INVESTIGATION INTO MAXIMUM WEIGHT MATCHINGS

Team Number : 18
1. Amith Bhat Nekkare(181IT105)
2. Ankit Gupta(181IT107)
3. Ayush Rahangdale(181IT109)
4. Kumsetty Nikhil Venkat(181IT224)

INTRODUCTION

Some Definitions..

Given a graph $G = (V,E)$, a **matching** $M \subset E$ such that no two edges share a common vertex.

A **maximum matching** (also known as maximum-cardinality matching) is a matching that contains the largest possible number of edges. There may be many maximum matchings.

Maximum Weight Matching :

The problem of finding, in a weighted graph, a matching in which the sum of weights is maximized is called the <u>maximum weight matching</u>(MWM) problem.

Maximum Cardinality Matching :

The problem of finding, in a graph, a matching which contains as many edges as possible is called the <u>maximum cardinality matching</u>(MCM) problem.

- The MCM problem is a special case of the MWM problem.

Many real world applications of the MWM problem, require graphs of such large size that the running time of the fastest available weighted matching algorithm is too costly.

Examples of such problems are the refinement of FEM nets, the partitioning problem in VLSI-Design, and the gossiping problem in telecommunications. There also exist applications were the weighted matching problem has to be solved extremely often on only moderately large graphs.

Therefore, there is considerable need for **approximation algorithms** for the weighted matching problem that are very fast, and that nevertheless produce very good results even if these results are not optimal.

REQUIREMENTS ANALYSIS

HARDWARE REQUIREMENTS

- Processors: Will run on Intel Pentium and newer
- Hard Disk : 1 TB
- RAM : 4 GB

SOFTWARE REQUIREMENTS

- OS : Ubuntu 16.04 and upwards
- Requires Python 3.8 interpreter.
- Python libraries such as numpy and networkx(to generate graphs).

PROBLEM STATEMENT

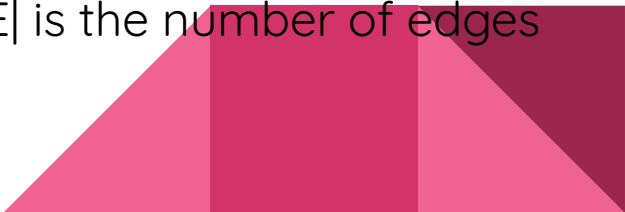To investigate and implement the following algorithms in the field of MWMs :

