A Survey on
Matching Algorithms in Graphs

By Rishitha Reddy Kallu

Abstract

This survey paper is regarding the matching algorithms in graph theory. In this paper, we first introduce the problem and then present the algorithms proposed by different researchers by giving their approaches of solving the problem in sections 2 to 5. We also give various lemmas and theorems proposed along with their proofs. The results of solving the assignment problems and maximum matching problems are presented in Section 6. The application of the assignment algorithm to the solution of the asymmetric traveling salesman problem and the maximum matching problem to the solution of the chain decomposition of a partially ordered set are covered in section 7.

1. Introduction

1.1 Problem Statement

The problem of personnel-assignment is finding the optimal assignment for a set of persons n to a set of jobs n, according to the workers ratings in the specific jobs to which they have been assigned. The optimal scenario will have n! number of possible assignments which is practically impossible. The goal is to create an effective algorithm that will enable optimal assignment to be acquired while minimizing the overall cost of assignments. The group of edges E in a finite graph G without loops, where no two edges are incident to the same vertex V, is referred to as an independent edge set, or Matching M. Maximal matching is one that does not belong to subset of any matching. Maximum matching in a graph occurs when a match has the highest cardinality relative to all other matches. In a bipartite graph, the problem to find maximum matching is where the cardinality is maximum.

1.2 Mathematical Statement

The problem can be represented in a matrix as a set of elements which are independent when no two lies in the same row and the same column of a matrix. The jobs are represented as J_j and person as M_i where the worker rating is R_{ij}. The goal is to choose n independent elements for matric R_{ij} in a way that the sum should be minimum. Perfect matching is obtained when it is represented on undirected bipartite graph. In a bipartite graph G = (S U T, A) where the ratings R_{ij} is in A and |S| = |T| = n, the assignment problem finds the pair of nodes in S to T in such a way that costs of arcs while pairing is minimized. This can be represented as min $\Sigma(R_{ij}\ X_{ij})$ where i is a part of S and j is a part of T. If i is a part of S, then $\Sigma(X_{ij}) = 1$ where j will also be a part of T. If j is a part of T, then $\Sigma(X_{ij}) = 1$ where i will also be a part of S.

In this survey paper, the algorithms proposed by the following people will be discussed briefly:
Section 2: H. W. Kuhn [1]
Section 3: W. Lipski Jr, F.P. Preparata [3]
Section 4: Egon Balas, Donald Miller, Joseph Miller, Paolo Toth [4]
Section 5: James Munkres [2]

2. Hungarian Method for the Assignment Problem [1]

2.1 Problem Representation

Let n be the number of jobs such that j extends until n, individuals be n such that i extends until n and the qualification matrix be Q = q_{ij} where q_{ij} will be 1 when i individual is qualified to do j job, else q_{ij} will be 0. The assignment is said to be incomplete when unassigned qualified individuals are assigned to unassigned job after a certain qualified individuals are assigned to given jobs; else the assignment is complete. The complete assignment can be improved by a method of transfer where assignment of a specific set of individuals who are in jobs are moved to unassigned jobs. The individuals tangled in such transfer are called essential individuals and job which is assigned to them is called essential job.

Lemma 1: Either an individual or a job becomes essential when a job is assigned to an individual, but not both.

Lemma 2: Assigned individual becomes essential when it is qualified for another unassigned job.

Lemma 3: The job is essential when job is assigned for every transfer.

Proof: Assuming job j_k is inessential and i_k is assigned j_{k-1} and transfers to reducing order of k, then j_k always is unassigned.

The key results established by these lemmas are as follows:

Theorem 1: If every transfer gives complete assignment, then either individual or job becomes essential or maybe both.

Proof: Let i be the individual who is qualified to do the job labeled j, then according to Lemma 1 either one of them is essential. According to Lemma 2, i becomes essential when i is assigned another job. According to Lemma 3, j becomes essential when transfers leave j assigned and i unassigned. Hence proving the theorem.

Theorem 2: Total budget allotted is not less than the jobs assigned to qualified individuals.

Proof: In optimal assignment if a part of ample budget is given, then it will not be less than assigned jobs since these are assigned to specific individuals who are labeled as qualified. The theorem is proved because the overall budget is more than this budget.

2.2 Algorithm

Step 1: In the matrix, find the least value in every row and every column and subtract that minimum value with other values in all the rows and columns respectively.

