Addis Ababa University

School of Graduate Studies

Faculty of computer and mathematical Science

Department of Mathematics

Project

On

Minimum cost Maximum cardinality matching problem

By: Addis Messay

Advisor: Berhanu Guta (PhD)

A Project Submitted to the Office of Graduate Programs of

Addis Ababa University in Partial fulfillment of the requirements for the Degree of Master of Science in mathematics

January, 2012

AA, Ethiopia

ABSTRACT

The complete set of this paper focuses on maximum cardinality matching and minimum cost maximum cardinality matching problem with their algorithms which solve cardinality matching in a graph $G = (N, A)$ where N is set of n nodes and A is set of m arcs. I have discussed cardinality matching problem by classifying into two versions called bipartite cardinality matching and nonbipartite cardinality matching problem.

ACKNOWLEGEMENT

First of all I would like to praise my Lord, who gave me everything I need and brought me from nothing to here. I also like to express my gratitude to Dr. Berhanu Guta, my advisor for this master project, who devoted a great part of his valuable time and give me very important supports and suggestions for this work to be successfully done. I would like to extend grateful acknowledgement to my family who support me all the time.

Thank you very much

Addis Messay

CONTENT

Title **Page**

ABSTRACT

The complete set of this paper focuses on maximum cardinality matching and minimum cost maximum cardinality matching problem with their algorithms which solve cardinality matching in a graph $G = (N, A)$ where N is set of n nodes and A is set of m arcs. I have discussed cardinality matching problem by classifying into two versions called bipartite cardinality matching and nonbipartite cardinality matching problem.

ACKNOWLEGEMENT

First of all I would like to praise my Lord, who gave me everything I need and brought me from nothing to here. I also like to express my gratitude to Dr. Berhanu Guta, my advisor for this master project, who devoted a great part of his valuable time and give me very important supports and suggestions for this work to be successfully done. I would like to extend grateful acknowledgement to my family who support me all the time.

Thank you very much

Addis Messay

Introduction

Matching problem is one of a particular class of combinatorial (or discrete) optimization problems; this problem illustrates the flow of ideas from network flows to other area of discrete optimization. Matching's in general graphs having long been a subject of investigation in both operations research and classical combinatorial analysis, although with rather different terminology and different motivations by investigators. In general terms, matching is a subgraph with the property that every node in the subgraph has degree zero or one. Cardinality matching problem is one version of matching problem in which the objective is to find a matching containing the maximum number of arcs. But in the minimum cost maximum cardinality matching problem, the objective is to find a maximum cardinality matching with a minimum cost among all maximum cardinality matchings. This problem arises in many different practical problems since we often wish to find the best way to pair objects or people together to achieve some desired goal.

This paper contains three chapters. In the first chapter, we discussed on some basic ideas of network flow and the theory of matching. In the second chapter, we briefly describe maximum cardinality matching problem on bipartite and nonbipartite graphs and the algorithms solving bipartite and nonbipartite cardinality matching problem with their examples. And in the last chapter we study on the minimum cost perfect matching problem on the general graph. It includes the general problem models (primal and dual) and the algorithm which solve it.

CHAPTER ONE

1. PRELIMINARY CONCEPT

1.1 BASIC DIFINITIONS

In this section, we introduce some of the basic definitions relating to graphs, paths, flows, and other related notions.

Directed Graphs: A directed graph G = (N, A) consists of a set N of nodes and a set A of arcs whose elements are ordered pairs of distinct nodes. An arc $(i, j) \in A$ is incident to node i and node j. The arc (i, j) is an outgoing arc to node i and an incoming arc to node j. whenever an arc $(i, j) \in A$, we say that node j is adjacent to node i. The numbers of nodes and arcs are denoted by n and m, respectively, and it is assumed throughout that $1 \leq N < \infty$ and $0 \leq A < \infty$.

Undirected Graph: We define an undirected graph in the same manner as we define a directed graph except that arcs are unordered pairs of distinct nodes.

Source node: is a node which has only outgoing arrows or edges.

Sink node: is a node which has only incoming arrows or edges.

Network: is a directed or undirected graph whose nodes and/or arcs have associated numerical values.

Degree: The indegree of a node is the number of incoming arcs of that node and its outdegree is the number of its outgoing arcs. The degree of a node is the sum of its indegree and outdegree.

Adjacency List:

Arc adjacency list: the arc adjacency list $A(i)$ of a node i is the set of arcs emanating from that node, that is, $A(i) = \{(i, j) \in A : j \in N\}$.

Node adjacency list: the node adjacency list $A(i)$ is the set of nodes adjacent to that node; in this case, $A(i) = \{j \in N : (i, j) \in A\}$.

Subgraph: A graph G' = (N', A') is a subgraph of G = (N, A) if N' $\subseteq$ N and A' $\subseteq$ A.

Path: A path $P = i_1 - i_2 - \ldots - i_r$ in a graph G = (N, A) is the sequence of adjacent arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{r-1}, i_r)$ were no arcs and also no intermediate node is repeated.

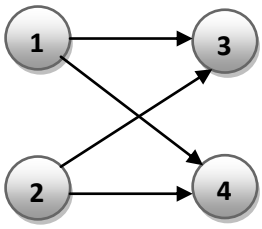
Cycle: A cycle is a path $i_1 - i_2 - \cdots - i_r$ together with the arc(i_r, i_1) or (i_1, i_r).

Connected Graph: we will say that two nodes i and j are connected if the graph contains at least one path from node i to node j. A graph is connected if every pair of its nodes is connected; otherwise, the graph is disconnected.

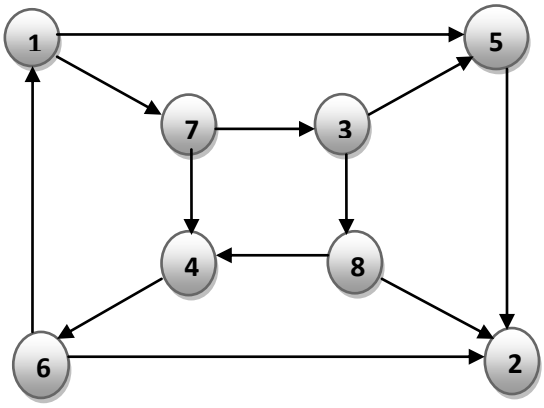
Tree: A tree is a connected graph that contains no cycle.

Rooted Tree: A rooted tree is a tree with a specially designated node, called its root; we regard a rooted tree as though it were hanging from its root.

Bipartite Graph: A graph G = (N, A) is a bipartite graph if we can partition its node set in to two subsets N_1 and N_2 so that for each arc (i, j) in A either (1) $i \in N_1$ and $j \in N_2$ or(2) $i \in N_2$ and $j \in N_1$. Therefore, for some clarity we can write G = (N, A) as $G = (N_1 \cup N_2, A)$ for bipartite graph. $Figure$ 1.1 gives two examples of bipartite graphs. $In\ figure, 1.1(a)$ $N_1 = \{1,2\}$ and $N_2 = \{3,4\}$. $In\ figure$ $1.1(b)$ $N_1 = \{1,2,3,4\}$ and $N_2 = \{5,6,7,8\}$.



(a)



(b)

Figure 1.1 Examples of bipartite graphs.

Flow: A flow of an arc is a variable that measures the quantity flowing through each arc. The flow of an arc (i, j) is simply a scalar (real number), which we usually denote by x_{ij}.

Flow vector: Given a graph $G = (N, A)$, a set of flows $X = \{x_{ij}/(i,j) \in A\}$ is referred to as a flow vector. A flow vector that corresponds to sending a positive amount of flow along a path is called path flow

Conformal flow Decomposition: It says that a nonzero flow vector X can be decomposed into the sum of k path flow vectors $x_1, x_2, \dots, x_k$ that conform to X, with k being at most equal to the sum of the numbers of arcs and nodes $n + m$. If X is integer, then $x_1, x_2, \dots, x_k$ can also be chosen to be integer.

1.2 Matching

In this section we will describe what matching means and introduce some notations related to matching. We also describe two broad versions of matching problems called bipartite and nonbipartite matching problem.

What is matching?

Let G = (N, A) be a graph.

Definition: A matching M is a subgraph of a graph G, with the property that every node in the subgraph has degree zero or one. That is, no two arcs in the subgraph are incident to the same node.

If all nodes of a graph are incident to arcs in the matching M, then the matching M is called a perfect matching.

Figure 1.2 illustrates these definitions. The arcs $\{(s,1), (2,5), (6,t)\}$ constitute a matching in the graph; we use thicker lines to the arcs in the matching.

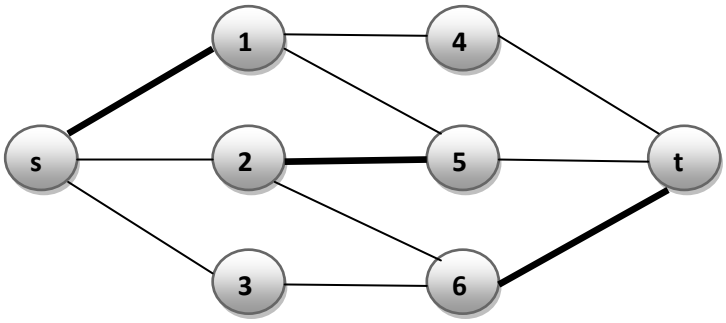


Figure 1.2 matching example

Note that: A matching can contain at most $\lfloor n/2 \rfloor$ arcs, where n is number of nodes in the graph. Since, in a matching M each node has at most degree of one.

Versions of Matching Problems

In this project we are interested only on the matching problem defined on bipartite and nonbipartite graph, which is related to find maximum cardinality matching, and minimum cost (weight) perfect matching problem.

First, we begin by examining bipartite matching problems. For the time being consider two versions of these problems:

(1) The cardinality matching problem in which we wish to find a matching containing the maximum number of arcs. That is, we wish to find maximum cardinality matching.
(2) The weighted matching problem in which we have a weight associated with each arcs and we wish to find a matching with the largest (the minimum) overall weight.
Note that for the cardinality matching problem, the weights associated to each arcs are one.

In case of nonbipartite matching problems, we also have the same problem versions as bipartite matching problems. But as we said earlier, in this paper we are focused on the cardinality versions of bipartite and nonbipartite matching problem mainly on the algorithms to find optimal matching, and on the weighted perfect matching problem.

1.2.1 Matching Terminologies and Notations

Let $A(i)$ denote the node adjacency list of node i; that is, $A(i) = \{j \in N: (i,j) \in A\}$ where G = (N, A) is a graph.

Matched Arcs and Matched Nodes

Let M be a matching of a graph G = (N, A). We refer to the arcs in M as matched arcs, and arcs not in M as unmatched arcs. We also refer to the nodes incident to matched arcs as matched

nodes and refer to the other nodes as unmatched. If (i, j) belongs to the matching M, we say that node i is matched to node j and node j is matched to node i.

Figure 1.3 illustrates these definitions. The arcs $\{(2,4), (3,5)\}$ constitute a matching in the given graph; we depict matched arcs using thicker lines.

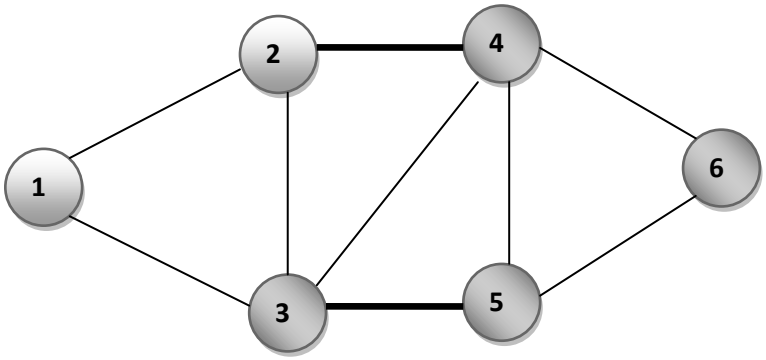


Figure 1.3 matching examples.

Alternating Paths and Cycles

We refer to a path $P = i_1 - i_2 - \cdots - i_k$ in the graph as an alternating path with respect to a matching M if every consecutive pair of arcs in the path contains one matched and one unmatched arc. In figure 1.3, paths $1-2-4-3-5$ and $1-2-4-3-5-6$ are alternating paths.

Even alternating paths: we refer an alternating path as an even alternating path if it contains an even number of arcs. For example in figure 1.3, an alternating path $1-2-4-3-5$ is an even alternating path.