1. Edmonds' MCM Algorithm
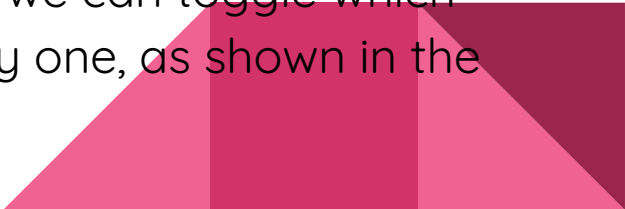2. The Drake-Hougardy MWM Algorithm

EDMONDS' ALGORITHM

Introduction

- Also known as the Blossom algorithm, the algorithm was developed by Jack Edmonds in 1961, and published in 1965.
- As mentioned earlier, the algorithm is a solution to the MCM problem, which is a subset of the MWM problem.
- Given a general graph $G = (V, E)$, the algorithm finds a matching M such that each vertex in V is incident with at most one edge in M and $|M|$ is maximized. The matching is constructed by iteratively improving an initial empty matching along augmenting paths in the graph.
- The algorithm runs in time $O(|E| * (|V|^2))$, where $|E|$ is the number of edges of the graph and $|V|$ is its number of vertices.
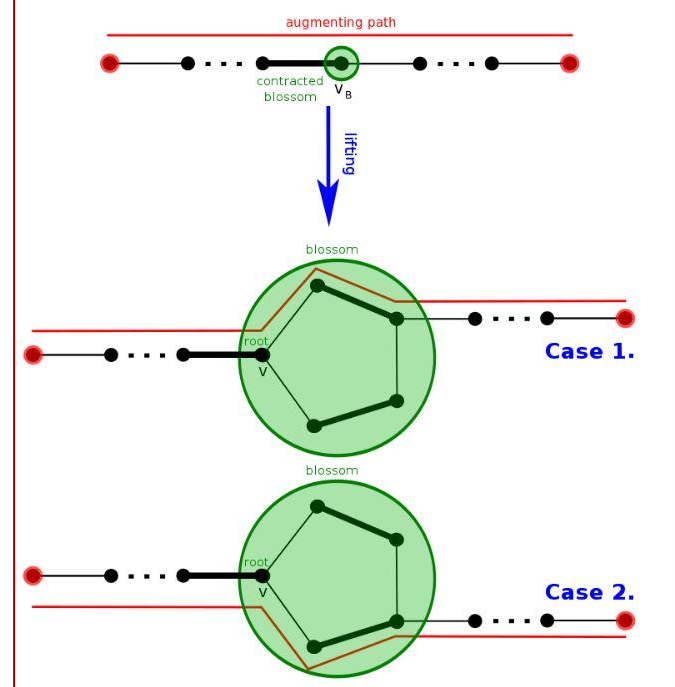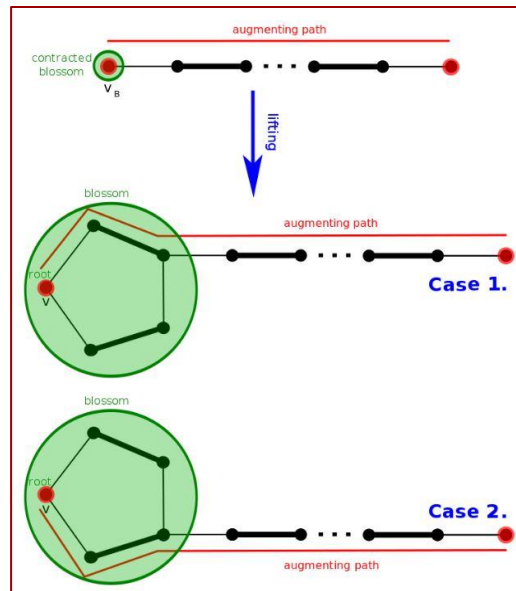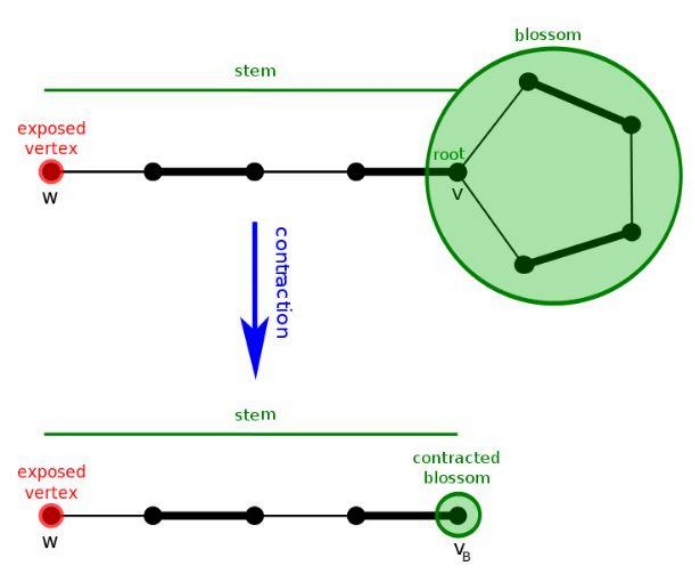
Definitions

For graph G with matching M, an **exposed vertex** v is a vertex that does not belong to M, but is in the graph G. That is V (G \ M) are exposed vertices.

