CS 3851 Algorithms - Lab 5: Breadth-First Search

Stuart Harley

Introduction

In this lab we are implementing the breadth-first search algorithm used on graphs. We also implement a friend-recommendation algorithm similar to what can be used in social media platforms using the breadth-first search algorithm.

We then perform tests on this algorithm using a provided training and testing graphs. We look at the number of friends recommended, the time it takes, the precision, and the recall all against varying max depths.

Importing Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import time

from graphs import *
from test_graphs import *
```

Testing Implementation of Graphs and BFS

In [2]:
```
!python test_graphs.py
```

```
..........
----------------------------------------------------------------------
Ran 10 tests in 0.001s

OK
```

Evaluating the Recommend_All_Friends Algorithm (Uses BFS under the hood)

Loading in the provided datasets.

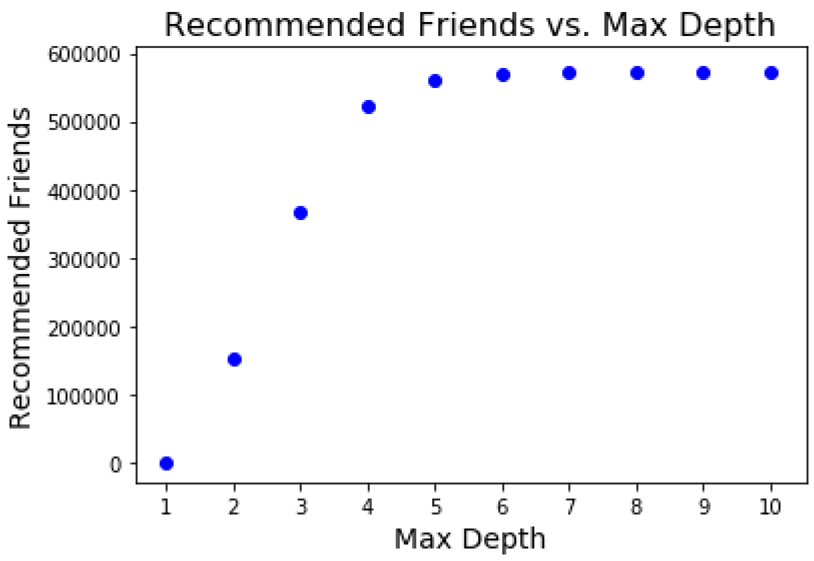
In [3]:
```
training_set, testing_set = load_data('facebook_data/training_set.tsv', 'faceb
ook_data/testing_set.tsv')
```

Evaluating how the number of recommendations varies with the max depth. Running the recommend_all_friends algorithm on the training set with max_depths of 1 to 10.

In [4]:
```
start = time.perf_counter()
recommended1 = recommend_all_friends(training_set, 1)
end = time.perf_counter()
r1_time = end-start
start = time.perf_counter()
recommended2 = recommend_all_friends(training_set, 2)
end = time.perf_counter()
r2_time = end-start
start = time.perf_counter()
recommended3 = recommend_all_friends(training_set, 3)
end = time.perf_counter()
r3_time = end-start
start = time.perf_counter()
recommended4 = recommend_all_friends(training_set, 4)
end = time.perf_counter()
r4_time = end-start
start = time.perf_counter()
recommended5 = recommend_all_friends(training_set, 5)
end = time.perf_counter()
r5_time = end-start
start = time.perf_counter()
recommended6 = recommend_all_friends(training_set, 6)
end = time.perf_counter()
r6_time = end-start
start = time.perf_counter()
recommended7 = recommend_all_friends(training_set, 7)
end = time.perf_counter()
r7_time = end-start
start = time.perf_counter()
recommended8 = recommend_all_friends(training_set, 8)
end = time.perf_counter()
r8_time = end-start
start = time.perf_counter()
recommended9 = recommend_all_friends(training_set, 9)
end = time.perf_counter()
r9_time = end-start
start = time.perf_counter()
recommended10 = recommend_all_friends(training_set, 10)
end = time.perf_counter()
r10_time = end-start
```

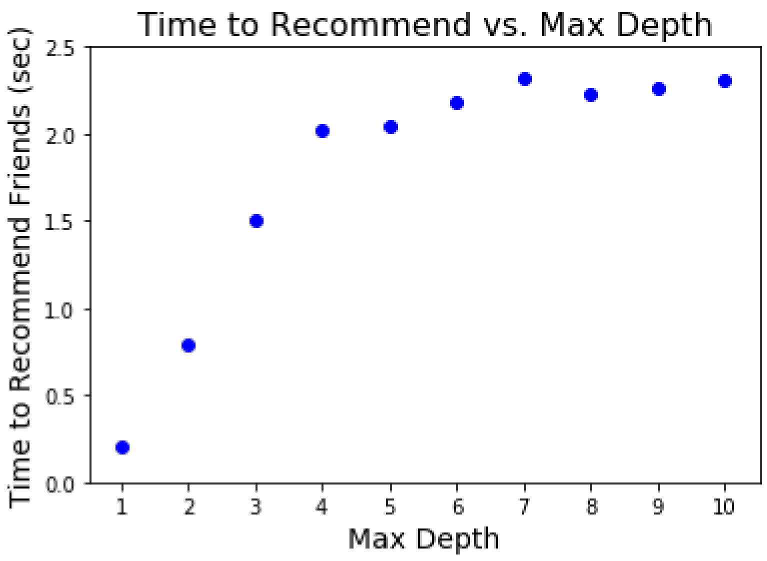
Plotting the number of edges (Number of Recommnedations) in the resulting graphs vs the max depth.

In [5]:
```
fig, axes = plt.subplots()
axes.scatter(1, recommended1.count_edges(), c='b')