Step 2: In the matrix, give the number of lines which have zeroes in it.

Step 3: For number of lines < n, do

> 3.1 Query for minimum value
> 3.2 Subtract the minimum value found from other values present
> 3.3 At intersection of lines, add the minimum value
> 3.4 Mark lines again covering all zeroes

Step 4: From each row and column, calculate the number of zeroes present.

Step 5: For allocations < n, do

> 5.1 Take allocation as random element and remove its row and column
> 5.2 Query the count of zeroes in rows and columns without adding the values from crossed out rows and columns

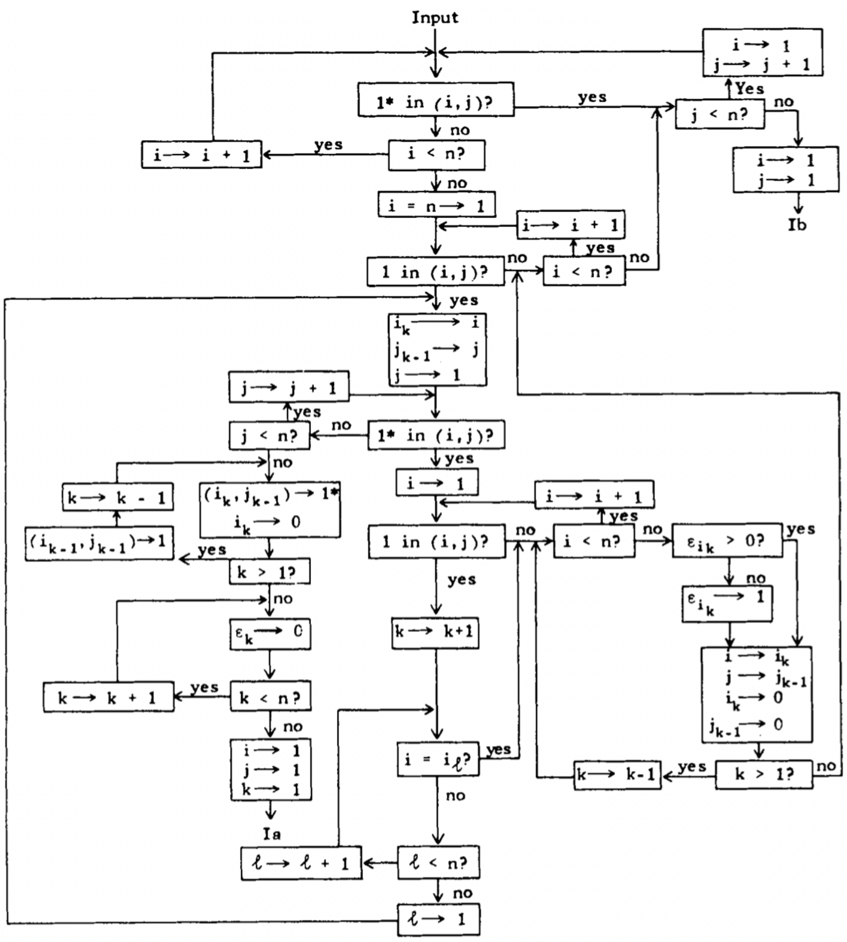


Figure 1: Flow diagram for the above-mentioned routine

Time Complexity: $O(n^3)$
The complexity is due to the nested for loops in the program which run until n.

3. Algorithm for Maximum Matchings in Convex Bipartite Graphs [3]

3.1 Definitions

Let G be an undirected bipartite graph G = (A, B, E) where vertices are A, B and E is set of edges represented in the form of (a, b) where a is in A and b is in B.
Matching: A subset M is called matching when no two edges in the subset M are incident to the same vertex.
Maximum Cardinality: The subset matching M is said to have maximum cardinality when it contains the maximum number of edges.

Lemma 1: There exists a maximum matching which involves (a,b) if it belongs to E and for all c in B(a) and A(b) is a subset of A(c)
Proof: In proof by negation, we consider maximum matching M which does not contain (a,b). When a is not matched, the edge to b is overridden with (a,b). Same happens when b is unmatched. For some c belongs to B and d belongs to A, there is (a,c), (d,b) which belongs to matching M. Since d belongs to A(b) which is a subset of A(c), we know that (d,c) belongs to E. Thus, the edges (a,b), (d,c) are overridden on the edges (a,c), (d,b).