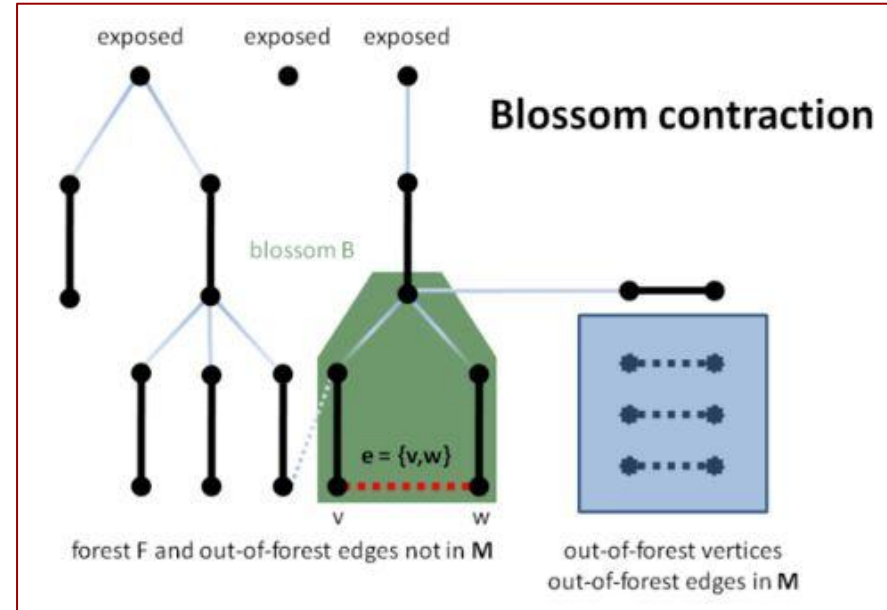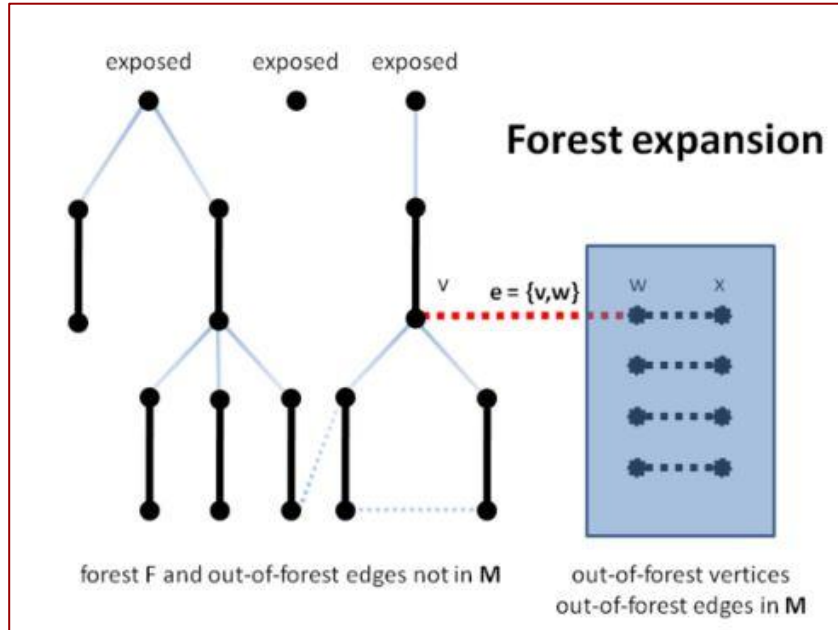Given a matching M, an **augmenting path** is an odd length path whose endpoints are distinct exposed vertices, and whose edges $e_1, \ldots, e_{2k+1}$ alternate such that $e_{2i+1} \notin M$ for all $0 \leq i \leq k$ and $e_{2j} \in M$ for all $1 \leq j \leq k$.