axes.scatter(2, recommended2.count_edges(), c='b')
axes.scatter(3, recommended3.count_edges(), c='b')
axes.scatter(4, recommended4.count_edges(), c='b')
axes.scatter(5, recommended5.count_edges(), c='b')
axes.scatter(6, recommended6.count_edges(), c='b')
axes.scatter(7, recommended7.count_edges(), c='b')
axes.scatter(8, recommended8.count_edges(), c='b')
axes.scatter(9, recommended9.count_edges(), c='b')
axes.scatter(10, recommended10.count_edges(), c='b')
axes.set_xlabel('Max Depth', fontsize=14)
axes.set_xticks(np.arange(0, 10) + 1)
axes.set_ylabel('Recommended Friends', fontsize=14)
axes.set_title('Recommended Friends vs. Max Depth', fontsize=16);
```



Plotting the time it takes to recommend friends vs the max depth.

In [6]:
```python
fig, axes = plt.subplots()
axes.scatter(1, r1_time, c='b')
axes.scatter(2, r2_time, c='b')
axes.scatter(3, r3_time, c='b')
axes.scatter(4, r4_time, c='b')
axes.scatter(5, r5_time, c='b')
axes.scatter(6, r6_time, c='b')
axes.scatter(7, r7_time, c='b')
axes.scatter(8, r8_time, c='b')
axes.scatter(9, r9_time, c='b')
axes.scatter(10, r10_time, c='b')
axes.set_xlabel('Max Depth', fontsize=14)
axes.set_xticks(np.arange(0, 10) + 1)
axes.set_ylabel('Time to Recommend Friends (sec)', fontsize=14)
axes.set_yticks([0, .5, 1.0, 1.5, 2.0, 2.5])
axes.set_title('Time to Recommend vs. Max Depth', fontsize=16);
```



Calculating the precisions of the recommended friends. Precision indicates how many of the recommended edges are in the testing test.

Precision = number of recommended edges in test set / number of recommended edges

In [7]:
```
# precision1 = len(testing_set.edge_set().intersection(recommended1.edge_set
())) / recommended1.count_edges() division by 0
precision1 = 0
precision2 = len(testing_set.edge_set().intersection(recommended2.edge_set()))
/ recommended2.count_edges()
precision3 = len(testing_set.edge_set().intersection(recommended3.edge_set()))
/ recommended3.count_edges()
precision4 = len(testing_set.edge_set().intersection(recommended4.edge_set()))
/ recommended4.count_edges()
precision5 = len(testing_set.edge_set().intersection(recommended5.edge_set()))
/ recommended5.count_edges()
precision6 = len(testing_set.edge_set().intersection(recommended6.edge_set()))
/ recommended6.count_edges()
precision7 = len(testing_set.edge_set().intersection(recommended7.edge_set()))
/ recommended7.count_edges()
precision8 = len(testing_set.edge_set().intersection(recommended8.edge_set()))
/ recommended8.count_edges()
precision9 = len(testing_set.edge_set().intersection(recommended9.edge_set()))
/ recommended9.count_edges()
precision10 = len(testing_set.edge_set().intersection(recommended10.edge_set
())) / recommended10.count_edges()
```

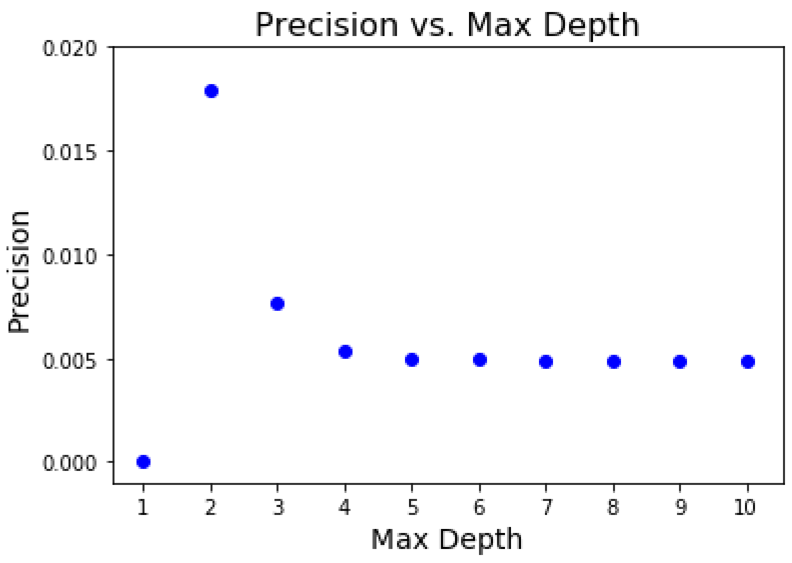