Odd alternating paths: we refer an alternating path as an odd alternating path if it contains an odd number of arcs. For example in figure 1.3, an alternating path $1-2-4-3-5-6$ is an odd alternating path.

An alternating cycle is an alternating path that starts and ends at the same node. For example in figure 1.3, $3-2-4-5-3$ is an alternating cycle.

Alternating Trees

We refer to a tree T in the graph as an alternating tree with respect to a matching M if each path between any two nodes in the tree is alternating path.

Augmenting Paths

We refer to an odd alternating path P with respect to a matching M as an augmenting path if the first and last nods in the path are unmatched. We use the terminology augmenting path because by redesignating matched arcs on the path as unmatched and unmatched arcs as matched, we obtain another matching of cardinality $|M| + 1$.

Example: in figure 1.3 an odd alternating path $1 - 2 - 4 - 3 - 5 - 6$ is an augmenting path with respect to a matching of cardinality 2, and if we interchange the matched and unmatched arcs on this path, we obtain the matching of cardinality 3 shown in figure 1.4.

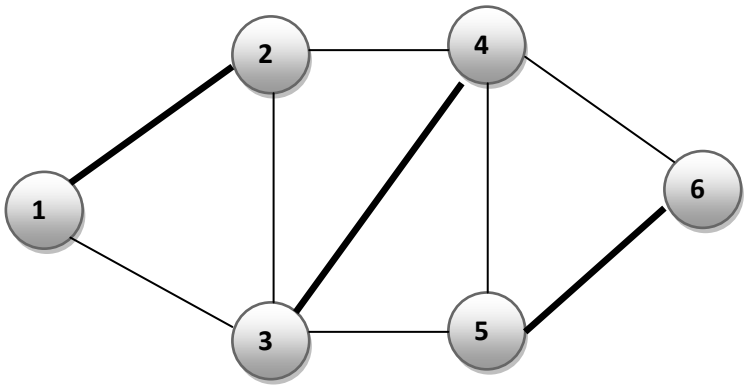


Figure 1.4 matching a larger cardinality then the matching in figure 1.3.

Symmetric Difference

The concept of symmetric difference of sets is quite important in matching theory. Let s_1 and s_2 be two sets.

The symmetric difference of sets s_1 and s_2, denoted $s_1 \oplus s_2$, is the set $s_1 \oplus s_2 = (s_1 \cup s_2) - (s_1 \cap s_2)$. In other words, it is the set of elements that are members of $s_1 or\ s_2$, but not members of both $s_1\ and\ s_2$.

Example:

If $s_1 = \{1,2,3,4\}$ and $s_2 = \{3,4,5,6\}$, then $s_1 \oplus s_2 = \{1,2,5,6\}$.

Now, let us see two important properties of symmetric difference in the context of matching.

Property 1.1 If M is a matching and P is an augmenting path with respect to M, then $M \oplus P$ is a matching of cardinality $|M| + 1$.

It is better to describe this property with an example. Consider a graph given in figure 1.4.

Given a matching $M = \{(1,5), (2,6), (8,3)\}$

of cardinality 3 and an augmenting path P

with respect to M is: $P = \{(4,8), (8,3), (3,7)\}$

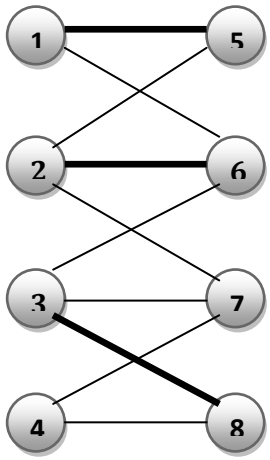


Figure 1.4 a given graph with a matching M.

Then, $M \oplus P = (M \cup P) - (M \cap P)$

$= \{(1,5),(2,6),(4,8),(3,7)\}$

This implies, $|M \oplus P| = 4 = |M| + 1$

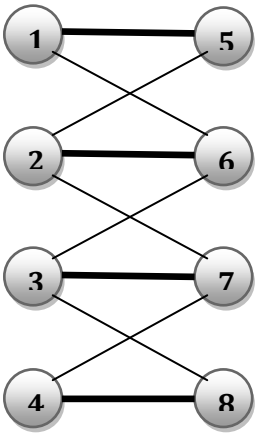


Figure 1.4 a graph with matching $M \oplus P$.

Therefore, the symmetric difference of a matching M and augmenting path P with respect to M $M \oplus P$ is a matching of cardinality $|M| + 1$. Observe that in the matching $M \oplus P$, all the matched nodes {1, 2, 3, 5, 6, 8} remain matched and two additional nodes {4, 7}, namely the first and last nodes of P, are matched.

Property 1.2 If M is a matching and P is an augmenting path with respect to M, then in the matching $M \oplus P$ all the matched nodes in M remain matched and two additional nodes, namely the first and last nodes of P, are matched.

Augmentation: the symmetric difference of the matching M with the augmenting path P is a set-theoretic way to interchange the matched and unmatched arcs in P. we refer to the process of replacing M by $M \oplus P$ as an augmentation.

CHAPTER TWO

2. Cardinality Matching Problem

In this section we study the cardinality matching problem on undirected graphs, in which we wish to identify a matching of maximum cardinality. It is one type of matching problem with the property that the weights associated to each arcs are one. A matching M of a graph $G = (N, A)$ is called maximum cardinality matching, if there is no other matching M^*, such that $|M^*| > |M|$. Note that maximum cardinality matching of a graph G may not be unique.

Generally, we can classify cardinality matching problem in to two versions based on the graph which defined on, namely bipartite cardinality matching problem and nonbipartite cardinality matching problem. In this paper we try to discuss on both of these problems mainly on the algorithms.

Before discussing the algorithm, let us state and prove an important theorem called augmenting path theorem which helps to obtain an optimal matching.

2.1 Augmenting path theorem

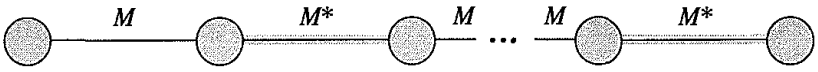
Our algorithm which we will discuss later for the cardinality matching problem depends crucially on the augmenting path theorem. Before we state and prove augmenting path theorem, we shall see the following lemma.

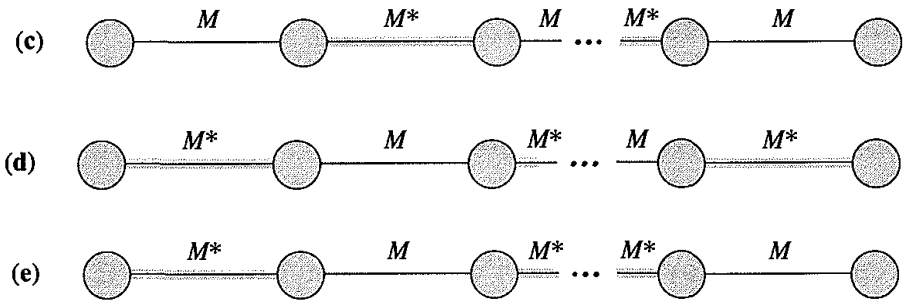
Lemma 2.1 let $G = (N, A)$ be a simple graph. If M and M^* are two matching of G, their symmetric difference defines the subgraph $G^* = (N, M \oplus M^*)$ with the property that every connected component is one of the three types shown in figure 2.1.

Type 1: Singleton nodes

(a) 

Type 2: Paths

(b) 



Type 3: Even-length cycles

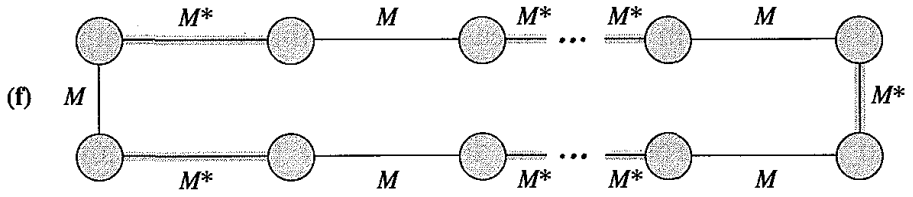


Figure 2.1 possible types of components formed by a symmetric difference of two matching M and M^*.

Proof

Let $i \in N$

In the subgraph $G^* = (N, M \oplus M^*)$; for all $i \in N$; $\deg(i) \leq 2$. Since M and M^* are matching's, there cannot be more than one arc of M and more than one edge of M^* incident with each node i of N.

Therefore, the only possible connected components with node degrees 0, 1 or 2 are given in figure 2.1. Hence the result.

Theorem 2.1 (Augmenting path theorem) A matching M is maximum matching if and only if there is no augmenting path with respect to M.

Proof

(a) Forward implication (necessary condition):

Let M be a maximum matching.

We prove it by contradiction

Suppose that there exist an augmenting path P with respect to M.

By property 1.1 we can obtain a matching $M^* = M \oplus P$ of cardinality

$$|M^*| = |M| + 1.$$

Hence the result.

(b) Backward implication (sufficient condition):

Let M be a matching such that there is no augmenting path relative to M.

Let M^* be a maximum matching.

This implies that, there cannot be an augmenting path relative to M^* (by forward implication of this theorem).

By lemma 2.1, the subgraph $G^* = (N, M \oplus M^*)$ may have one or more of the three types of components. But $G^* = (N, M \oplus M^*)$ has no odd components of type 2. Since M and M^* contains no augmenting path.

Therefore, $|M - M^*| = |M^* - M|$.

Hence $|M| = |M^*|$ which proves that M is a maximum matching.

Next, we state and prove an alternative version of augmenting path theorem as a corollary.

Corollary 2.1 If a node i is unmatched in a matching M, and this matching contains no augmenting path that starts at node i, then node i is unmatched in some maximum matching.

Proof

Given a matching M and a node i is unmatched in M and also a matching M contains no augmenting path that starts at node i.

Let M^* be a maximum matching

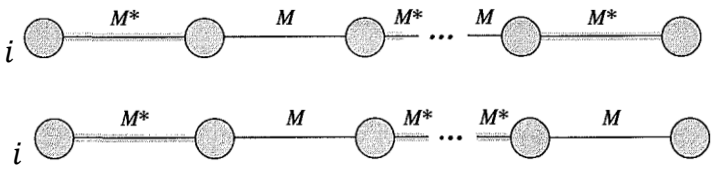
If node i is unmatched in M^*,

Then node i is unmatched in a maximum matching M^*. Hence the result.

If node i is matched in M^*,

Consider the matching $M \oplus M^*$.

By lemma 2.1 and node i is unmatched in M, the only possible components of a matching $M \oplus M^*$ are paths with node i as the starting node.



But, there is no augmenting path that starts at node i. Since M^* is maximum matching. So that the first possibility is not happened.

Therefor, the only remaining possibility is the second one which is even alternating path P with node i as the starting node.

Hence, a matching $M' = M^* \oplus P$ is also a maximum matching in which node i is unmatched. Hence the result.

2.2 Bipartite Cardinality Matching Problem

As we defined earlier, a graph $G = (N, A)$ is bipartite if its set of nodes N can be partitioned into two sets say N_1 and N_2 such that every arc in A has one endpoints in N_1 and the other in N_2. Bipartite cardinality matching problem is a cardinality matching problem which is defined on bipartite graph. It is easy to solve than nonbipartite cardinality matching problem because we can transform it into maximum flow problem.

2.2.1 Problem Reduction and Equvalence

One important feature of bipartite matching problem is that we can reduce (or transform) it into standard network flow problems (or maximum flow problem).

Reduction of bipartite cardinality matching problem to maximum flow problem

Now, we will see the steps how bipartite cardinality matching problem transformed into a maximum flow problem in a simple network.

Consider a bipartite undirected graph $G = (N_1 \cup N_2 , A)$.

Step 1:

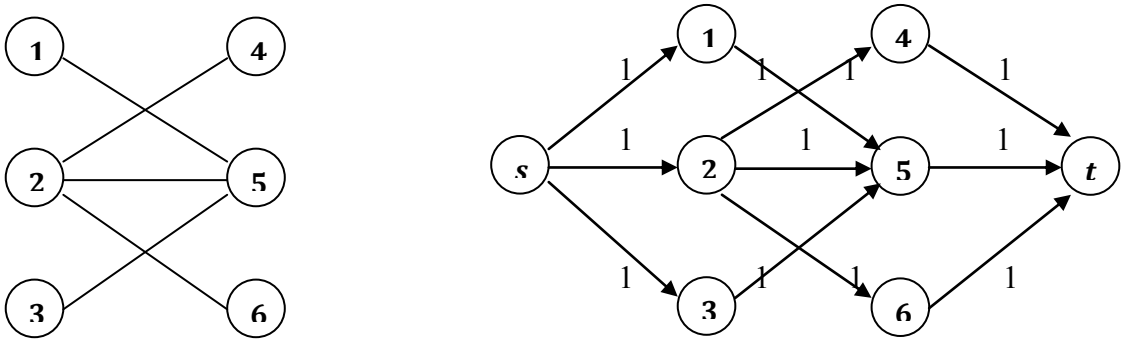
First create a directed version of the underlying graph G by designating all arcs as pointing from the nodes in N_1 to the nodes in N_2.