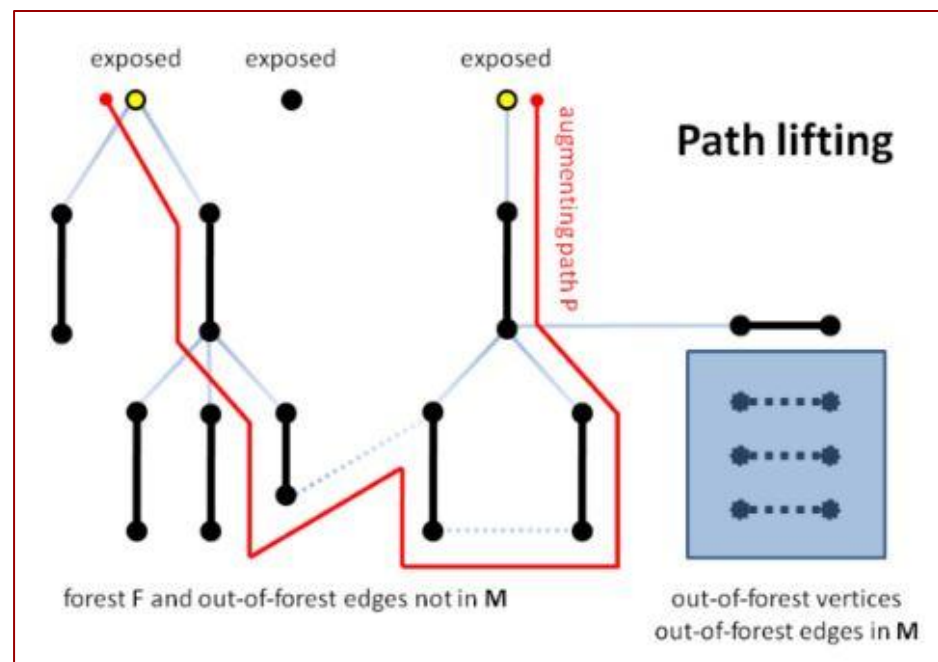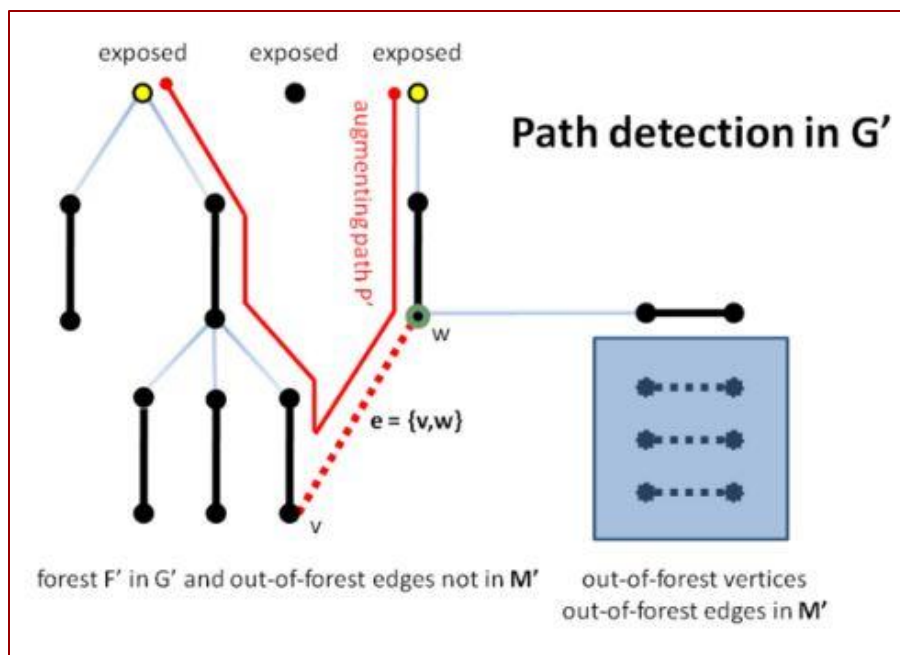
An augmenting path as defined above is an augmenting path of size $2k + 1$ and has k edges belonging to the matching and $k + 1$ which are not part of the matching. Our goal in finding augmenting paths is that we can toggle which edges are included in order to increase the matching by one, as shown in the following theorem.