Calculating the recalls of the recommended friends. Recall indicates how many of the edges in the testing set are in the recommendations.

Recall = number of test set edges in recommended set / number of test set edges

In [8]:
```
recall1 = len(testing_set.edge_set().intersection(recommended1.edge_set())) /
testing_set.count_edges()
recall2 = len(testing_set.edge_set().intersection(recommended2.edge_set())) /
testing_set.count_edges()
recall3 = len(testing_set.edge_set().intersection(recommended3.edge_set())) /
testing_set.count_edges()
recall4 = len(testing_set.edge_set().intersection(recommended4.edge_set())) /
testing_set.count_edges()
recall5 = len(testing_set.edge_set().intersection(recommended5.edge_set())) /
testing_set.count_edges()
recall6 = len(testing_set.edge_set().intersection(recommended6.edge_set())) /
testing_set.count_edges()
recall7 = len(testing_set.edge_set().intersection(recommended7.edge_set())) /
testing_set.count_edges()
recall8 = len(testing_set.edge_set().intersection(recommended8.edge_set())) /
testing_set.count_edges()
recall9 = len(testing_set.edge_set().intersection(recommended9.edge_set())) /
testing_set.count_edges()
recall10 = len(testing_set.edge_set().intersection(recommended10.edge_set()))
/ testing_set.count_edges()
```

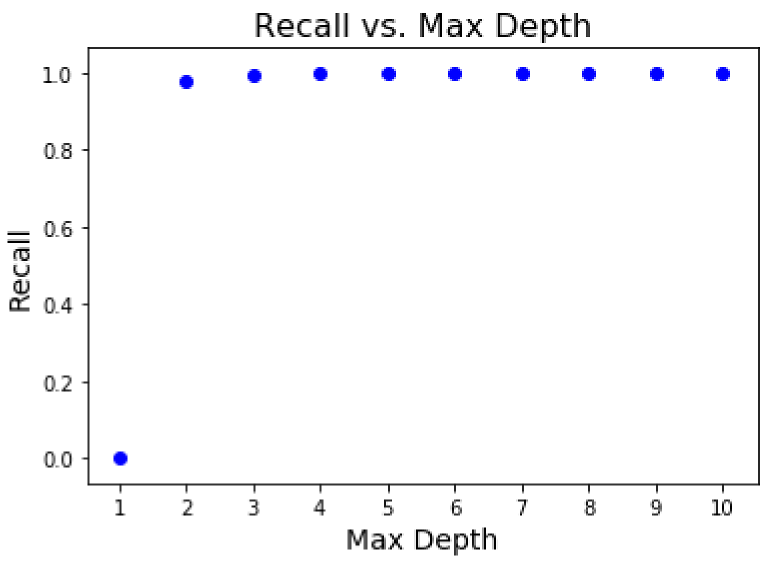
Plotting the precisions vs the max depth.

In [9]:
```python
fig, axes = plt.subplots()
axes.scatter(1, precision1, c='b')
axes.scatter(2, precision2, c='b')
axes.scatter(3, precision3, c='b')
axes.scatter(4, precision4, c='b')
axes.scatter(5, precision5, c='b')
axes.scatter(6, precision6, c='b')
axes.scatter(7, precision7, c='b')
axes.scatter(8, precision8, c='b')
axes.scatter(9, precision9, c='b')
axes.scatter(10, precision10, c='b')
axes.set_xlabel('Max Depth', fontsize=14)
axes.set_xticks(np.arange(0, 10) + 1)
axes.set_ylabel('Precision', fontsize=14)
axes.set_yticks([0, .005, .010, .015, .02])
axes.set_ylim(-.001)
axes.set_title('Precision vs. Max Depth', fontsize=16);
```



Plotting the recalls vs the max depth.

In [10]:
```
fig, axes = plt.subplots()
axes.scatter(1, recall1, c='b')
axes.scatter(2, recall2, c='b')
axes.scatter(3, recall3, c='b')
axes.scatter(4, recall4, c='b')
axes.scatter(5, recall5, c='b')
axes.scatter(6, recall6, c='b')
axes.scatter(7, recall7, c='b')
axes.scatter(8, recall8, c='b')
axes.scatter(9, recall9, c='b')
axes.scatter(10, recall10, c='b')
axes.set_xlabel('Max Depth', fontsize=14)
axes.set_xticks(np.arange(0, 10) + 1)
axes.set_ylabel('Recall', fontsize=14)
axes.set_title('Recall vs. Max Depth', fontsize=16);
```



Reflection Questions

1) Once a max depth of 6 is reached the number of recommended friends plateuas. This is because everyone in the training set graph is connected to everyone else by about 6 friends or less.