Step 2:

Introduce a source node s and a sink node t with an arc connecting s to each Node in N_1 and an arc connecting each node in N_2 to t. And set the capacity of each arc in the network to one.

Example:

Figure 2.2 illustrates this transformation. We refer to the transformed network as G' = (N', A'). Note that the network G' is a simple network since each arc has a unit capacity and every node in N_1 has one incoming arc and every node in N_2 has one outgoing arc. We can observe that for every bipartite cardinality matching problem on $|N_1|$ + $|N_2|$ nodes, there is a corresponding maximum flow problem in an ($|N_1|$ +$|N_2|$ + 2)-node flow network.



(a) (b)

Figure 2.2 transforming a bipartite cardinality matching problem to a maximum flow problem: (a) original network; (b) unit capacity maximum flow network.

Establishing a One to One Correspondence

To establish a one to one correspondence between a matching of cardinality K in the original network and an integral flow of value K in the transformed network:

First, consider a given matching $\{(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)\}$ of cardinality K in the original network G. Then we construct a flow in the transformed network G' as follows:

Step 1: set the flow on each of the matched arcs equal to one.

Step 2: to satisfy the mass balance constraints, set the flow on the arcs (s, i_r) and (j_r, t) equal to one for all $r = 1,2, \dots, k$.

Therefore it is clear that this choice gives us a flow of value k from node s to node t.

Second, consider a given integral flow of value K from node s to node t in the transformed network G'. Then we can specify a corresponding matching in the original network G as follows:

By flow decomposition, the integral flow of cardinality K decomposes into K paths of the form $s - i_1 - j_1 - t, s - i_2 - j_2 - t, \dots, s - i_k - j_k - t$. Since each of the arcs incident to nodes s and t have a unit capacity, no two nodes in N_1 or N_2 appear in more than one of these paths.

Therefore the K arcs $\{(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)\}$ define a matching.

We have thus established equivalence relation between matching's in the original network and integral flow in the transformed network.

2.2.2 Bipartite Cardinality Matching Algorithm

There are many algorithms which solves bipartite cardinality matching problem those algorithm can be studied based on their complexity and detailed description. But in this project we deal on the detail of one of those best algorithms rather than on their complexity.

One way of solving bipartite cardinality matching problem is that, first by transform it into maximum flow problem and then solving these maximum flow problem means solving the

original problem. Since we are already established an equivalence relation between them. However, the algorithm which we will see need not to transform the matching problem into maximum flow problem.

The augmenting path theorem suggests the following algorithm for solving the cardinality matching problem. Start with a feasible matching M (which might be a null matching) of a bipartite graph $G = (N, A)$ and then repeat the following step for every unmatched node $r \in N$. Try to identify an augmenting path starting at node r. If we find such a path P, replace M with $M \oplus P$; otherwise, delete node r and all the arcs incident to it from the graph.

At each iteration, the algorithm reduces the number of unmatched nodes by at list one, either by deleting a node or by matching it. Since matched nodes remain matched throughout the algorithm (by property 1.2), when the algorithm terminates, each node in the remaining subgraph, say G', is matched. Consequently, the matching M must be a maximum matching for G'. By corollary 2.1 which implies that the deletion of nodes does not reduce the number of arcs in a maximum cardinality matching, the matching M is also a maximum matching in G.

The main idea of this algorithm is to find whether or not the graph contains an augmenting path starting at unmatched node r.

Identifying augmenting path

To find whether or not the graph contains an augmenting path starting at unmatched node r, the algorithm first construct an alternating tree rooted at node r. In the alternating tree each path starting from node r to another node is an alternating path.

To identify augmenting path the algorithm assign label (even or odd) to nodes in the process of constructing alternating tree. All nodes in the alternating tree are called labeled nodes and the other nodes are unlabeled. Labeled nodes are of two types, namely even or odd. Node i is even or odd depending on whether the number of arcs in the unique path from the root node r to node i in the alternating tree is even or odd respectively.

Note that: whenever an unmatched node (other than the root node r) has an odd label, the path (or alternating path) joining the root node r to this node is an augmenting path.

Constructing alternating tree (or search procedure)

Search procedure is the process used to construct alternating tree. For convenience, we assign the label 'E' to even labeled nodes and the label 'O' to odd labeled nodes. In the process of constructing alternating tree the search procedure assigns label to nodes in the following manner:

It first assigns an even label to root node. Then the search procedure scans labeled nodes one by one. For even labeled node i, the search procedure scans its node adjacency list $A(i)$ and assign an odd label to every node j in $A(i)$ (provided that node j is unlabeled). On the other hand, for odd labeled node i, the search procedure scans its unique matched arc (i, j) and if node j is unlabeled, it assigns an even label to node j.

This process is terminates when all labeled nodes are scanned, or it has assigned an odd label to unmatched node, thus discovering augmenting path.

Now let us see the general algorithmic description step by step;

Step 0 (Start)

- The undirected bipartite graph $G = (N_1 \cup N_2 , A)$ is given.
- Let M be any matching, possibly the empty matching.
- No nodes are labeled

Step 1 (Labeling: constructing alternating tree and finding augmenting path)

- Choose one unmatched node $r \in N_1 \cup N_2$ and give an even label to node r and initialize $LIST = \{r\}$ where, $LIST$ is a set which store labeled nodes.
- Select and remove a node i from the set $LIST$ if $LIST \neq \emptyset$.
- If a selected node i has an even label, then scan its node adjacency list $A(i) = \{j \in (N_1 \cup N_2) : (i, j) \in A\}$. That is,

  For every node $j \in A(i)$ **do**
  - If node j is unmatched ($or\ j \in A(i)\ and\ j \in (N_1 \cup N_2) - M$), then assign an odd label to node j and set $q := j$, and $pred(q) := i$.

At this time the algorithm discover an augmenting path which terminates at node q. STOP and go to step 2.

- If node j is matched and unlabeled ($or\ j \in A(i), j \in M\ and\ j \notin LIST$), then set $pred(j) := i$ and assign an odd label to node j, and add it to $LIST$.

- If a selected node i has an odd label ;
  Let j be a unique node matched to node i, and then **do**
  - If node j is unlabeled ($or\ (i,j) \in M\ and\ j \notin LIST$), then set $pred(j) := i$ and assign an even label to node j, and add it to $LIST$.
- If $LIST = \emptyset$, STOP.

Step 2 (Augmentation)

- If an augmenting path P is found in step 1 which terminates at node j, then trace the augmenting path P by starting at node j and traversing the predecessor indices. That is: if the predecessor of j is "i", the second- to-last node in the path is i. If the predecessor of i is "k", the third-to-last node is k, and so on; until the root node r is found.
- Update the matching M using the operation $M := M \oplus P$. That is, augment a matching M by adding to M all arcs in the augmenting path that are not in M and removing from M those which are.
- Remove all labels from nodes and return to step 1.

Step 3 (Termination Criteria)

- An augmenting path P is not found in step 1, delete node r and all arcs incident to it from the graph G. Return $G := G^*$ where G^* is the new graph.
- If all nodes in the graph G are matched; therefore, the matching is maximum and STOP.

Before we see an example, let us state an important property called a unique label property.

> **Unique label property:** a graph is said to possess a unique label property with respect to a given matching M and a root node r if the search procedure which is used to construct alternating tree assigns a unique label to every labeled node (that is even or odd) irrespective of the order in which it scans labeled nodes.

This unique label property used to answer the question: when the algorithm fails to find an augmenting path, can we conclude that the network contains no such path?

If the graph possesses the unique label property, the algorithm will always discover an augmenting path if one such path exists. In other words, if the graph possesses a unique label property, the answer of the above questions will be 'yes'. To show this statement is true:

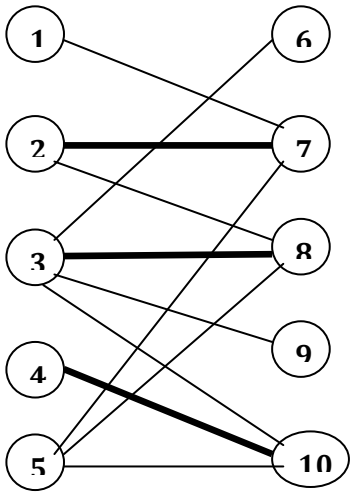
> Suppose that the network contain an augmenting path $r - i_1 - j_1 - i_2 - j_2 - \cdots - i_k - j_k - q$ from the root node r to unmatched node q with respect to the matching M.
>
> Then, if the search procedure scans the nodes $r, i_1, j_1, i_2, j_2, \ldots$ in order, the algorithm will assign even labels to nodes $r, j_1, j_2, \ldots, j_k$ and odd labels to nodes $i_1, i_2, \ldots, i_k, q$.
>
> Since the graph possesses the unique label property, the algorithm would assign the same labels no matter in which order the search procedure scans the labeled nodes.
>
> Therefore, the search procedure will always assign an odd label to node q and will discover an augmenting path.

Note that: bipartite graphs satisfy the unique label property with respect to any matching and any root node. Since for a bipartite graph $G = (N_1 \cup N_2, A)$, if the root node $r \in N_1$, every labeled node in N_1 will receive an even label and every labeled node in N_2 will receive an odd label.

Therefore, we can conclude that our bipartite cardinality matching algorithm obtains optimal matching in bipartite graph and in any graph which possesses the unique label property.

Example: Apply bipartite cardinality matching algorithm on the given bipartite graph to obtain maximum cardinality matching. Assume that the algorithm scans labeled nodes in first-in, first-out order and scans the adjacency list of any node in increasing order of the node number.

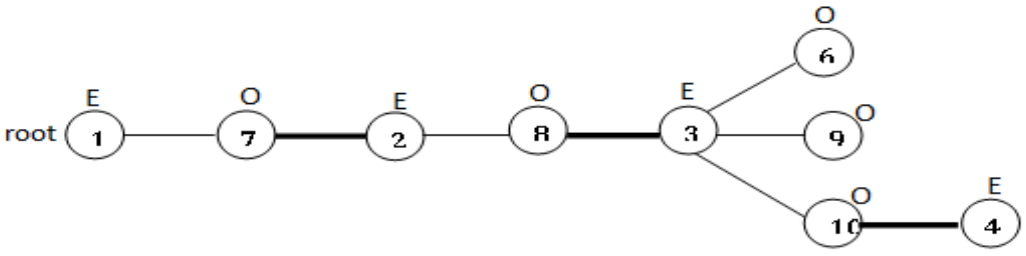
Let a matching $M = \{(2,7), (3,8), (4,10)\}$ is given.



Solution

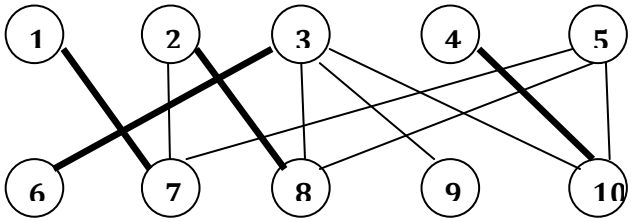
Constructing alternating tree

- Select unmatched node 1 as a root node and assign an even label to it.
- Scan the nodes in the following order;
  $1(even), 7(odd), 2(even), 8(odd), 3(even), 6(odd), 9(odd), 10(odd), 4(even)$
- When the algorithm scan node 3, it labeled unmatched node 6 and gives an odd label to it. Therefore the augmenting path $P = 1 - 7 - 2 - 8 - 3 - 6$ is identified.