Example

Forest expansion

v   e = {v,w}   w   x

forest F and out-of-forest edges not in **M**

out-of-forest vertices
out-of-forest edges in **M**

Blossom contraction

blossom B

e = {v,w}

v   w

forest F and out-of-forest edges not in **M**

out-of-forest vertices
out-of-forest edges in **M**

Path detection in G'

exposed   exposed   exposed

augmenting path P'

$e = \{v, w\}$

w

v

forest F' in G' and out-of-forest edges not in **M'**

out-of-forest vertices
out-of-forest edges in **M'**

Path lifting

exposed   exposed   exposed

augmenting path P

forest F and out-of-forest edges not in **M**

out-of-forest vertices
out-of-forest edges in **M**

Pseudo code

Algorithm 1 Blossom Algorithm: Find Maximum Matching

1: **procedure** SEQ_FIND_MAXIMUM_MATCHING(G, M)
2: $\quad P = $ SEQ_FIND_AUG_PATH(G, M)
3: $\quad$ **if** $P == [\,]$ **then**
4: $\quad\quad$ **return** M
5: $\quad$ **else**
6: $\quad\quad$ Add alternating edges of P to M
7: $\quad\quad$ **return** SEQ_FIND_MAXIMUM_MATCHING(G, M)

Algorithm 2 Sequential Blossom Algorithm: Find Augmenting Path

1: **procedure** SEQ_FIND_AUG_PATH(G, M)
2:     F = empty forest
3:     nodes_to_check $\leftarrow$ exposed vertices in G
4:     **for** v in nodes_to_check **do**
5:        Add v as single-node tree to F
6:        node_to_root$(v) = v$
7:     in G, mark all matched edges (all edges in M)
8:     **for** v in forest_nodes **do**
9:        **while** there exists an unmarked edge $e = (v, w)$ **do**
10:           **if** $w \notin F$ **then** $\triangleright$ Vertex w must be in M
11:              SEQ_ADD_TO_FOREST(M, F, v, w)
12:           **else**
13:              **if** dist$(w,$ node_to_root$(w))$ % 2 == 0 **then**
14:                 **if** node_to_root$(v) \neq$ node_to_root(w) **then**
15:                    P = SEQ_RETURN_AUG_PATH$(F, v, w,$ node_to_root$)$
16:                 **else**
17:                    P = SEQ_BLOSSOM_RECURSION(G, M, F, v, w)
18:                 **return** P
19:              **else**
20:                 # Do nothing
21:           mark edge e
22:     **return** empty path

Algorithm 3 Sequential Blossom Algorithm: Add to Forest

1: **procedure** SEQ_ADD_TO_FOREST(M, F, v, w)
2:     $x \leftarrow$ vertex adjacent to w in M
3:     add edges $(v, w), (w, x)$ to tree(v) in F
4:     add vertex x to nodes_to_check
5:     node_to_root(w) = node_to_root(v)
6:     node_to_root(x) = node_to_root(v)

Algorithm 4 Sequential Blossom Algorithm: Return Aug Path

1: **procedure** SEQ_RETURN_AUG_PATH($F, v, w, node\_to\_root$)
2:     root_v = node_to_root(v)
3:     root_w = node_to_root(w)
4:     $P1 \leftarrow$ SHORTEST_PATH(F,root_v,v)
5:     $P2 \leftarrow$ SHORTEST_PATH(F, w,root_w)
6:     **return** $P1 + P2$

THE DRAKE- HOUGARDY APPROXIMATION ALGORITHM

Introduction

In this project, we have implemented two different solutions to the MWM problem: Edmonds' Algorithm and the the Drake-Hougardy solution.

The Drake-Hougardy Algorithm, as an approximation algorithm, provides a linear-time approximate solution to the MWM problem with a performance ratio of ⅔ - ε, where ε is an arbitrarily small number.

An approximation algorithm has a **performance ratio** of c, if for all graphs it finds a matching with a weight of at least c times the weight of an optimal solution.

The quality of an approximation algorithm for the weighted matching problem is measured by its performance ratio.

Definitions

Let M be a matching in a graph G = (V, E).