3.2 Algorithm

The input to the algorithm would be the values n which are elements in A and m which are elements in B. The output would be the dictionary of matched values where M[j] would indicate the A vertex which is mapped to B where j belongs to B. In a convex bipartite graph, the pseudo code for detecting maximum matching is as follows:

```
begin
    for $i := 1$ to $m$ do
        begin $A_i := \emptyset$
                PRED[$i$] := $i - 1$
                SUCC[$i$] := $i + 1$
        end
    SUCC[0] := 1, PRED[$m+1$] := $m$
    for $j := 1$ to $n$ do $A_{\mathrm{BEG}[j]} := A_{\mathrm{BEG}[j]} \cup \{j\}$
    for $j := 1$ to $n$ do (* find vertex to be matched to $j$ *)
        begin $i :=$ FIND($j$)
            if $i \leq$ END[$j$] then MATCH[$j$] := $i$
                        else MATCH[$j$] := $\Lambda$ (* $i$ unmatched *)
            UNION($i$, SUCC[$i$], SUCC[$i$])
            SUCC[PRED[$i$]] := SUCC[$i$]
            PRED[SUCC[$i$]] := PRED[$i$]
        end
end
```

Figure 2: Pseudo Code for finding maximum matching

Time Complexity : O(m+nA(n))
The loops take O(m+n) steps and the other loop takes O(nA(n)) steps to find union operations.

3.3 Generalization of Convex Bipartite Graphs on Tree-Ordered Set

The above written code can be broadened to form directed tree in which elements in A form the paths. The tree is directed with A elements as vertices and for every b belongs to B, the directed path is A(b). The tree degenerates to a single path in the convex case. The directed tree is represented as S which is an array with A as the vertex set. This gives successor as S[a] where a is any vertex from the vertex set A. For convex case, A(b) which can be written by <BEG(b), END(b)>. Here set of vertices from the tree is written as A(b) which begins as BEG(b) and ends at END(b). A topological ordering of A can be produced from array S in O(m) time. The nonincreasing distance to root of a vertex can be produced from linear ordering of vertex set A elements.

4. Shortest Augmenting Path Algorithm for Assignment Problem [4]

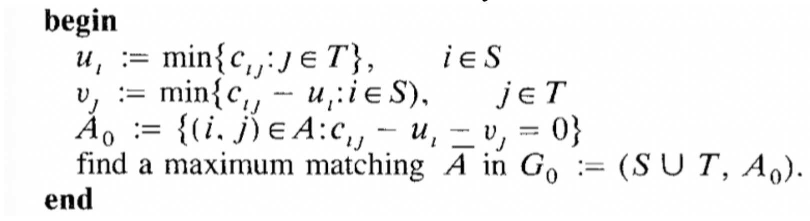
4.1 Problem Definition

The assignment problem has the goal of reducing total assignment cost after assigning n number of people to n number of tasks where single task can be done by a single person. In shortest augmenting path algorithm, changes are not required on the initial partial assignment but when several augmenting paths are used in parallel then it requires to recalculate new dual variables. The cost needs to be reduced when augmented trees are in parallel.

4.2 Algorithm and Analysis

The pseudo code for the procedure of finding an optimal solution in the subgraph with partial assignment by finding the maximum cardinality matching in the subgraph is written in 3 phases as follows:

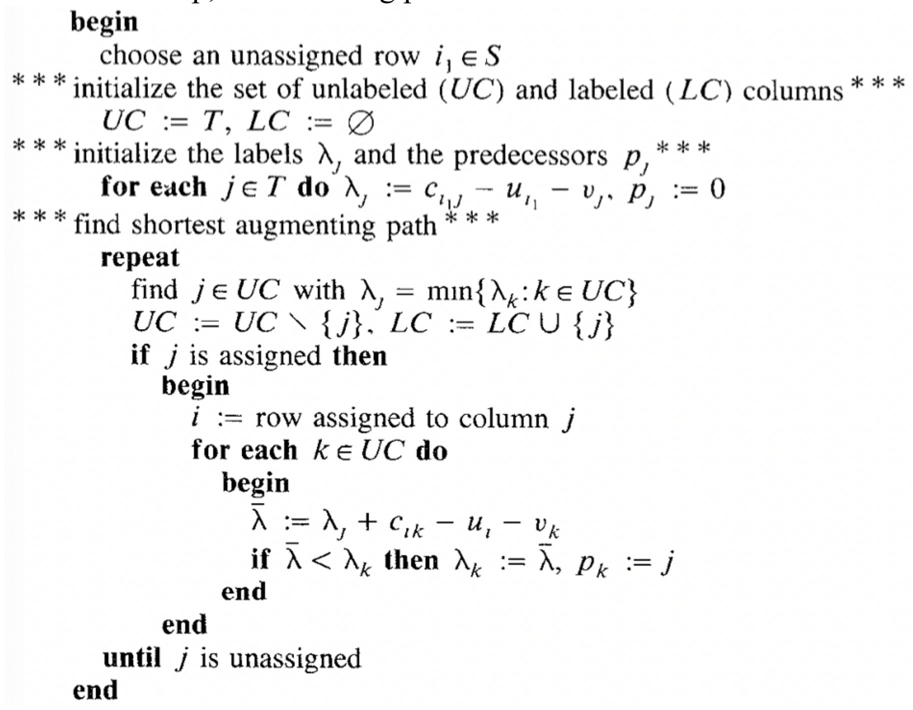
Phase 0: This is the initial solution where the assigned nodes give the optimal partial assignment in the subgraph. Firstly, an achievable answer is drawn and in the subgraph which has arcs with reduced cost as zero, the maximum cardinality match is found.



```
begin
  $u_i := \min\{c_{ij} : j \in T\}, \quad i \in S$
  $v_j := \min\{c_{ij} - u_i : i \in S\}, \quad j \in T$
  $A_0 := \{(i, j) \in A : c_{ij} - u_i - v_j = 0\}$
  find a maximum matching $\bar{A}$ in $G_0 := (S \cup T, A_0)$.
end
```

Figure 3: Phase 0 for finding optimal solution in subgraph pseudo code

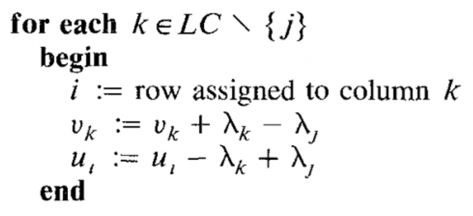
Phase 1: Next step involves finding augmenting path where shortest augmenting path is to be found from an unassigned node. The procedure modifies Dijkstra's algorithm to find node successors. In a while loop, the following pseudo code is written.



```
begin
  choose an unassigned row $i_1 \in S$
*** initialize the set of unlabeled ($UC$) and labeled ($LC$) columns ***
  $UC := T, \ LC := \emptyset$
*** initialize the labels $\lambda_j$ and the predecessors $p_j$ ***
  for each $j \in T$ do $\lambda_j := c_{i_1 j} - u_{i_1} - v_j, \ p_j := 0$
*** find shortest augmenting path ***
  repeat
    find $j \in UC$ with $\lambda_j = \min\{\lambda_k : k \in UC\}$
    $UC := UC \setminus \{j\}, \ LC := LC \cup \{j\}$
    if $j$ is assigned then
      begin
        $i$ := row assigned to column $j$
        for each $k \in UC$ do
          begin
            $\bar{\lambda} := \lambda_j + c_{ik} - u_i - v_k$
            if $\bar{\lambda} < \lambda_k$ then $\lambda_k := \bar{\lambda}, \ p_k := j$
          end
      end
  until $j$ is unassigned
end
```

Figure 4: Phase 1 for finding optimal solution in subgraph pseudo code

Phase 2: This step involves recalculating the dual variables which are generated from the above phase in the primal assignment. The new assignment is generated from the augmenting path where it matches few pairs from the previous. The end point for the algorithm is when pairing is completed for all the nodes.



```
for each $k \in LC \setminus \{j\}$
  begin
    $i$ := row assigned to column $k$
    $v_k := v_k + \lambda_k - \lambda_j$
    $u_i := u_i - \lambda_k + \lambda_j$
  end
```

```
        $u_{i_1} := u_{i_1} + \lambda_j$
    * * * update current assignment * * *
        while $p_j \neq 0$ do
          begin
            $i$ := row assigned to column $p_j$
            $\bar{A} := \bar{A} \cup \{(i, j)\} \setminus \{(i, p_j)\}$
            $j := p_j$
          end
        $\bar{A} := \bar{A} \cup \{(i_1, j)\}$
      end
```

Figure 5: Phase 2 for finding optimal solution in subgraph pseudo code

Time Complexity: $O(n^3)$

The phase 0 is completed in O(nz) time in which z denotes arcs with zero reduced cost. The phase 1 has nested for loop in a while loop which runs until n, so it takes $O(n^3)$ time. The phase 2 has a for loop and a while loop.

