Comparison Between Matching Algorithms

Rathijeet Bhave

October 31, 2014

Outline

This project is mainly divided into two parts.

- In **part-1** we try to find and study different algorithms that provide matching between two sets of a bipartite graph.
- Implement them in a programming language (python in this case) to be used as a solution to various applications.
- In **part-2** we try to compare between the results of these algorithms when same input is given to all of them.
- Specifically to compare the extent of sub-optimality caused due to constraints like stability in matching or maximization/minimization of weights of edges

Objective

Objective of this project is to address allocation problems like

- admissions of student to colleges.
- allocation of TA's to courses.
- allocation of committee's to students who have applied for PHD for conducting their interviews.

Students to Colleges Allocation

- In this allocation we want to consider the preferences of both students and colleges.
- In this case there should be no student-college pairing such that the student prefers some other college to which he is currently matched and the college is also better off with other student in place of him.
- In other words we want a **stable matching.**

TA to Course Allocation

- In this allocation students give preferences of courses and instructors give preferences of students.
- Even if there is a conflict of preferences, the student can be allocated to professor's choice of course.
- In this matching the criteria is that no student should be left without a TA duty.
- In other words we want a **maximal matching**.

Invigilation Duty Allocation

- The instructors give preference of centers according to their choices but exam centers does not give their choices of instructors.
- The only constraint is a fixed number of instructors are to be allocated per exam center.
- The preference order of instructors can be considered as weights of edges from instructors to centers.
- In this problem we want that no exam center should be left without enough invigilators but we also want to consider preference of invigilators as much as possible.
- In other words we want matching to be **maximal along with the criteria that edge weights should be maximized.**

Prelimenaries

Weighted bipartite graphs : These are graphs in which each edge (i, j) has a weight, or value, $w(i, j)$. The weight of matching M is the sum of the weights of edges in **M**, w(M) $= {}_{e \in M}$ w(e).

Matching : A matching is a subset $M \subseteq E$ such that $\forall v \in V$ at most one edge in M is incident upon v.

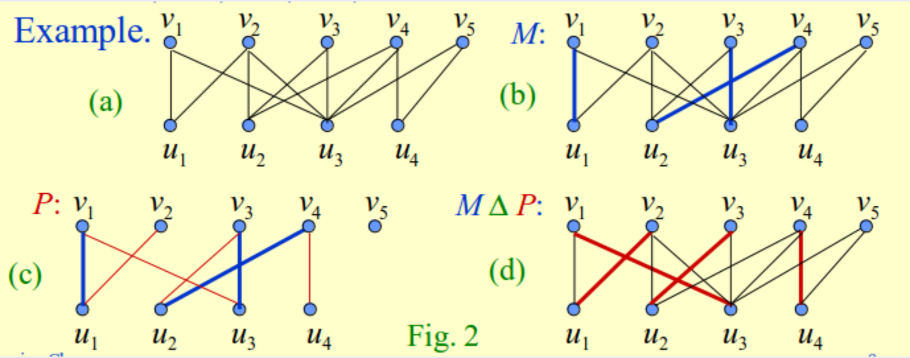
Prelimenaries

Maximum Matching : A Maximum Matching is a matching M such that every other matching M' satisfies $|M'| \leq |M|$

Perfect Matching : A perfect matching is a matching in which every vertex is adjacent to some edge in M. Let M be a matching of G. Vertex v is matched if it is endpoint of edge in M ; otherwise v is free.

Prelimenaries

Augmenting Path : An alternating path is augmenting if both end-points are free. Augmenting path has one less edge in M than in $E - M$,thus replacing the M edges by the $E - M$ ones increments size of the matching by one.



[1]

Hungarian algorithm

Introduction

- The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time.
- The running time complexity of this algorithm is $\mathcal{O}(n^3)$.
- This algorithm gives us a matching in which the edge weights are maximized/minimized as per our requirement.
- A matching is said to be maximum if sum of weights of all the edges in the matching is more than any other perfect matching.