Augmentation

- Update the given matching M as $M := M \oplus P = \{(1,7), (2,8), (3,6), (4,10)\}$
- Remove all labels from all nodes

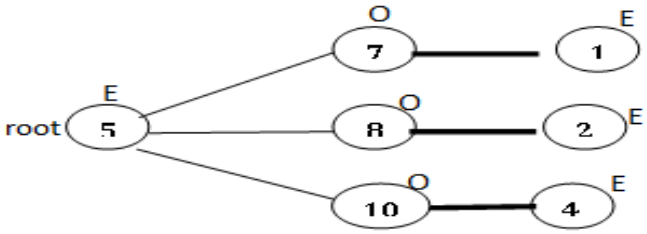


Constructing alternating tree

- Select unmatched node 5 as a root node and assign an even label to it.
- Scan the nodes in the following order;

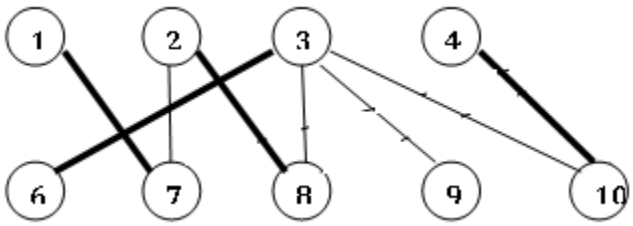
$$5(even), 7(odd), 8(odd), 10(odd), 1(even), 2(even), 4(even)$$

- Fail to identify augmenting path



Terminating criteria

- Delete node 5 and all arcs incident to it from the graph.
- Then we left with only one unmatched node 9; therefore, the matching $M = \{(1,7), (2,8), (3,6), (4,10)\}$ is maximum matching.



2.3 Nonbipartite Cardinality Matching Problem

A cardinality matching problem which is defined on nonbipartite graph is called nonbipartite cardinality matching problem. It is hard to solve than bipartite matching problem because we cannot transform it into standard network flow problem.

2.3.1 Flowers and Blossoms

In this sub-section we study on some basic concepts of nonbipartite graph which are helpful to design an algorithm.

To solve nonbipartite cardinality matching problem, just like the bipartite one the main idea is to find whether or not the graph contains an augmenting path. As we discussed in section 2.2.2, our algorithm for bipartite cardinality matching problem works on all graphs which satisfies the unique label property. But nonbipartite graphs might not satisfy the unique label property. Therefore, we have to modify our bipartite cardinality matching algorithm in order to work for nonbipartite cardinality matching problem.

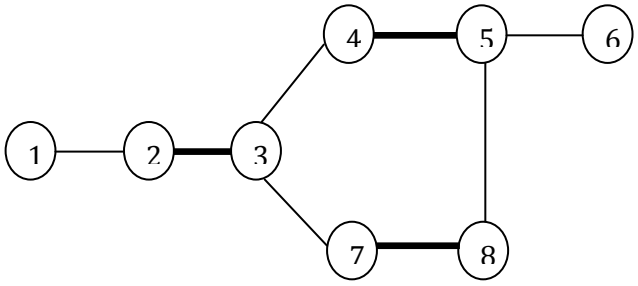
The difficulties come from the search procedure; the search procedure might fail to detect an augmenting path even though the nonbipartite graph contains one. That is, depending on the order in which the search procedure scans labeled nodes, we may get both odd-length and even-length alternating path which connect the root node and some other labeled node.

Example: Shows that, depending on the order in which the search procedure scans labeled nodes augmenting path may not be detect even though the nonbipartite graph contains one:

Given a nonbipartite graph and

a matching $M = \{(2,3), (4,5), (7,8)\}$

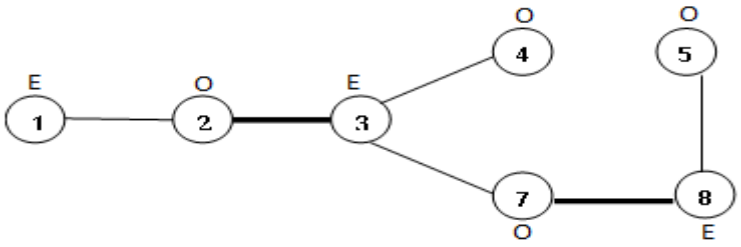
Let node 1 be root node



See the two different alternating trees with the same root node of the given graph; one has an augmenting path but not the other:-

Case 1:

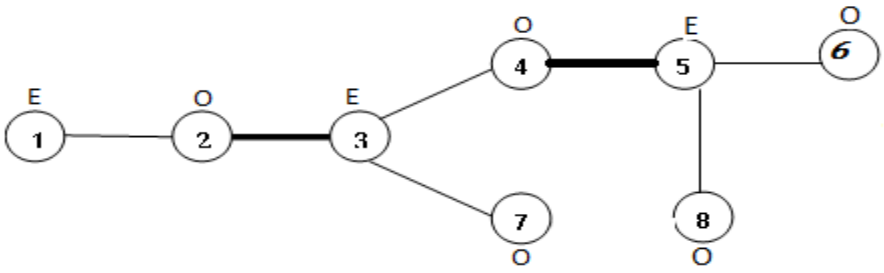
If the nodes scans in order of 1,2,3,7,8,5, and then the alternating tree is;



Observe that node 5 has an odd label and its unique matched arc is (5,4), but node 4 is already scanned. Thus the search procedure fails to discover an augmenting path.

Case 2:

If the search procedure scans the nodes in order of 1,2,3,4,5, and then the alternating tree is;



Observe that node 5 has an even label and when it is scanned, the search procedure gives an odd label to unmatched node 6. Thus the augmenting path $1-2-3-4-5-6$ is discovered.

In the above example, two different cases arise because we can connect node 5 to the root node 1 by both an odd-length $1-2-3-7-8-5$ and an even-length $1-2-3-4-5$ alternating path. Therefore, depending on the order in which the search procedure scans labeled nodes, node 5 might receive an even or an odd label. But since the search procedure assign only one label to any node (either even or odd), assigning an odd label to node 5 (in case of 1) prevents us from giving the node an even label in subsequent stages, so we miss the opportunity to give node 6 an odd label.

The root cause of the difficulty in solving a nonbipartite cardinality matching problem by our bipartite cardinality matching algorithm is the presence of certain subgraphs called flowers, composed of particular types of paths and odd cycles. (Note that: since bipartite graphs contains no odd cycles, they never contain any flowers.)

Definition of Flowers and Blossoms

Definition: a flower, defined with respect to a matching M and a root node r, is a subgraph with two components;

(1) **Stem:-** A stem is an even-length alternating path that start at the root node r and terminates at some node w. There is a possibility that r = w, in which case we say that the stem is empty.

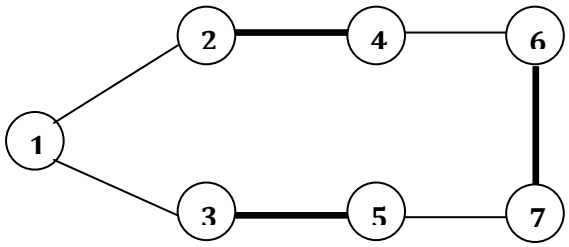
(2) **Blossom:-** A blossom is an odd-length alternating cycle that starts and terminates at the terminal node w of a stem and has no other node in common with the stem.

❖ We refer to the terminal node w as the base of the blossom and we denote a blossom by B.

Example 1: An example of flower which has an empty stem (that is, the root node r and the terminal node w of a stem are the same);

Given:

- A nonbipartite graph $G = \{(1,2), (1,3), (2,4), (3,5), (4,6), (5,7), (6,7)\}$

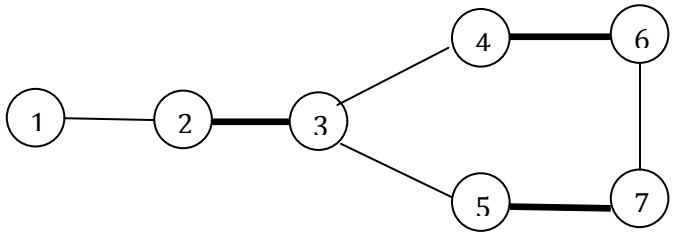


- A matching $M = \{(2,4), (6,7), (3,5)\}$
- A root node r = 1
- A blossom $B = 1 - 2 - 4 - 6 - 7 - 5 - 3 - 1$

Example 2: An example of flower which has a nonempty stem;

Given:

- A nonbipartite graph $G = \{(1,2), (2,3), (3,4), (3,5), (4,6), (5,7), (6,7)\}$



- A matching $M = \{(2,3), (4,6), (5,7)\}$
- A root node r = 3
- A stem is $1 - 2 - 3$
- A blossom $B = 3 - 4 - 6 - 7 - 5 - 3$

Properties of Flower

Property 2.1 A blossom contains K matched arcs for some integer $K \geq 1$ and spans $2K + 1$ nodes.

Property 2.2 The base of a blossom is an even node.

Property 2.3 Every node i in the blossom (except its base) is reachable from the root node (or from the base of the blossom) through two distinct alternating path; one has even length and the other has odd length. The even alternating path to node i terminates with a matched arc, and the odd alternating path to node i terminates with an unmatched arc.

Contracting a Blossom

If nonbipartite graph contains a blossom with respect to the current matching M and the root node r, our search procedure for bipartite cardinality matching algorithm fail to identify an augmenting path even if the graph contains one. Each node i in the blossom is qualified to receive an even label because the graph contains an even alternating path from the root node r to node i (property 2.3). But the search procedure will give even labels to some nodes in the

blossom and odd labels to others. Notice that when the search procedure scans even-labeled nodes, it can label nodes outside the blossom by searching along all unmatched arcs incident to nodes in the blossom; however, it label only the nodes in the blossom when it scans odd-labeled nodes.

If we could give all the nodes in the blossom an even label, whenever we detect a blossom, the search procedure would always detect an augmenting path if it exists. To achieve this objective; one of the more popular approaches is to contract (or shrink) the blossom into a single node.

- Let us see the steps how a blossom $B = i_1 - i_2 - \cdots - i_k - i_1$ is replace by a single new node say $'b'$ (or contracting a blossom):

  Step 1: Introduce a new node b and define its adjacency list $A(b) = A(i_1) \cup A(i_2) \cup \ldots \cup A(i_k)$.

  Step 2: Update the adjacency list of every node $j \in A(b)$ by executing $A(j) = A(j) \cup \{b\}$.

  Step 3: To be able to recover information about the nodes within the blossom that we have contracted into the single node b:

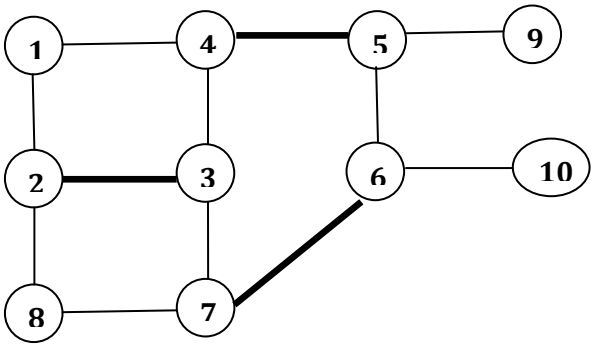
  - Form a circular doubly linked list of nodes $i_1, i_2, \ldots, i_k$.
  - Delete the nodes $i_1, i_2, \ldots, i_k$ and all arcs incident to these nodes from the graph.

  Note that: this operation requires the updating of the adjacency list of all the nodes that are adjacent to the deleted nodes.

- We refer to the resulting graph $G^c = (N^c, A^c)$ as the contracted graph.
- $A^c(i)$- denote the adjacency list of a node i in G^c.
- M^c- denote the corresponding matching in the contracted graph.
- We refer to a new node b as a pseudonode. A pseudonode is always an even node because it merges the entire blossom into its base, which is always even. Consequently, contracting the entire blossom into a single even pseudonode amounts to assigning even labels to each blossom node in the original graph.

Example: illustrates a contraction of a blossom;

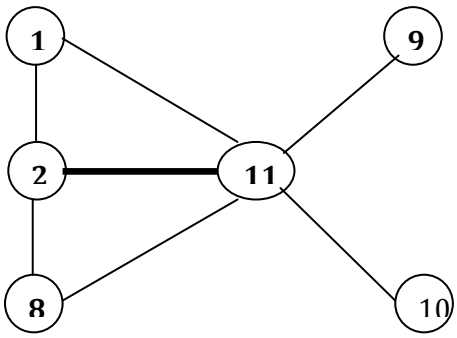
Given a nonbipartite graph:



- A matching $M = \{(2,3), (4,5), (6,7)\}$
- A flower is $1 - 2 - 3 - 4 - 5 - 6 - 7 - 3$
- A blossom $B = 3 - 4 - 5 - 6 - 7 - 3$

Contracting a given blossom:

- Let a pesudonode $b = 11$ and the adjacency list of pseudonode $A^c(11) = \{1,2,8,9,10\}$.
- $M^c = \{(2,11)\}$, and the contracted graph shows below;



2.3.2 Nonbipartite Cardinality Matching Algorithm

The general algorithmic description of our nonbipartite cardinality matching algorithm is the same as our bipartite cardinality matching algorithm with the exception of a change in the search procedure. This algorithm; start with a feasible matching M of a nonbipartite graph $G = (N, A)$

and try to identify an augmenting path starting at unmatched node r. If the algorithm discovers such a path P, it will replace a matching M with $M \oplus P$; otherwise, it will delete node r and all the arcs incident to it from the graph.