5. Algorithm for Assignment Problems [2]

In this method proposed by James Munkres [2], the markings for coverings of the elements of the matrix are done depending on the number of covered lines by the element, whereas in the Hungarian method[1], at step 2 a procedure to cover all zeros by a set of lines should be calculated from the matrix.

5.1 Algorithm

Initially, no lines are covered, and no zeros are starred or primed.

Step 1: In a matrix, take a row A and subtract each element in this row with the minimum of this row. Subtract each element of each column with the minimum of that column.

Step 2: Consider a zero Z in the matrix. Z is starred if in its row and column, there is no starred Z present.

Step 3: Cover each column containing a starred zero. The process is terminated when all columns are covered. The starred zeros indicate the allocations.

Step 4: Choose a row containing non-covered zero and prime it. If this row does not contain a starred zero Z, then go-to next step, else cover the row having the starred zero Z and uncover the column of Z. Repeat this step until all zeros are covered and note the minimum value.

Step 5: A series of alternating starred and primed zeros should be constructed like:

5.1 Z0 = uncovered primed zero found in previous step
Z1 = starred zero in the column of Z0
Z2 = primed zero in row of Z1

5.2 Repeat the process until there is a primed zero in which a starred zero in its column is not present.

5.3 Star all the prime zeros and unstar all the starred zeros

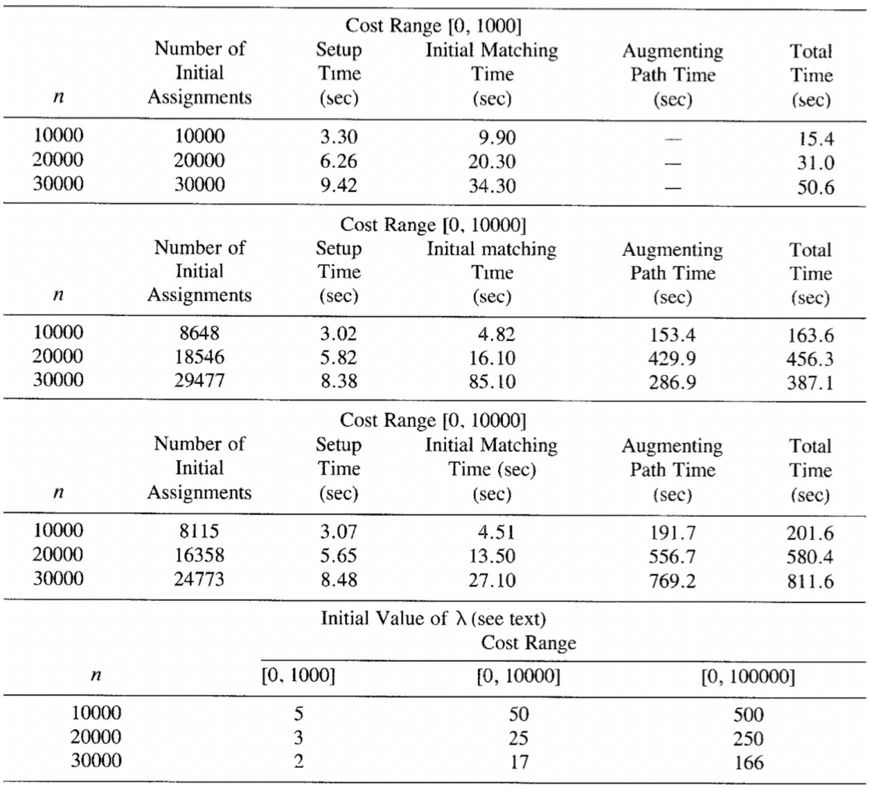
5.4 Remove all primes and uncover all lines in the matrix

5.5 Go to step 3

Step 6: For every covered row, add the value of step 4 to all elements and for every covered column, subtract the value of step 4 from all the elements. Then go back to step 4.

6. Results

Assignment Problem: [4] In C language, the algorithm for shortest augmenting path was implemented and executed on 14 processor computer. On the graphs, after solving fully dense assignment problems, summary of results of different node sizes are recorded in the below table which shows that all values are less than the threshold λ.