The Hungarian method

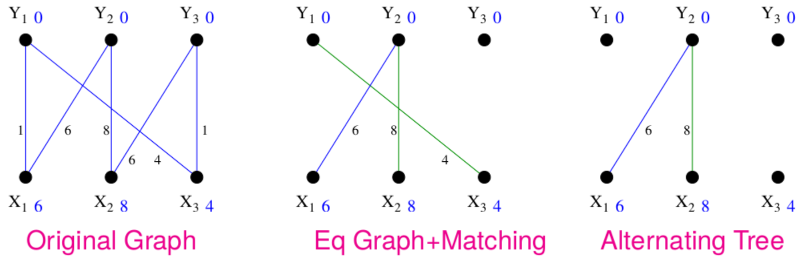
1. Generate initial labellings ℓ and matching M in E_ℓ .
2. If M is perfect, stop.
   Otherwise pick free vertex $u \in X$. Set $S = u$, $T = \phi$.
3. If $N_\ell(S) = T$, update labels (forcing $N_\ell(S) \neq T$). Set

$$\alpha_\ell = min_{x \in X, y \notin T}\{\ell(x) + \ell(y) - w(x,y)\}$$

   and

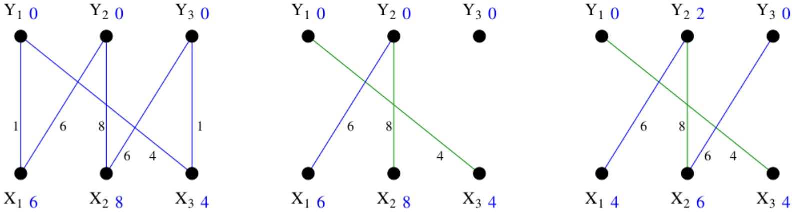
$$\ell'(v) = \begin{cases} \ell(v) - \alpha_\ell & \text{if } v \in s \\ \ell(v) + \alpha_\ell & \text{if } v \in T \\ \ell(v) & \text{otherwise} \end{cases}$$

4. If $N_\ell(S) \neq T$, pick $y \in \{N_\ell(S) - T\}$.
   - If y free, (u - y) is augmenting path. Augment M and go to 2.
   - If y matched, say to z, extend $S = S \cup \{z\}$, $T = T \cup \{y\}$.
   - Go to 3.



[2]

- Initial Graph, trivial labelling and associated Equality Graph.
- Initial matching: $(x_3, y_1), (x_2, y_2)$
- $S = \{x_1\}, T = \phi$.
- Since $N_\ell(S) \neq T$ Choose $y_2 \in N_\ell(S) - T$.
- y_2 is matched so $S = \{x_1, x_2\}, T = \{y_2\}$.
- At this point N_ℓ (S) = T , so goto 3.



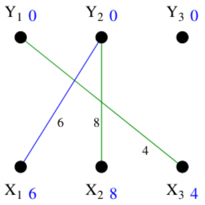
Original Graph   Old E_ℓ and $|M|$ new Eq Graph

[2]

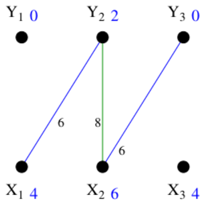
- $S = \{x_1, x_2\}$, $T = \{y_2\}$ and $N_\ell(S) = T$
- calculate α_ℓ as

$$\ell'(v) = \min_{x \in S, y \notin T} \begin{cases} 6+0-1, & (x_1, y_1) \\ 6+0-0, & (x_1, y_3) \\ 8+0-0, & (x_2, y_1) \\ 8+0-6, & (x_2, y_3) \end{cases} = 2$$

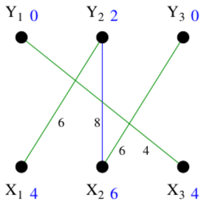
- Reduce labels of S by 2; Increase labels of T by 2.
- $N_\ell(S) = \{y_2, y_3\} = \{y_2\} = T$.



Orig E_ℓ and M



New Alternating Tree



New M

[2]

- $S = \{x_1, x_2\}, N_\ell(S) = \{y_2, y_3\}, T = \{y_2\}$
- Choose $y_3 \in N_\ell(S) - T$ and add it to T

- y_3 is not matched in M so x_1, y_2, x_2, y_3 is an alternating path.
- After augmenting we get a mew matching M.

Complexity analysis

In each phase of algorithm, $|M|$ increases by 1 so there are at most $\mathcal{O}(|V|)$ phases. Now we find how much work needs to be done in each phase.
In implementation, $\forall y \notin T$, keep track of
$slack_y = \min_{x \in S}\{\ell(x) + \ell(y) - w(x,y)\}$.

- Initializing all slacks at beginning of phase takes $\mathcal{O}(|V|)$ time.
- In step 4 we must update all slacks when a vertex moves in set S. This takes $\mathcal{O}(|V|)$ time. As at-most $|V|$ vertices can move to set S the running time of this phase will be $\mathcal{O}(|V|^2)$.