Identifying augmenting path

Like the bipartite algorithm , nonbipartite cardinality matching algorithm also construct an alternating tree to find whether or not the graph contains an augmenting path starting at unmatched node r.

Constructing alternating tree (or search procedure)

The modified search procedure construct an alternating tree in the following manner:

> As the search procedure of the bipartite matching algorithm proceeds, it assigns even or odd labels to the nodes. Although the algorithm will never relabel an already labeled node, it will identify the possibility of assigning an odd label to a node with an even label, or of assigning an even label to a node with an odd label. When we find that we can, for the first time, assign a node, say node i, a label other than what it already has, and we suspend the search procedure. At this point we have discovered an even as well as an odd alternating path to node i. If we trace back the predecessor indices of these paths until we encounter the first common node on these paths, the arc we have traced constitute a blossom and the first common node (which has an even label) is the base of the blossom. Then contract the blossom into a pseudonode, and continue the search procedure.
>
> It is possible to perform several contractions before we either discover an augmenting path (in the contracted graph) or run out of nodes to scan, which indicate that the nonbipartite graph contains no augmenting path starting from the root node r.

Now let us see the general algorithmic description step by step;

Step 0 (Start)

- The undirected nonbipartite graph $G = (N, A)$ is given.
- Let M be any feasible matching, possibly the empty matching.

- No nodes are labeled.

Step 1 (Labeling: constructing alternating tree and finding augmenting path)

- Set $A^c(i) \coloneqq A(i)$ for all nodes $i \in N$, where $A(i)$ is the node adjacency list in G.
- Choose one unmatched node $r \in N$ and give an even label to node r and initialize $LIST = \{r\}$, where $LIST$ is a set which stores labeled nodes.
- Select and remove a node i from the set $LIST$ if $LIST \neq \emptyset$.
- If a selected node i has an even label, then scan its node adjacency list $A^c(i) = \{j \in N / (i,j) \in A\}$. That is,

  For every node $j \in A^c(i)$ **do**

  - If node j has an even label, then go to step 2 (or contract (i,j)).
  - If node j is unmatched ($or\ j \in A^c(i)\ and\ j \in N - M$), then assign an odd label to node j and set $q \coloneqq j$, and $pred(q) \coloneqq i$.
    At this time the algorithm discover an augmenting path terminates at node q. STOP and go to step 3.
  - If node j is matched and unlabeled ($or\ j \in A^c(i), j \in M\ and\ j \notin LIST$), then set $pred(j) \coloneqq i$, assign an odd label to node j and add it to $LIST$.
- If a selected node i has an odd label;

  Let node i be matched to node j which is unique, then **do**

  - If node j has an odd label, then go to step 2 (or contract(i,j)).
  - If node j is unlabeled, then set $pred(j) \coloneqq i$, assign an even to node j and add it to $LIST$.
- If $LIST = \emptyset$, STOP

Step 2 (Contracting a Blossom)

- If the case contract(i,j) is happen in step 1, **do**
  - Trace back the predecessor indices of nodes i and j to identify a blossom B.

- Create a new node b called pseudonode and define $A^c(b) = \cup_{k \in B} A^c(k)$.
- Assign an even label to node b and add it to $LIST$.
- For each node $j \in A^c(b)$ **do**

  $A^c(j) = A^c(j) \cup \{b\}$
- Form a circular doubly linked list of nodes in B.
- Delete the nodes in B from the graph and update the data structure.

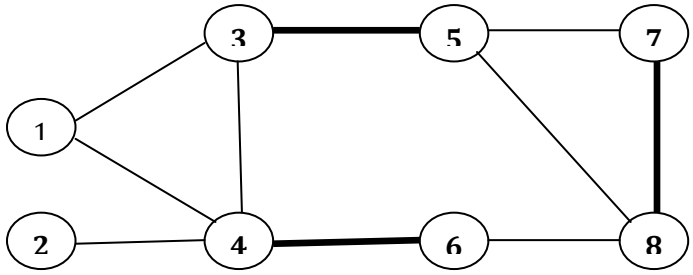
Step 3 (Augmentation)

- If an augmenting path P' is found in step 1 which terminates at node q, then trace the augmenting path P' by starting at node q and traversing the predecessor indices.
- If the path P' contains pseudonodes, then expand the corresponding blossoms and obtain an augmenting path P in the original network.
- Update the matching M using the operation $M = M \oplus P$.
- Remove all labels from nodes and return to step 1.

Step 4 (Termination Criteria)

- An augmenting path P' is not found in step 1, delete node r and all arcs incident to it from the graph G. Return $G := G^*$, where G^* is the new graph formed by deleting node r and all arcs incident to it from the graph G.
- All nodes in the graph G^* are matched; therefore, the matching is maximum and STOP.

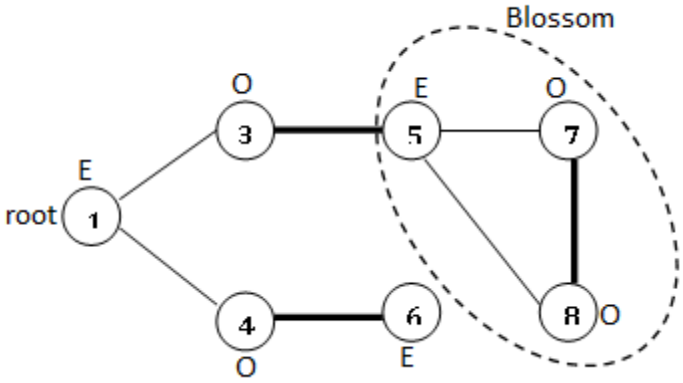
Example: Apply the nonbipartite cardinality matching algorithm to find maximum cardinality matching on the given nonbipartite graph. Let a matching $M = \{(3,5), (4,6), (7,8)\}$ is given. Assume that the algorithm scans labeled nodes in the first-in, first-out order and labels the adjacency list of any node in increasing order of the node number.



Solution

Constructing alternating tree

- Select node 1 as the root node.
- Scans the nodes in the following order; 1,3,4,5,6,7
- When the algorithm scan node 7 (which has odd label), it select its unique matched arc (7,8) and it gives even label to node 8, but node 8 already has an odd label. So that, the algorithm discovers the blossom $B_1 = 5 - 7 - 8 - 5$.



Contracting the blossom $B_1 = 5 - 7 - 8 - 5$

- Introduce a new node called pseudonode numbered 9.
- To contract the blossom nodes into the pseudonode 9:

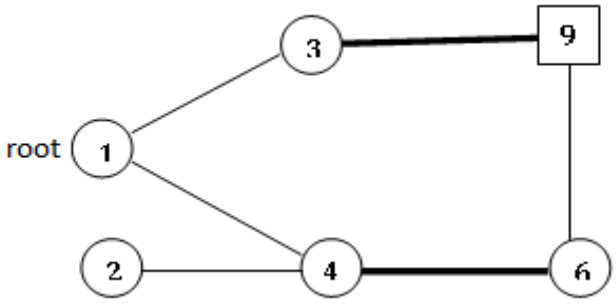
$A(9) = \{3,5,6,7,8\}$

For each node $j \in A(9)$, we have $A(j) = A(j) \cup \{9\}$ such as;

$A(3) = \{1,4,5,9\}$

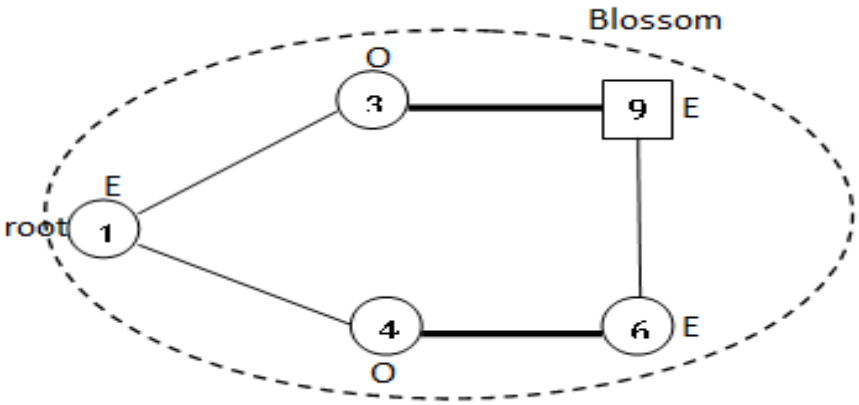
$$A(6) = \{4,8,9\}$$

- Delete the nodes in B_1 from the original graph
- Assign an even label to node 9
- To distinguish a pseudonode from a node of the original graph, we use a square instead of a circle. Therefore, the contracted graph is;



Continuing the search procedure

- At this time, node 9 is the only unscand node and it has even label.
- When we scan the adjacency list of node 9, we discover another blossom $B_2 = 1-3-9-6-4-1$ because node 6 already has an even label.



Contracting the blossom $B_2 = 1-3-9-6-4-1$

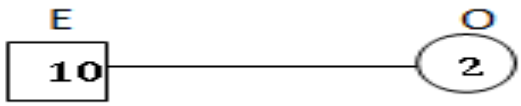
- Introduce another new pseudonode numbered 10.
- To contract the blossom nodes into the pseudonode 10:

$$A(10) = \{1,2,3,4,6,9\}$$

For each node $j \in A(10)$, we have $A(j) = A(j) \cup \{10\}$ such as;

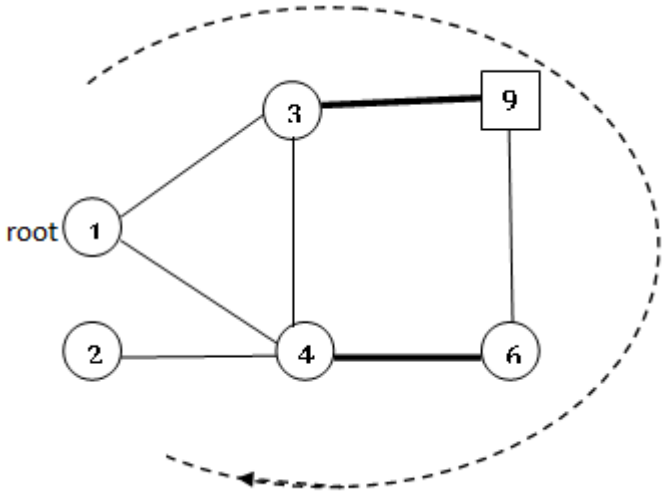
$A(2) = \{4,10\}$

- Delete the nodes in B_2 from the graph
- Assign an even label to node 10
- When we scan the pseudonode 10, we assign an odd label to the unmatched node 2 and then discover an augmenting path $10 - 2$.



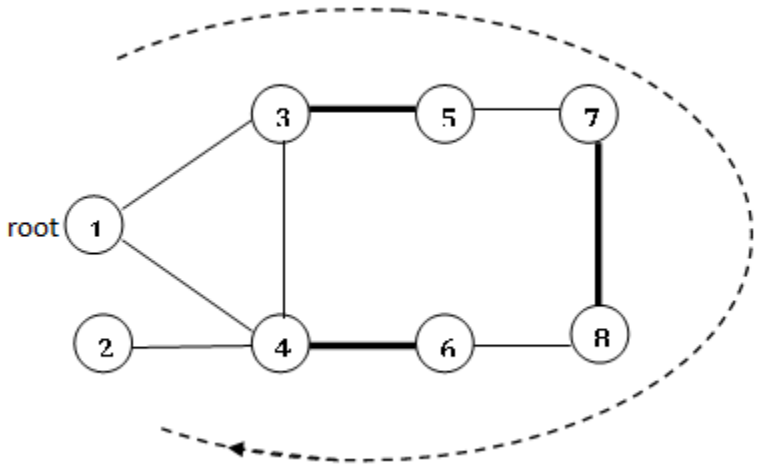
Expanding the pseudonode 10

- Node 2 is adjacent to the blossom node 4.
- To the arc (2,4), we add the even alternating path from the root node to node 4. Therefore we get the path $1 - 3 - 9 - 6 - 4 - 2$.