If the weight of M is not maximum, then one can replace some set of edges of M by some other set of edges of E such that the new set thus obtained is again a matching and has strictly larger weight than M .
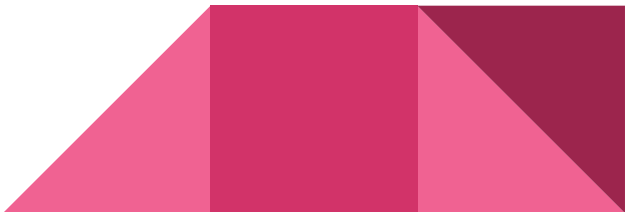
A process which removes the set R and adds a set S ⊂ E to M is defined as an **augmentation**. The set S is called the <u>augmenting set</u> for this augmentation.

The **gain** of an augmentation with augmenting set S is defined as w(S) – w(R), which is the increase of weight it achieves.

A **short augmentation** is an augmentation such that all edges in the augmenting set are adjacent to some edge e ∈ E. Such an edge e is called a **center** of this short augmentation.

For a constant β > 1, a **β-augmentation** is a short augmentation such that the augmenting set S has the property that w(S) ≥ β · w(R).
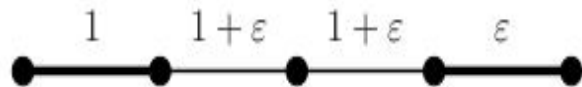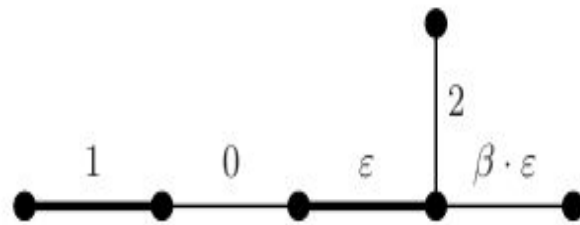
Fig. 2. Examples motivating the use of β-augmentations. Edges belonging to the matching are shown in bold.

We must also be careful to pick the choices for the beta augmentation carefully, as with the wrong choice, the total weight of the matching may only increase by smaller values, leading to increase in the running time.

Hence, for the purposes of the algorithm, the β- augmentation with the largest possible gain should be selected. But, since selecting the best β- augmentation will take longer time, we strike a balance by selecting a "good" β- augmentation.

A β- augmentation with center e is called **good** if it achieves at least a $(β – 1)/(β – 1/2)$ fraction of the gain that a best β-augmentation with center e can achieve.

Main Algorithm

Algorithm improve_matching $(G = (V, E), w : E \to \mathbb{R}^+, M)$

1   make M maximal
2    $M' := M$
3   **for** $e \in M$ **do begin**
4   **if** there exists a β-augmentation in M' with center e
5   **then** augment M' by a good β-augmentation with center e
6   **end**
7   **return** M'

RESULTS

Edmond's Algorithm

with n = 50 , d = 0.7
[33, 32]
[36, 35]
[38, 37]
[41, 34]
[44, 11]
[44, 12]
[44, 43]
[49, 48]
[47, 17]
[47, 21]
[47, 16]
[47, 14]
[47, 11]
[47, 12]

            Took 0.976536750793457 seconds
       with n = 50 , d = 0.9
[34, 33]
[3, 35]
[37, 3]
[37, 1]
[37, 36]
[39, 3]
[39, 1]
[39, 38]
[11, 42]
[11, 43]
[45, 11]
[45, 9]
[45, 5]
[45, 1]
[45, 7]
[45, 3]
[45, 44]
[49, 48]
[47, 3]

            Took 1.4810512065887451 seconds

Drake - Hougardy Approximation Algorithm

Drake - Hougardy Approximation Algorithm

CONCLUSION

In this project, we have investigated the MWM problem and implemented two algorithms as its solutions : the Edmonds' Algorithm and the Drake-Hougardy Approximation Algorithm.

In the approximate algorithm, the weight of the matching obtained is 0.67 times the weight of the exact solution, as expected. The program takes around 0.1 seconds to output a matching for an input of 50 nodes on average.

In the blossom algorithm, we have obtained the running times for different number of nodes. We also print the final matching obtained.

THANK YOU