- In step 3, $\alpha_\ell = \min_{y \in T} slack_y$ and can therefore be calculated in $\mathcal{O}(|V|)$ time from the slacks. This in worst case would be done at most $|V|$ times per phase.This is because only $|V|$ vertices can be moved to set S. So this step also takes $\mathcal{O}(|V|^2)$ time.
- There are $|V|$ phases and $\mathcal{O}(|V|^2)$ work per phase. so the total running time is $\mathcal{O}(|V|^3)$

$$InputMatrix = \begin{pmatrix} 7 & 2 & 8 \\ 8 & 0 & 2 \\ 4 & 8 & 7 \end{pmatrix}$$

Hopcroft Karp Algorithm

- In this algorithm we find a maximal family of vertex-disjoint shortest-length augmenting paths and augment all of them together in a single stage.
- This improvement will help us to bring the time complexity down to $\mathcal{O}(n^{2.5})$.

The algorithm is as follows.

M $= \phi$

while *there is an M -augmenting path* **do**

  Find a maximal family F of vertex-disjoint shortest M -augmenting paths;

  set $M = M \oplus F$

end

return M

Algorithm 1: Hopcroft Karp Algorithm

Theorem

M is a maximum matching if and only if there is no augmenting path relative to M.

Theorem

Let M be a matching. Suppose $|M| = r$, and suppose that the cardinality of a maximum matching is s, $s > r$. Let n be the total number of vertices. Then there exists an augmenting path relative to M of length $\leq \frac{n}{s-r} - 1$.

Hint

$$(s - r)(\ell + 1) \leq n.$$

Theorem

Let M be a matching, P a shortest augmenting path relative to M, and P' an augmenting path relative to $M \oplus P$. Then

$$|P'| \geq |P| + 2|P \cap P'|$$

.

Hint

$$|M \oplus N| = |P \oplus P'| = |P| + |P'| - 2|P \cap P'| \geq |P1| + |P2| \geq 2|P|$$

.

Corollary

Let P be a shortest augmenting path relative to a matching M, and Q be a shortest augmenting path relative to $M \oplus P$. Then, if $|P| = |Q|$, the paths P and Q must be node-disjoint.

Finding augmenting paths

Corollary

After each phase the shortest augmenting path is strictly longer than the shortest augmenting path of the previous phase.

Theorem

A maximal set of vertex disjoint minimum length augmenting paths can be found in $\mathcal{O}(m)$ *time.*

Python Implementation

First step: To find the length of minimum length augmenting path.
Construct a graph G_i in the same way as 'H' was constructed in 22.
From G_i, construct a subgraph $G_{i'}$ described below.
Let L_0 be the set of free nodes relative to M_i in A and define $L_j (j > 0)$ as follows:

$$E_{j-1} = \{u \rightarrow v \in E(G_i) \mid u \in L_{j-1}, v \in L_0 \cup L_1 \cup ... \cup L_{j-1}\}$$

$$L_j = v \in V(G_i) | \text{ for some u}, u \rightarrow v \in E_{j-1}.$$

Define $j* = min\{j | L_j \cap \{\text{free nodes in B}\} \neq \phi\}$
G_i' is formed with $V(G_i')$ and $E(G_i')$ as defined below.
If $j* = 1$, then

$$V(G_i') = L_0 \cup (L_1 \cap \{\text{free nodes in B}\})$$

$$E(G_i') = \{u \rightarrow v | u \in L_0 \text{ and } v \in \{\text{free nodes in B}\}\}.$$

If $j* > 1$, then

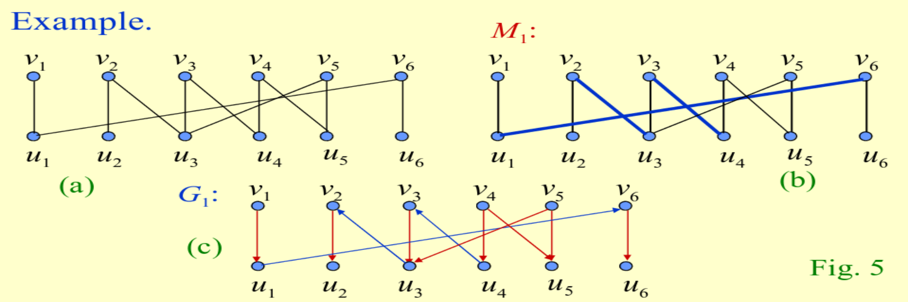
$$V(G_i') = L_0 \cup L_1 \cup ... \cup L_{j*-1} \cup (L_{j*} \cap \{\text{free nodes in B}\}),$$

$$E(G_i') = E_0 \cup E_1 \cup ... \cup E_{j*-2} \cup \{u \rightarrow v | u \in L_{j*-1} andv \in \{\text{free nodes in B}\}.$$

Second step: To find maximal set of shortest augmenting paths.
Data structure stack is used to temporarily store the augmenting paths.
c-list of a vertex 'v' is defined as all the vertices which are connected to 'v' through an edge. The algorithm is as follows.