Cost Range [0, 1000]

n	Number of Initial Assignments	Setup Time (sec)	Initial Matching Time (sec)	Augmenting Path Time (sec)	Total Time (sec)
10000	10000	3.30	9.90	—	15.4
20000	20000	6.26	20.30	—	31.0
30000	30000	9.42	34.30	—	50.6

Cost Range [0, 10000]

n	Number of Initial Assignments	Setup Time (sec)	Initial matching Time (sec)	Augmenting Path Time (sec)	Total Time (sec)
10000	8648	3.02	4.82	153.4	163.6
20000	18546	5.82	16.10	429.9	456.3
30000	29477	8.38	85.10	286.9	387.1

Cost Range [0, 10000]

n	Number of Initial Assignments	Setup Time (sec)	Initial Matching Time (sec) (sec)	Augmenting Path Time (sec)	Total Time (sec)
10000	8115	3.07	4.51	191.7	201.6
20000	16358	5.65	13.50	556.7	580.4
30000	24773	8.48	27.10	769.2	811.6

Initial Value of λ (see text)

n	Cost Range [0, 1000]	[0, 10000]	[0, 100000]
10000	5	50	500
20000	3	25	250
30000	2	17	166

Figure 6: Arbitrarily Generated Assignment Problems

Maximum matching problem: [3] Convex Bipartite Graph algorithm applied on simple chessboard which is a finite collection on a plane with integer coordinates. The chessboard is cut vertically making some nonadjacent squares. The chessboard rows are numbered lexicographically, and columns are numbered by its abscissa. Then, a bipartite graph G is defined where edge E a and b if column a and row b intersect, thus having a common square. Matching in G gives nonattacking rooks and maximum cardinality set on this can be obtained by the discussed algorithm in linear time.

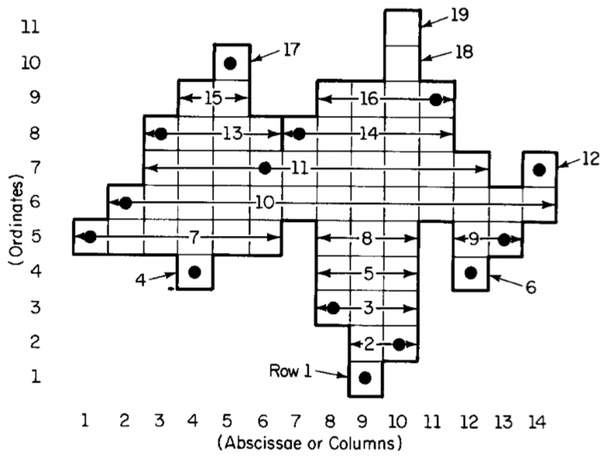


Figure 7: Numbered chessboard with nonattacking rooks from the algorithm

7. Applications

The assignment problem aims to find an optimal combination of assigning n persons to n number of tasks in a way that the assigned total cost should be minimum. The maximum matching problem is where maximum cardinality graph should be obtained. These problems have various direct and indirect applications and a few of them are discussed below:

7.1 Asymmetric TSP

On a directed graph, one of the most commonly known assignment is asymmetric traveling salesman problem [4]. The assignment problem can be obtained by relaxation after clearing subtour elimination restrictions from integer programming formulation of ATSP. Through relaxation for arbitrarily generated costs, the optimal assignment value decreases with size of the problem. Hence, manageable size of search trees can be obtained with random costs by applying brand and bound to ATSP.

7.2 Chain decomposition of poset

One of the applications of maximum matching problem of partially ordered set is chain decomposition [3]. The correspondence between both problems allows to apply such efficient techniques which are computationally proper to calculate the minimum service aircraft number in various routes. Furthermore, possible types of schedules can be obtained from a bipartite graph by counting maximum matchings.

8. References

[1] Kuhn, Harold W. "The Hungarian method for the assignment problem." Naval research logistics quarterly 2.1-2 (1955): 83-97

[2] Munkres, James. "Algorithms for the assignment and transportation problems." Journal of the society for industrial and applied mathematics 5.1 (1957): 32-38.

[3] W. Lipski Jr, F.P. Preparata. "Efficient Algorithms for Finding Maximum Matchings in Convex Bipartite Graphs and Related Problems."Acta Informatica 15, 329-346 (1981)

[4] Egon Balas, Donald Miller, Joseph Pekny, Paolo Toth. "A Parallel Shortest Augmenting Path Algorithm for the Assignment Problem."