Expanding the pseudonode 9

- To the arc (2,4), we add the even alternating path from the root node to node 4 in the following graph and we obtain the augmenting path $1 - 3 - 5 - 7 - 8 - 6 - 4 - 2$ in the original network.

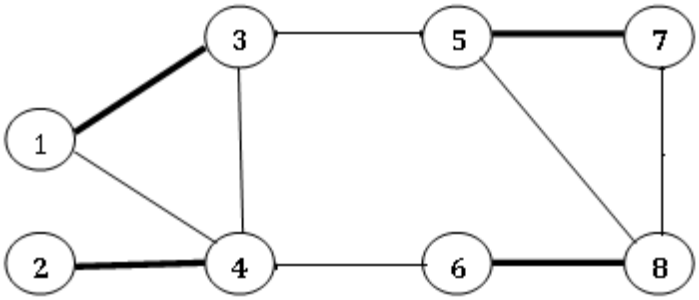


Augmentation

- We now update the given matching M by using the operation $M \oplus P$,

  Where, $P = 1 - 3 - 5 - 7 - 8 - 6 - 4 - 2$

  $$M \oplus P = \{(1,3), (5,7), (6,8), (2,4)\}$$



- All nodes in the graph are matched, therefore the matching $M \oplus P = \{(1,3), (5,7), (6,8), (2,4)\}$ is maximum cardinality matching

Correctness of the Nonbipartite Matching Algorithm

To show that the algorithm correctly finds a maximum matching, we need to show that:

(1) Whenever we find an augmenting path in the contracted graph we can also find an augmenting path in the original graph.

(2) By contracting blossoms we do not add or omit augmenting paths.

To prove this result, we assume that we contract only one blossom. If we do contract more than one blossom, we can use this result iteratively to prove the validity of multiple contractions.

Lemma 2.2 Let G^c be the contracted graph and M^c be the matching in the contracted graph. If the contracted graph G^c contains an augmenting path P^c starting at the root node r (or the pseudonode containing r) with respect to the matching M^c, then the original graph G contains an augmenting path starting at the root node r with respect to the matching M.

Proof

Assume that we contract only one blossom B with respect to a matching M.

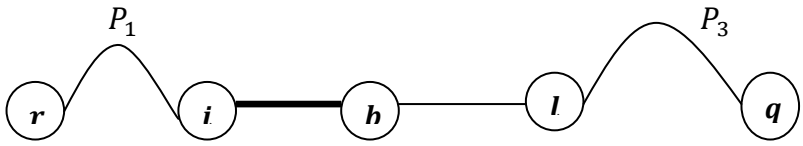
Let w be the base of the blossom B and let b be the pseudonode created by contracting a blossom B.

Case 1:

If $b \notin P^c$, then P^c is an augmenting path in G with respect to the matching M

Case 2:

Suppose that b is an interior node of P^c . This implies the structure of P^c looks like:



Where P_1 and P_3 are alternating path.

By property 2.2, the pseudonode b is an even node and the alternating path from node r to b ends with a matched arc.

$$P^c = \{P_1, (i, b), (b, l), P_3\}$$

Observe that, in the original graph G node l is incident to some node in the blossom, say node k.

By property 2.3, the original graph G contains an even alternating path from node w to node k that ends with a matched arc. Let P_2 denote this path.

Observe that the path $P = \{P_1, (i,w), P_2, (k,l), P_3\}$ is an augmenting path in the original graph G.

Similarly, if b is the first node of the augmenting path P^c, then $r = w$ and the path $P = \{P_2, (k,l), P_3\}$ is an augmenting path in the original graph G. Hence the result

Lemma 2.3 Let G^c be the contracted graph and M^c be the matching in the contracted graph. If G contains an augmenting path from node r to node q with respect to a matching M, then G^c contains an augmenting path from node r (or the pseudonode containing r) to node q with respect to the matching M^c.

Proof

Assume that we contract only one blossom B with respect to a matching M.

Let w be the base of the blossom B and let b be the pseudonode created by contracting a blossom B.

Without loose of generality, suppose that G contains an augmenting path P from node r to node q with respect to a matching M and that node r and q are the only unmatched nodes in G.

If the augmenting path P has no node in common with the nodes in the blossom B, then P is also an augmenting path in the contracted graph G^c .

If P have some nodes in common with the blossom B:

Case 1: When the stem is empty

This implies node r is the base of B and the pseudonode b in G^c contains node r.

Let node i be the last node of P that lies in the blossom.

For some node j and for some unmatched arc (i,j), P has the form; $P = \{P_1, (i,j), P_2\}$ where $P_1 and\ P_2$ are alternating path.

Observe that the path P_1 might have some arcs in common with the blossom B.

Therefore, the path $P^c = \{(b, j), P_2\}$ is an augmenting path in the contracted graph G^c. Hence the result

Case 2: When the stem is nonempty

Let P_3 denote the even alternating path from node r to node w.

Consider the matching $M' = M \oplus P_3$. In M' node r is matched and node w is unmatched. M and M' have the same cardinality, since P_3 is even.

Since G contains an augmenting path with respect to M, G must also contain an augmenting path with respect to M' (by augmenting path theorem).

With respect to M' nodes w and q are the only unmatched nodes in G. So that G must contain an augmenting path between these two nodes.

Let $M^{c\prime}$ denote the matching in the contracted graph G^c corresponding to the matching M' in the graph G.

In the matching M', the blossom B has an empty stem. By case 1, this implies the graph G contains an augmenting path after we contract the nodes of the blossom. Consequently, G^c contains an augmenting path with respect to the matching $M^{c\prime}$.

Since $M^{c\prime}$ and M^c have the same cardinality, G^c must also contain an augmenting path with respect to the matching M^c. Hence the result

Note: The preceding two lemmas show that the contracted graph contains an augmenting path starting at root node r if and only if the original graph contains one. Therefore, nonbipartite cardinality matching algorithm correctly computes a maximum matching in nonbipartite graph.

CHAPTER THREE

3. Minimum Cost Perfect Matching Problem

In this chapter we consider the problem of finding a minimum cost perfect matching in the general undirected graph $G = (N, A)$ with weights or cost c_{ij} associated to each edges and $|N| = n$ and $|A| = m$. We assume that n is even, otherwise there is no perfect matching in G. If M is a perfect matching in G, it is also a maximum cardinality matching in G.

3.1 problem formulation and optimality condition

Let us see the general problem formulation of minimum cost perfect matching problem on the general undirected graph G.

Let M be a perfect matching and $X = (x_{ij})$ be a $0-1$ perfect matching vector defined on A, where $x_{ij} = 1$ if $(i, j) \in M$, 0 otherwise. For each node $i \in N$, we can define

$$\sum_{(i,j)\in A} x_{ij} = 1$$

Let $Y \subseteq N$ with $|Y|$ is odd and $|Y| \geq 3$. Then we can define

$$\sum_{\{(i,j)\in A/\ i\wedge j\in Y\}} x_{ij} \leq (|Y| - 1)/2$$

which is called matching blossom inequality or matching blossom constraint corresponding to Y.

Note: Each subset of N of odd cardinality greater than or equal to 3 leads to a blossom inequality, and every matching vector X satisfies all matching blossom inequalities.

Let $\{Y_1, Y_2, \dots, Y_L\}$ be the set of all distinct subsets of N of odd cardinality greater than or equal to three. Therefore, the general linear programming (LP) model of minimum cost perfect matching problem is:

Minimize

$$Z(X) = \sum_{(i,j)\in A} c_{ij}x_{ij}$$

Subject to ..... **(3.1)**

$$\sum_{(i,j)\in A} x_{ij} = 1, \ \ for\ all\ i \in N$$

$$\sum_{\{(i,j)\in A/\ i \wedge j \in Y_\sigma\}} x_{ij} \leq \frac{(|Y_\sigma| - 1)}{2}, \ \ \sigma = 1\ to\ L$$

$$x_{ij} \geq 0\ , \ for\ all\ \ (i,j) \in A$$

Every perfect matching vector in G is feasible to problem (3.1) and every integer feasible vector for problem (3.1) is a perfect matching vector in G. So, if an optimum solution of problem (3.1) is an integer vector, it is a minimum cost perfect matching vector in G. We will discuss a primal-dual algorithm for solving problem (3.1) known as the blossom algorithm for the minimum cost perfect matching problem.

To write the dual of problem (3.1), associate a dual variable π_i with the constraint corresponding to node i in problem (3.1), and a dual variable μ_σ with the blossom inequality corresponding to the odd subset Y_σ, $\sigma = 1\ to\ L$. The dual variables π_i are only associated with original nodes in G and hence are called **original node prices**. The dual variables μ_σ are known as **pseudonode prices**.

Let $\pi = (\pi_i) = (\pi_1, \pi_2, \dots, \pi_n)$ and $\mu = (\mu_\sigma) = (\mu_1, \mu_2, \dots, \mu_L)$. Since the number of odd subset of nodes L, grows exponentially with n, the vector μ has a lot of entries.

Given the dual solution (π, μ), define for each $(i,j) \in A$;

$$\mu^-(i,j) = \sum \mu_\sigma \text{ , where both } i,j \in Y_\sigma$$

The dual of problem (3.1) is:

$$\text{Maximize} \quad w(\pi,\mu) = \sum_{i\in N}\pi_i - \sum_{\sigma=1}^{L}(|Y_\sigma|-1)(\mu_\sigma)/2$$

$$\text{Subject to} \quad d_{ij}(\pi,\mu) \le c_{ij} \text{ , for each } (i,j)\in A$$

$$\mu \ge 0 \qquad \text{..... (3.2)}$$

Where,

$$d_{ij}(\pi,\mu) = \pi_i + \pi_j - \mu^{-}(i,j)$$

Given a dual feasible solution (π,μ), define $A_*(\pi,\mu)$ as

$$A_*(\pi,\mu) = \{(i,j): (i,j)\in A \text{ and } d_{ij}(\pi,\mu) = c_{ij}\}$$

Since (π,μ) satisfies the first constraint in problem (3.2) as an equation for each edge in $A_*(\pi,\mu)$, they are called **equality edges** with respect to (π,μ), and the subnetwork $G_*(\pi,\mu) = (N, A_*(\pi,\mu))$ is known as the **equality subnetwork** with respect to (π,μ) for problem (3.1).

Complementary slackness optimality condition

The complementary slackness conditions for optimality in the primal, dual pair of problems (3.1) and (3.2) are:

(1) If $x_{ij} > 0$, then $d_{ij}(\pi,\mu) = c_{ij}$, for each $(i,j)\in A$

(2) If $\mu_\sigma > 0$, then $\sum_{\{(i,j)\in A:\, i,j\in Y_\sigma\}} x_{ij} = (|Y_\sigma|-1)/2$, $\quad \sigma = 1,2,\dots,L$

Let $G' = (N', A')$ be the current network at some stage after some blossom have been shrunk (contract). Node in N' are called current nodes, they may be either pseudonodes, or original nodes in N not contained in any blossoms shrunk so far. Let $b \in N'$ be a current node. We will say that an original node i is inside b or that b contains i inside it, if either $i = b$, or if i is a node in the blossom corresponding to b. An original edge $(i,j)\in A$ is said to be inside b if both i and j are inside b.

3.2 Blossom Algorithm for the Minimum Cost perfect Matching Problem

This algorithm is initiated with an initial dual feasible solution (π^o, μ^o) in which $\mu^o = 0$. It maintains a matching vector X, and (π, μ) always feasible to problem (3.2). It alternates between changing the matching vector X, keeping (π, μ) constant, using the maximum cardinality matching algorithm of section 2.3.2 in the equality subnetwork $G_*(\pi, \mu)$ or changing (π, μ) keeping X constant. The algorithm always maintains that if $x_{ij} = 1$, then $(i, j) \in A_*(\pi, \mu)$, so that, the complementary slackness optimality condition (1) holds always.

The pseudonode prices $\mu_\sigma > 0$ always imply that the associated Y_σ is the set of original nodes inside an existing pseudonode, so the complementary slackness optimality condition (2) holds always. This also guarantees that even though the dual vector μ has a large number of entries, all but at most $(n/2)$ of them will be zero at every stage. Hence, it is only necessary to store values of μ_σ associated with each at that stage of the algorithm.

