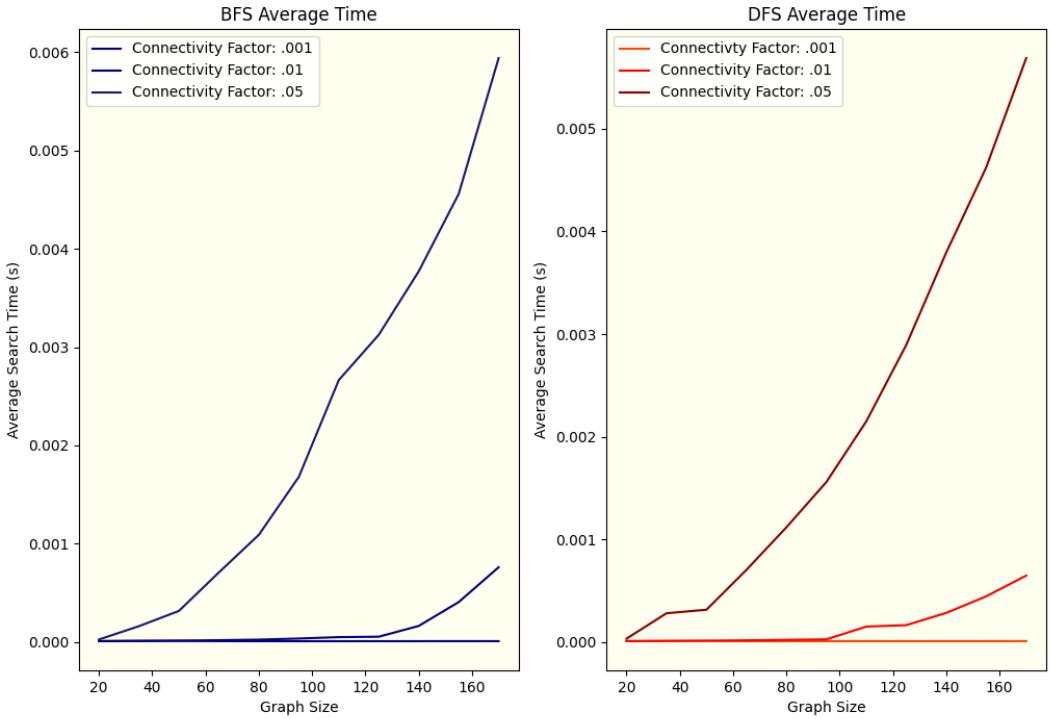
Writing Application Problem

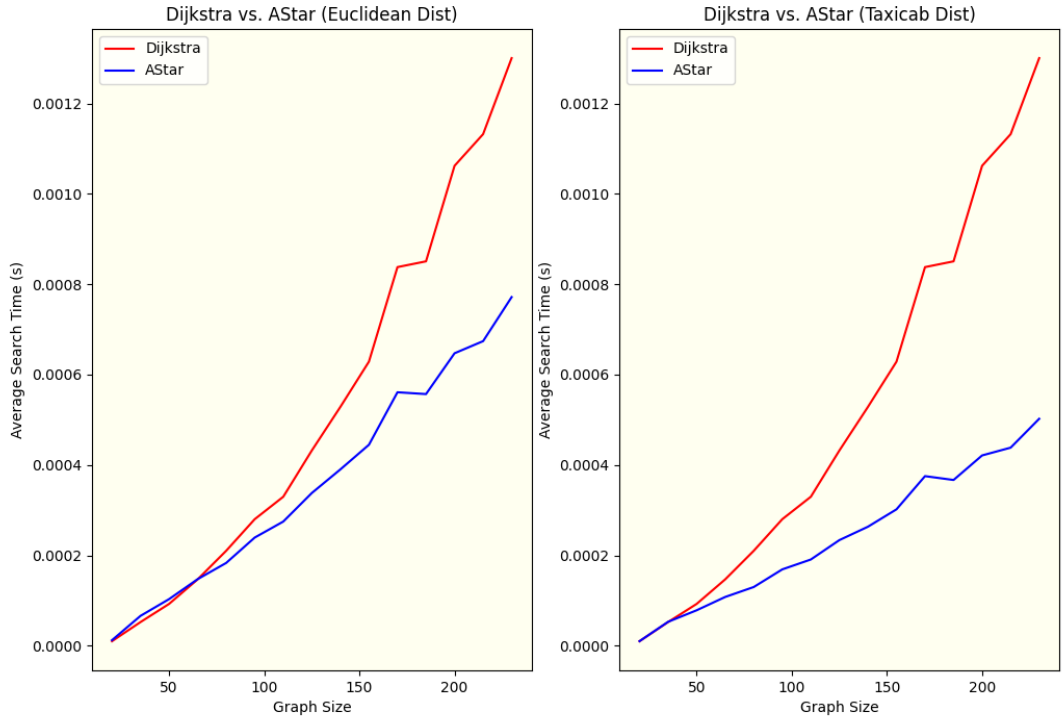
- **Complete the questions below - copy and paste them into a word doc - and include this pdf document as you are submitting your Project10 on Mimir.**
- Use project10 comparison.py file to answer questions
- Matplotlib installation resources: how-to-install-matplotlib-in-python and install python packages on PyCharm
- **Writing Questions**
  - Question 1 (2pts): Run the first test (test1) from comparison.py file and include the figure produced here.



  - Question 2 (2pts): The time complexity of both breadth-first and depth-first graph searches is O(|V | + |E|), where V is the set of vertices of the graph, and E is the set of edges. In the figure produced above, the ”connectivity factor” of a graph is the probability that given any two vertices, say v1 and v2, there exists an edge from v1 to v2. Note that in a fully-connected graph, where every vertex is connected to every other vertex, we have E=V^2. Why in the figure above does BFS/DFS performance appear linear for small connectivity factors, but not so far larger factors?

It appears linear for small connectivity factors because it does not have to run through as many vertices while if they the connectivity factor is larger, it has to run though more vertices, increasing the search time, making it appear less/non-linear.

- Question 3 (2pts): Run the second test (test2) and include the figure produced here.



- Question 4 (2pts): Notice that AStar search notably outperforms Dijkstras algorithm on the graphs provided. In one or two sentences, informally explain why. What does AStar take advantage of?

AStar outperforms Dijkstras algorithm because AStar takes advantage of using a heuristic to look for a better path and in most cases does not evaluate a vertex more than once, while Dijkstra explores all the possible ways, potentially visiting a vertex again.

- Question 5 (2pts): Suppose you were to call the AStar search as follows: What issues(s) may this cause?

```
graph.a_star(vertex_1, vertex_2, metric = lambda v1, v2 : random.rand())
```

Potentially calling a random number will create a giant number, causing the runtime to be massive

Would error- should be np.random.rand()

Creates an array of random values