Suman Karki
Advanced Analysis of Algorithm
11529935

A Simple Approximation Algorithm for the Weighted Matching Problem

Introduction

In graph theory, matching is the problem in which one finds the set of edges that do not have a set of common vertices. [1] In other words, one can say matching involves finding the set of edges in the graph where each vertex has either one or zero edge incident on it. The matching problem is very applicable in the field of flow networks, scheduling and planning, modeling bonds in chemistry, graph coloring, the stable marriage problem, neural networks in artificial intelligence and many more.

Formal Definition of Matching: Given a graph G (V, E): where V is the set of vertices and E is the set of edges, the matching problem is to find the set of edges M that is the subset of the edges E such that no two edges in M are incident. [2] In the given paper, the solution is proposed for the weighted matching problem: modified version of matching problem in which each edge has a weight associated with it. In this problem, we need to find the matching of the graph that has maximum weight.

For e.g., We can see the following graph of 10 nodes with 17 edges. Each edge has a weight associated with it.

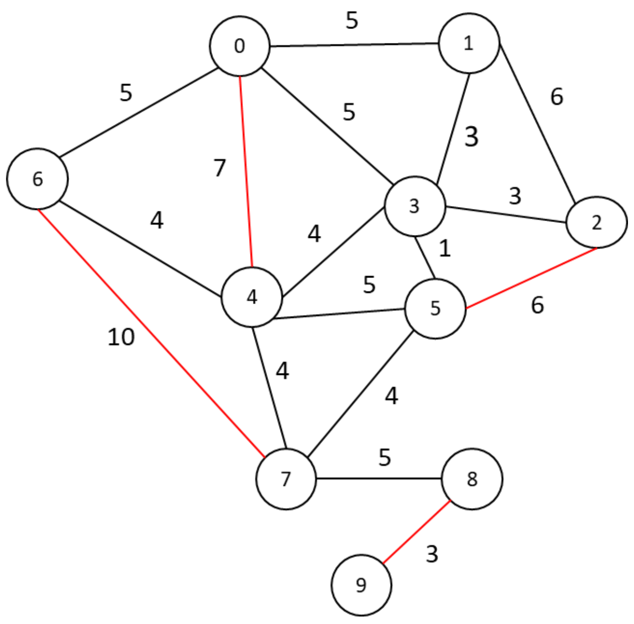


Figure 1: Example for Maximal Matching

The graph has many matching solutions, some of the few are:

1. { (0,4), (1,2), (5,7), (8,9) } ➔ 17
2. { (0,4), (1,3), (2,5), (6,7), (8,9) } ➔ 29
3. { (0,1), (2,5), (6,7), (8,9) } ➔ 24
4. { (0,4), (1,2), (3,5), (6,7), (8,9) } ➔ 27

We can see, various solutions have different sums, but the optimal solution for this graph (edges colored with red) is the second one in above list with maximum weight of 29.

The paper has presented a linear time approximation algorithm to solve the weighted matching problem and has compared its algorithm to the greedy approach. The greedy algorithm to solve the problem is given below:

```
Greedy Matching:
     Input: G (V, E), W_i
     Output: maximum weight
1. M = φ
2. while E ≠ φ, do
3.    Find e, the heaviest edge in E
4.    M = M ∪ e
5.    Remove e and all edge incident to e from E
6. end
```

The greedy approach is an approximation algorithm. The algorithm begins with the edge that has maximum weight and removing all other edges that are incident to it and repeating the same task until all edges are removed. It is believed that if the list of edges fed to this algorithm are sorted then it will take O(|E|log|V|) time to find the solution. [2]

The authors has improved the algorithm by introducing the random factor and by extending the path in one direction only as long as possible. The following is the algorithm they proposed:

```
Path Growing Algorithm:
     Input: G (V, E), W_i
     Output: maximum weight
1.  M1 = φ, M2 = φ, i=1
2.  While E ≠ φ, do
3.     Randomly choose a vertex x with at least degree 1
4.     While x has neighbor, do
5.          Let (x, y) be the heaviest edge incident on x
6.          Add (x, y) on M_i
7.          i = 3-i
8.          Remove x from G
9.          x = y
10.    End
11. End
12. Return max( w(M1), w(M2) )
```

The path growing algorithm keeps track of two sets of edges. Both sets of edges are the maximal matching. Only the set which has maximum weight is returned as solution. The algorithm runs for O(|E|) time. [2]