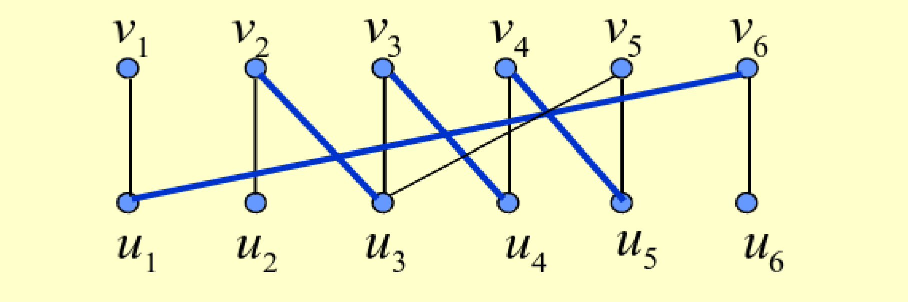
```
let v be the first element in L_0; push(v, stack); mark v;
while stack is not empty do
    v := top(stack);
    while c-list(v) ≠ φ do
        let u be the first element in c-list(v);
        if u is marked then
            remove u from c-list(v)
        else
            push(u, stack);
            mark u; v := u;
        end
    end
    if v is neither in L_{j*} nor in L_0 then
        pop(stack)
    else
        if v is in L_{j*} then
            output all the elements in stack; (all the elements in stack make up an
            augmenting path.)
            remove all elements in stack;
            let v be the next element in L_0;
            push(v, stack); mark v;
        end
    end
end
```

Algorithm 2: Augmrnting path algorithm



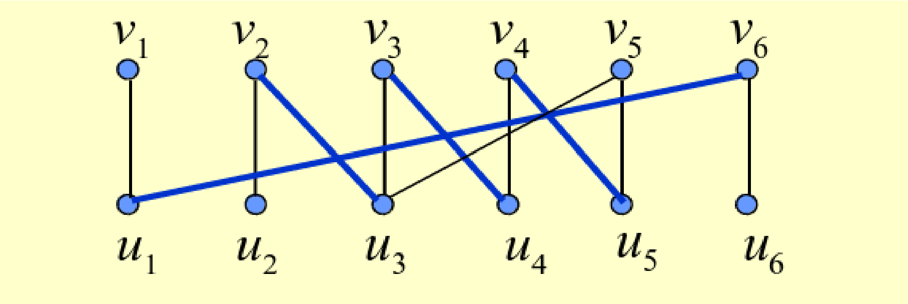
[1]

- $L_0 = \{v_1, v_4, v_5\}$
- $E_0 = \{(v_1, u_1), (v_4, u_4), (v_4, u_5), (v_5, u_3), (v_5, u_5)\}$
- $L_1 = \{u_1, u_3, u_4, u_5\},\ j^* = 1.$
- $V(G_1') = \{v_1, v_4, v_5\} \cup \{u_5\}$
- $E(G_1') = \{(v_4, u_5), (v_5, u_5)\}$



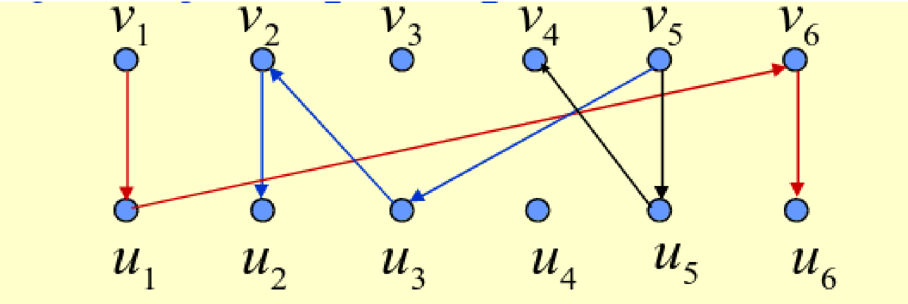
[1]

- $v_4 \to u_5$ and $v_5 \to u_5$ are augmenting paths out of which $v_4 \to u_5$ is selected
- $M_2 = M_1 \oplus \{v_4 \to u_5\}$



[1]