2) Precision decreases as the max depth increases. The testing set is a graph of friends that are the best friends to recommend. As max depth increases, the recommended friends are further and further away from the person they are being recommended to. There are also more of them as max depth increases. Therefore, the precision of the recommendations decreases because primarily only the recommendations from the lower max depths are going to be present in the testing set.

3) Recall increases as max depth increases. Recall measures how many of the connections in the testing set were present in the recommendations from the training set. Since the training set recommends more friends as the max depth increases, it becomes more likely that every connection in the testing set will be recommended.

4) Our stated goal is to generate an initial set of recommendations which we can further filter with another algorithm. If we needed to compare all users, we would have 784 * 786 = 614,656 pairs to look at. For this reason, I would choose a max depth of 2 to generate recommendations from. A max depth of 1 doesn't recommend any friends because if you are 1 connection away from someone, then they are already your friend. The next up is a max depth of 2 which still recommends 153,388 connections. This is still ~195 recommendations per user. This is more than enough to then further filter and present a lot of options to each user. Also, because these people are closer to you in the graph, they are more likely to be relevant than recommendations with a higher max depth.

5) Breadth first search is more appropriate than depth first search for this problem. This is because the closer users are going to be better recommendations than the further away users. Breadth first search searches all vertices at increasing distances from the source. So it will search all the most relevant friends first before going to less relevant recommendations. Depth first search expands downwards first, so it will search a lot of friends that are further away first which are going to be less relevant recommendations.

6) We modified the breadth-first search to limit the depth because, since we do not specify a goal vertex to reach (and stop at), the breadth first search would continue until every vertex has been searched. For our goal of recommending friends, it is not neccesary for us to search the entire graph, just the closest users which will be the most relevant recommendations. Therefore, we limit the max depth.

Conclusion

Social media platforms can use a "simple" breadth-first search algorithm to generate an initial list of recommended friends. These recommendations can then be further refined by more computationally expensive algorithms to generate the recommended friends that show up on our accounts.