When we apply the maximum cardinality matching algorithm on the equality subnetwork, if there is unmatched node and augmenting path is not found in the current equality subnetwork, the approach now goes to a dual solution change step. The purpose of this is to obtain a new dual feasible solution, the equality subnetwork corresponding to which allows a matching of higher cardinality than the previous one. This dual solution change step is designed to satisfy the following properties.

i. All present matching edges (which are equality edges now) remain equality edges after the change, so that the present matching vector X continues to satisfy the complementary slackness optimality condition (1) together with the new dual solution.

ii. The equality, nonequality status of all original edges contained within any existing pseudonode remains unchanged, and all in tree current equality edges remain equality edges after the dual solution change. So, all the existing alternating trees in the current equality subnetwork, are also contained in the new current equality subnetwork after the dual solution change.

iii. In the new current equality subnetwork obtained after the dual solution change, at least one of the trees can grow, or there is at least one augmenting path, etc. This

makes it possible to repeat the application of the maximum cardinality matching algorithm in the new current equality subnetwork.

iv. In the new dual solution, some of the dual variables μ_σ may be given positive values, but if $\mu_\sigma$is positive, the corresponding subset of original nodes Y_σ will always be the set of original nodes inside an existing pseudonode. We have already seen that the present matching vector satisfies the blossom inequality corresponding to every existing pseudonode as an equation, so this guarantees that the complementary slackness condition (2) continues to hold.

Changes in Blossoms after an Augmentation step

Let $X, (\pi, \mu)$ be the solution pair at some stage during the matching change phase of the algorithm. If an augmenting path P' is discovered, we would augment. In the maximum cardinality matching algorithm we then discard all the existing blossoms (pseudonodes) after augmentation. However, some of these blossoms may correspond to pseudonodes associated with a positive μ_σ in the present dual solution (π, μ) and discarding these blossoms (pseudonodes) will violate the property maintained by the algorithm (that is, $x_{ij} = 1$ implies that $(i, j) \in A_*(\pi, \mu)$). So, in this algorithm, all the blossoms along the augmenting path are retained after augmentation, but changes have to be made in the stored data on the blossoms corresponding to them, to reflect the change in the matching caused by the augmentation step. This is called the operation of **revising all the blossoms along *P'***.

We describe this operation for a pseudonode b on P'. Let P be the augmenting path in $G_*(\pi, \mu)$ corresponding to P'. Let i_1 be the base node of a blossom B. when the pseudonode b along P' containing the blossom B within them is expanded, we will get the portion of the augmenting path passing through the blossom corresponding to b. We will denote this portion of the path by P^o. We consider two cases.

Case 1: The Base of a Blossom *B* is an Intermediate Node on *P*

In this case the augmenting path *P* passes through the matching edge incident at the base node of *B*, and leaves through either the base node or some nonbase node within *B*. We consider two subcases.

Subcase 1: *P* passes only through the base node of *B*

In this subcase P^o appears as in Figure 3.1(a). After the augmentation is carried out, (q_1, i_1) becomes a nonmatching (unmatched) edge and (i_1, q_2) becomes a matching edge. The only change needed in the stored data on this simple blossom, is to change the label on its base node i_1 as in Figure 3.1(b).

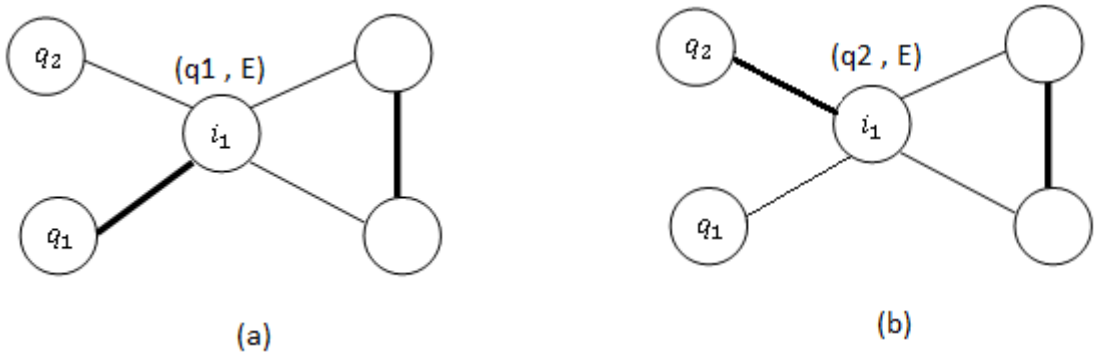


Figure 3.1: before augmentation (a), and after (b). Matching edges are thick. In figure 3.1(a), $P^o = q_2 - i_1$. The nodes q_1 , q_2 are not on the blossom. Stored label on i_1 is entered by its side.

Subcase 2: *P* contains some Nonbase Nodes Contained in *B*

In this subcase P^o appears as in Figure 3.2. The i-nodes are nodes on the blossom corresponding to b, and q_1 , q_2 are nodes outside *B*. The odd cycle in this blossom remains the same, but augmenting changes the matching edges along P^o in to nonmatching edges and vice versa.

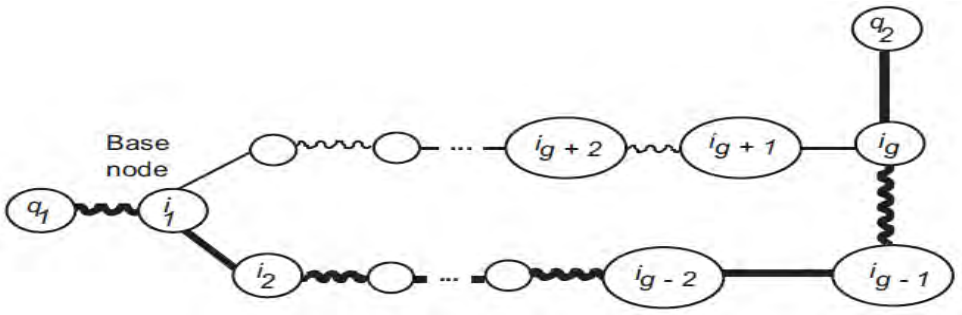


Figure 3.2: Matching edges are wavy. P_0, the thick path, contains more than one node on the simple blossom corresponding to B. See Figure 3.3 for position after augmentation.

After augmentation (see Figure 3.3) (i_1, i_2) is a matching edge, and i_1 is no longer the base node, i_g becomes the new base node of this blossom, and its base will be the revised base of i_g (that is, after the corresponding change due to augmentation is carried out for i_g). Change the label on i_g in the stored data to (q_2, E). Change the labels on both the neighbor nodes of i_g on the odd cycle, i_{g+1} and i_{g-1}, to (i_g, O). Change the labels on i_{g+2}, i_{g-2} to (i_{g+1}, E) and (i_{g-2}, E) respectively. Keep on changing the labels along the odd cycle this way until all the node labels are changed. The last pair of nodes to be relabeled in this manner is the new identifying pair of nodes for this blossom. It is clear that the odd alternating cycle in this simple blossom, the new matching and nonmatching edges in it, can all be retrieved by the procedures discussed earlier using the revised labels on the nodes.

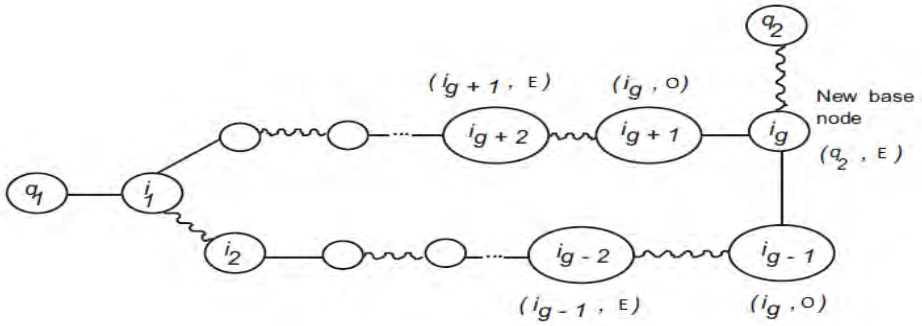


Figure 3.3: Position after augmentation in the simple blossom in Figure 3.1. New base node and new node labels are indicated.

Case 2: The Base of a blossom *B* Is a Terminal Node of *P*

In this case i_1, the base node of *B* is an unmatched node. We again consider two subcases.

Subcase 1: *P* Contains Only the Base Node of *B*

In this subcase P^o appears as in Figure 3.4(a). After the augmentation (q_1, i_1) becomes a matched edge, change the stored label on i_1 as in Figure 3.4 (b).

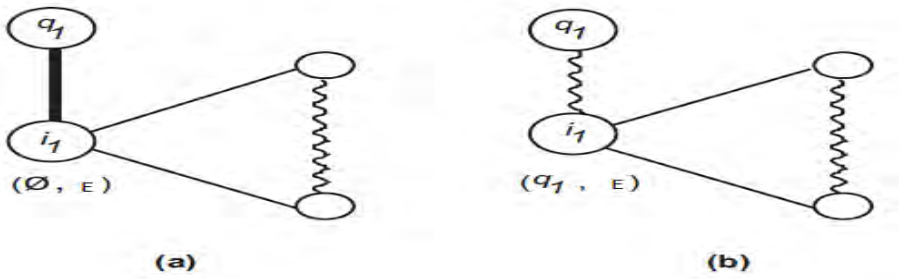


Figure 3.4: Before augmentation (a), and after (b). P^o is thick in Figure (a), and contains only unmatched base node i_1. q_1 is not on the simple blossom of *B*. Stored label on i_1 is entered by its side.

Subcase 2: *P* Contains Some Nonbase Nodes Contained in *B*

In this subcase P^o appears as in Figure 3.5(a). In this subcase i_g becomes the new base node after augmentation, and the revision is carried out as in Subcase 2 of Case 1. See Figure 3.5(b).

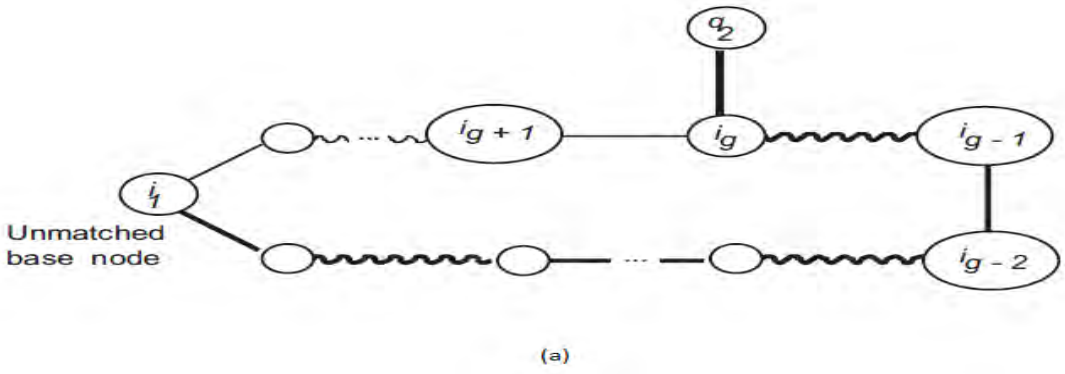


Figure 3.5(a): Matching edges are wavy. P^o is thick, it contains more than one node on the simple blossom of **B**. See Figure 3.5(b) for position

after augmentation.

After augmentation, we remove all labels from the current nodes of alternating tree which contains the augmenting path. This operation is called **chopping down the tree**. In this operation only the tree structure are eliminated, but all the current nodes on it are left as unlabeled nodes in the current network. But in our algorithm of section 2.3.2, all the blossoms are thrown away after augmentation.

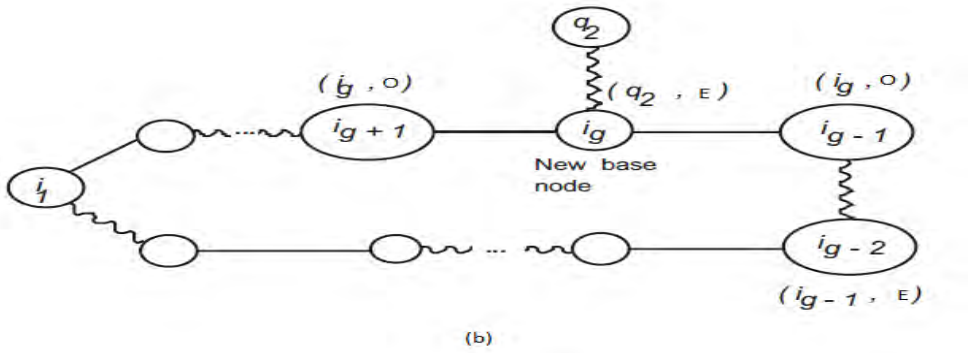


Figure 3.5(b): New base node, new labels after augmentation are indicated.