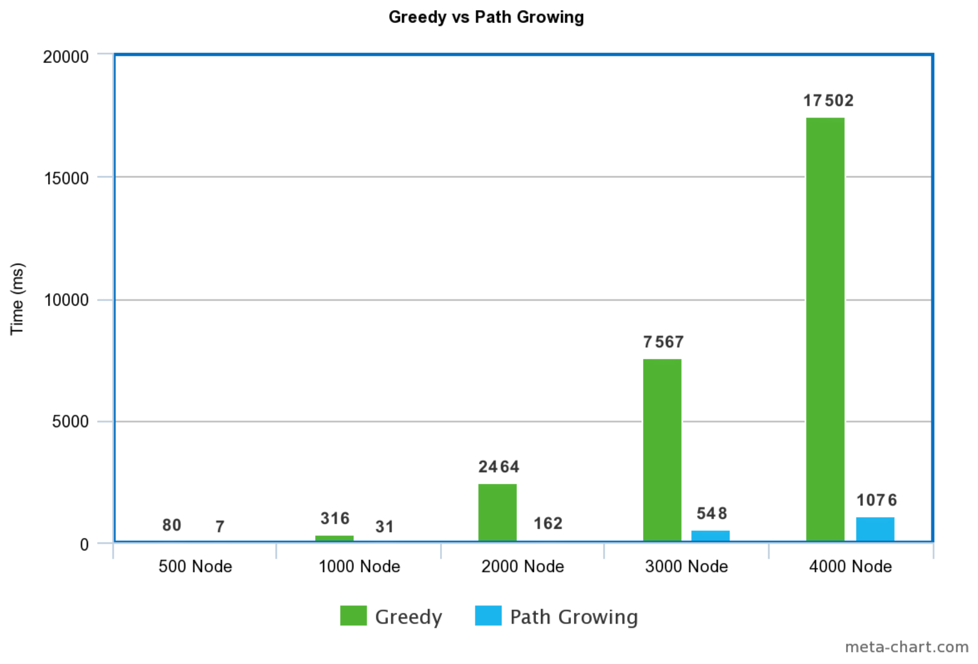
Implementation

To study the both algorithm, I have implemented both algorithm. For this I have used Java as programming language and for the organization of the edges of the graph, I have used adjacency matrix as the data structure. I monitored the run-time of the algorithm in second for the various graph that ranging from 500 vertices to 4000 vertices. The program was run in the machine with 4GB memory, Intel i5, 2.3 GHz with Java 1.8

The following table shows the run-time of the various dataset that I used for both Greedy and the Path growing algorithm.

	500	1000	2000	3000	4000
Number of Edges	124750	499500	1999000	4498500	7998000
Greedy Algorithm (ms)	80	316	2464	7567	17502
Path Growing Algorithm (ms)	7	31	162	548	1076

The graph for the same dataset is shown below:



Findings

The author has provided simple algorithm that is not hard to understand and the way it derives the solution is also really elegant. There is somewhat pretty much similarity in Path Growing algorithm and Greedy approach, as the few differences from greedy approach are the random picking of vertices and keeping track of two edge list while extending the current path in one direction. It's praise worthy that the algorithm does perform well than greedy approach and gives solution faster than its counterparts. While running the algorithm, I started with the graph having fewer edges and vertices (vertices count was less than 200), what I found was the maximum weight returned by greedy approach was larger than that of path growing algorithm (mostly 3 times out of 5), but the weight both algorithm returned was larger than the ½ of optimal value (which was tested with the graph above). But then again, the goal was to find the solution in O(|E|) time and has weight at least ½ of the optimal solution.

As the number of vertices and edges grew, the performance of the path-growing algorithm got better and better. Not only that, the maximum weight returned by the path-growing algorithm was also larger than that of greedy algorithm.

Conclusion

In my perspective, I liked the algorithm. With a creative modification to Greedy approach, it improves the running time by the factor of log(|V|). It is simple to understand, does not contain any complex calculations.

[1] Matching (graph theory), website, https://en.wikipedia.org/wiki/Matching_(graph_theory)

[2] D. E. Drake, S. Hougardy, A Simple Approximation Algorithm for the Weighted Matching Problem, 15th July, 2002.

[3] Meta chart, https://www.meta-chart.com