- $L_0 = \{v_1, v_5\}$
- $E_0 = \{v_1, u_1), (v_5, u_3), (v_5, u_5)\}$
- $L_1 = \{u_1, u_3, u_5\}$
- $E_1 = \{(u_1, v_6), (u_3, v_2), (u_5, v_4)\}$
- $L_2 = \{v_2, v_4, v_6\}$
- $E_2 = \{(v_2, u_2), (v_4, u_4), (v_6, u_6)\}$
- $L_3 = \{u_2, u_4, u_6\}$



[1]

- $j^* = 3$
- $V(G_2') = L_0 \cup L_1 \cup L_2 \cup \{u_2, u_6\}$
- $E(G_2') = E_0 \cup E_1 \cup \{(v_2, u_2), (v_6, u_6)\}$
- Start a DFS from v_1, v_5 to obtain augmenting paths.
- Final matching $= \{(v_1, u_1), (v_6, u_6), (v_2, u_2), (v_5, u_3), (v_3, u_4), (v_4, u_5)\}$

Complexity analysis

- Algorithm of finding a maximal set of vertex disjoint shortest length augmenting paths can be implemented in $\mathcal{O}(m)$ time.
- Let M be the matching obtained after exactly $\sqrt{n}$ phases.
- Each augmenting path from now on is atleast of length $2\sqrt{n}+1$. This is because after each phase the length of shortest augmenting path increases by atleast 2.
- Let M^* be the maximum matching. So there exists atleast $|M^*|-|M|$ vertex disjoint augmenting paths.
- The length of shortest augmenting path will utmost be $\frac{n}{|M^*|-|M|}-1$.

- Following inequality holds

$$\sqrt{n} \leq (\text{length of shortest M -augmenting path}) \leq \frac{n}{|M^*| - |M|}$$

and so on simplification we get

$$|M^*| - |M| \leq \sqrt{n}$$

From this point onwards, we need at most $\sqrt{n}$ more iterations,
- Thus overall no more than $2\sqrt{n}$ iterations are needed.
- The overall running time can now be written as $\mathcal{O}(\sqrt{n}m)$.
- In a bipartite graph the number of edges cannot be more than $n^2/4$. So the overall running time would be

$$\mathcal{O}(\sqrt{n} \times n^2) = \mathcal{O}(n^{2.5})$$

Gale Shapely Algorithm

Introduction

This algorithm gives solution to Stable matching problem. A matching is stable whenever it is not the case that both the statements hold true:

1. some given element A of the first matched set prefers some given element B of the second matched set over the element to which A is already matched.
2. B also prefers A over the element to which B is already matched. Algorithm for finding solution of stable marriage problem is given below.

```
Initialize all m ∈ M and w ∈ W to free
while ∃ free man m who still has a woman w to propose to do
    w = m's highest ranked woman to whom he has not yet proposed
    if w is free then
        (m, w) become engaged
    else
        *(some pair (m', w) already exists)*
        if w prefers m to m' then
            (m, w) become engaged
            m' becomes free
        else
            (m', w) remain engaged
        end
    end
end
```

Algorithm 3: Gale Shapely algorithm

$$malepref = \begin{pmatrix} 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \\ 2 & 3 & 1 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix} femalepref = \begin{pmatrix} 2 & 1 & 4 & 3 \\ 4 & 3 & 1 & 2 \\ 1 & 4 & 3 & 2 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

First round : $1 \rightarrow free, 2 \rightarrow 3, 3 \rightarrow 2, 4 \rightarrow 4$
Second round: $1 \rightarrow 4, 2 \rightarrow 3, 3 \rightarrow 2, 4 \rightarrow free$
Third round: $1 \rightarrow 1, 2 \rightarrow 3, 3 \rightarrow 2, 4 \rightarrow 1$

$$answer = \begin{pmatrix} 1 & 1 \\ 2 & 4 \\ 3 & 2 \\ 4 & 3 \end{pmatrix}$$

Complexity Analysis

- The worst case for this algorithm would be when one man gets his last preference and all others get their penultimate preferences i.e when the number of proposals are maximum.
- Assume there are 'n' men and 'n' women. Each iteration has atleast one proposal.
- Only an engaged women can reject. So no man can be rejected by all women.
- No man proposes twice to same women.So total number of proposals are upper bounded by n^2. Hence the running time of this algorithm is $\mathcal{O}(n^2)$.

- John E.Hopcroft and Richard M.Karp *AN $n^{5/2}$ Algorithm For Maximum Matchings In Bipartite Graphs*
- H. W. Kuhn *The Hungarian Method For The Assignment Problem* Bryn Yaw College 1955
- Dan Gusfield and Robert W.Irving, *Stable Marriage Problem Structure and* Oxford Handbook of Innovation The MIT Press Cambridge Massachusetts London England 1989,

THANK YOU