When the growth of the remaining alternating trees in the present $G_*(\pi, \mu)$ is resumed, it is possible that some of the unlabeled pseudonodes get labeled. In this process they may get label either as even or odd. In Section 2.3.2, pseudonodes were always even labeled nodes, and carried the same label as their base nodes. Here these labels may be quite different. Also, pseudonodes may be labeled as odd nodes.

In the algorithm to be discussed in this section a freshly shrunk (contract) pseudonode always gets labeled as an even node, its label at that time will be the same as that on the base node of the corresponding simple blossom before it was shrunk. So, any odd labeled pseudonode in the current network must have received that label after remaining as an unlabeled node for some time. Hence these pseudonodes may prevent us from discovering augmenting paths in the current equality subnetwork $G_*(\pi, \mu)$ with the present matching. The only reason for keeping such pseudonodes is to satisfy the property maintained by the algorithm (That is, $x_{ij} = 1$ implies that $(i, j) \in A_*$), if the dual variables μ_σ corresponding to them are strictly positive in the present

dual solution. Therefore, whenever there is a pseudonode which is an odd labeled current node associated with $\mu_\sigma = 0$ in the present dual solution, we unshrink that pseudonode into the simple blossom corresponding to it. This unshrinking operation is discussed next.

Unshrinking an Inner Labeled Pseudonode Associated with $\mu_\sigma = 0$

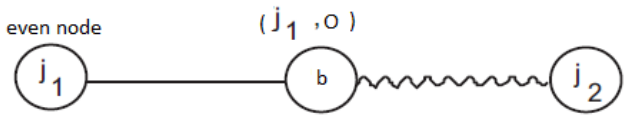


Figure 3.6: Wavy edge is a current matching edge. Odd labeled pseudonode b associated with $\mu_\sigma = 0$ to be unshrunk. Label of b is by its side.

This operation is carried out only on pseudonodes which are odd labeled current nodes associated with $\mu_\sigma = 0$. Let b with label (j_1, O) be such a pseudonode. So, j_1, is an even node and (j_1, b) is a current unmatched edge. By earlier discussion such a pseudonode will always be matched, let (b, j_2) be the current matching edge incident at it. See Figure 3.6. Let $G_b(N_b, A_b)$ be the simple blossom corresponding to b. Since (j_1, b) is a current in-tree unmatched edge, there must exist $i_1 \in N_b$ such that (j_1, i_1) is a current equality edge at the stage that pseudonode b was formed. Either i_1 is the base node of b, or there exists an alternating path in G_b from i_1 beginning with the matching edge incident at i_1, to its base node. Let this path be i_1, (i_1, i_2) , i_2, . . . , (i_{g-1}, i_g), i_g, with i_g being the base node of G_b (if i_1 is itself the base node, $g = 1$, and this path is the empty path containing no edges). So, (i_g, j_2) is a current matching edge at the stage that pseudonode b was formed. See Figure 3.7.

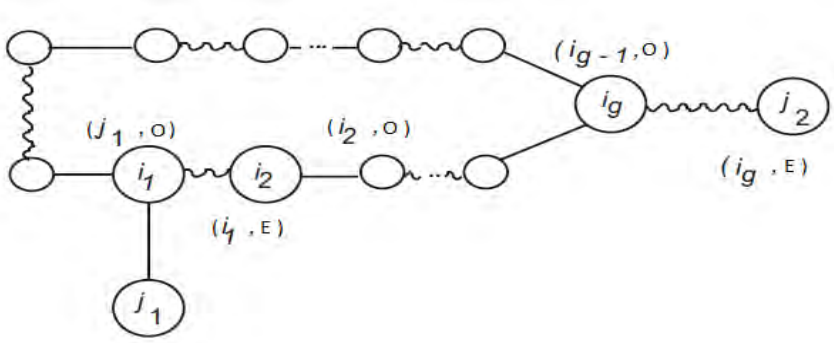


Figure 3.7: Connecting the tree through the alternating path in the simple blossom, after a pseudonode is unshrunk.

Unshrinking of b replaces it with the simple blossom G_b with all its nodes and edges on it. Each of the current edges of the form (p, b) are replaced by edges of the form (p, i) for all $i \in N_b$. After the unshrinking, all the nodes in N_b become current nodes. Update the current matching by replacing the single current matching edge (b, j_2) from it by (i_g, j_2) and all the matching edges within the simple blossom G_b. The alternating tree that contained b before unshrinking, is connected again by labeling the newly introduced nodes along the path $j_1, (j_1, i_1), i_1, \dots, i_g, (i_g, j_2), j_2$ alternately as odd and even nodes as indicated in Figure 3.7.

Now let us see the general algorithmic description step by step.

Step 0 (Initialization)

- Define an initial dual feasible solution to be

  $$(\pi^o = (\pi_i^o), \mu^o)$$

  Where,

  $$\mu^o = 0 \; and \; \pi_i^o = (1/2)(\min.\{c_{ij} : (i,j) \in A\} \; for \; each \; i \in N.$$

- Choose an initial matching M_o (could be empty) in the equality subnetwork $G_*(\pi^o, \mu^o)$.
- No nodes are labeled.

Step 1 (Labeling: Constructing Alternating Tree and finding Augmenting Path)

- If there are no unmatched nodes, go to step 7. Otherwise, choose an unmatched node r and give an even label to node r.
- Initialize $LIST = \{r\}$ where, $LIST$ is a set which store labeled nodes.
- Select and remove a node i from the set $LIST$ if $LIST \neq \emptyset$. If $LIST = \emptyset$, go to step 5.
- If a selected node i has an even label, scan it for each edge (i, j) in the current equality edges and $j \neq pred(i)$. That is;

  For each edge $(i, j) \in A_*(\pi, \mu)$ and $j \neq pred(i)$ **do**

  - If node j is unmatched node, assign an odd label to node j and set $q := j$, and $pred(q) := i$.

At this time the algorithm discover an augmenting path joining two unmatched nodes. STOP and go to step 2.

- If node j already has an even label, the alternating tree containing i and j has blossom, go to step 3.
- If node j is unlabeled and matched, set $pred(j) := i$, assign an odd label to node j and add it to $LIST$.

➢ If a selected node i has an odd label, let (i, j) be the current matched edge incident to node i, then **do**

- If node j already has an odd label, the alternating tree containing i and j has blossom, go to step 3.
- If node j is unlabeled, set $pred(j) := i$, assign an even label to node j and add it to $LIST$.

Step 2 (Augmentation)

➢ We come to this step when an augmenting path P' is found in step 1 which terminates at unmatched node q. Trace the augmenting path P' by starting at node q and traversing the predecessor indices. The path P' is an augmenting path between two unmatched nodes r and q.

➢ If the augmenting path P' contains pseudonodes, expand the corresponding blossoms and obtain an augmenting path P in the present equality subnetwork in G.

➢ Update the matching M using the operation $M = M \oplus P$ and revise all the blossom along P'. Remove all labels from nodes and if there are no unmatched nodes, go to step 7, otherwise return to step 1

Step 3 (Blossom Shrinking or Contracting)

➢ If the case contract (i, j) is happen in step 1, **do**

- Identify a blossom B by tracing back the predecessor indices of nodes i and j. Create a new node b called pseudonode and contract it in to a node b, exact as in step 2 of the algorithm in section 2.3.2.
- Assign an even label to node b and add it to $LIST$ and go back to step 1.

Step 4 (pseudonode unshrinking)

- Unshrink all pseudonodes that are current odd labeled nodes with the associated pseudonode price $\mu_\sigma = 0$ in the present dual solution. Revise the set of the current matching edges and the set of current nodes accordingly. Include in the list all the new even labeled nodes in the blossoms corresponding to the unshrunk pseudonodes. Repeat this procedure again if necessary, until there are no pseudonodes that are odd labeled current nodes associated with $\mu_\sigma = 0$. Go back to step 1.

Step 5 (Dual Solution Change)

- We reach this step if we do not yet have a perfect matching, and for some unmatched root node the alternating tree does not contain augmenting path. This implies that the present matching is a maximum cardinality matching in the current equality subnetwork.
- Let (π, μ) be the present dual feasible solution. Compute the following using the convention that the minimum in the empty set is $+\infty$.

$\delta_1 =$ Min $\{c_{ij} - d_{ij}(\pi, \mu)$: $(i,j) \epsilon A, i[j]$ is inside an even [an labeled] current node$\}$

$\delta_2 =$ Min $\{\frac{1}{2}(c_{ij} - d_{ij}(\pi, \mu))$: $(i,j) \epsilon A$, i and j are inside distinct even current nodes$\}$

$\delta_3 =$ Min $\{\frac{1}{2}\mu_\sigma$: σ such that Y_σ is the set of original nodes inside a current odd labeled

Pseudonode$\}$

$\delta =$ Min $\{ \delta_1, \delta_2, \delta_3 \}$

- If $\delta = +\infty$, go to step 6. If δ is finite, it will be positive, define the new dual solution to be $\pi^1 = (\pi_i^1), \mu^1 = (\mu_\sigma^1)$ where

$$\pi_i^1 = \begin{cases} \pi_i + \delta & for\ all\ i\ inside\ even\ current\ nodes \\ \pi_i - \delta & for\ all\ i\ inside\ odd\ current\ nodes \\ \pi_i & for\ all\ i\ inside\ unlabeled\ nodes \end{cases}$$

$$\mu_\sigma^1 = \begin{cases} \mu_\sigma + 2\delta & if\ Y_\sigma\ is\ the\ set\ of\ original\ nodes\ in\ a\ current\ even\ labeled\ pesudonode \\ \mu_\sigma - 2\delta & if\ Y_\sigma\ is\ the\ set\ of\ original\ nodes\ in\ a\ current\ odd\ labeled\ pseudonodes \\ \mu_\sigma & otherwise \end{cases}$$

- Find $G_*(\pi^1, \mu^1)$. Include all even current nodes in the $LIST$. If $\delta < \delta_3$ go to step 1.

Step 6 (Infeasibility)

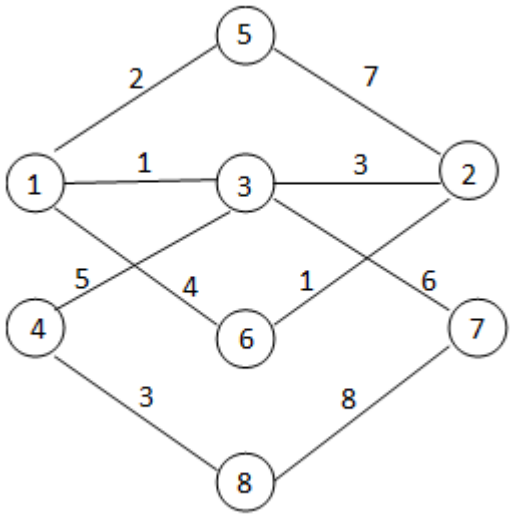
- We come to this step if $\delta = +\infty$ in a dual solution change step. In this case there exists no perfect matching in G.

Step 7 (optimality)

- We come to this step if the matching in the present $G_*(\pi, \mu)$ is a perfect matching, it is a minimum cost perfect matching in G, and the corresponding perfect matching vector is an optimum solution of problem (3.1). **Terminate.**

Example:

Find the minimum cost perfect matching in the following graph by using the above algorithm.



Solution

Iteration 1

- $\mu^o = 0$ and $\pi^o = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{3}{2}, 1, \frac{1}{2}, 3, \frac{3}{2})$
- Choose $X^o = 0$ and $M_o = \emptyset$
- The set of unmatched node is N
- Since $\mu^-(i,j) = 0$, $d_{ij}(\pi, \mu) = \pi_i^o + \pi_j^o$ for all $(i,j) \in A$

$d_{15}(\pi^o, \mu^o) = \frac{3}{2} \neq c_{15}$, $\boldsymbol{d_{13}(\pi^o, \mu^o) = 1 = c_{13}}$, $d_{16}(\pi^o, \mu^o) = 1 \neq c_{16}$,

$d_{25}(\pi^o, \mu^o) = \frac{3}{2} \neq c_{25}$, $d_{23}(\pi^o, \mu^o) = 1 \neq c_{23}$, $\boldsymbol{d_{26}(\pi^o, \mu^o) = 1 = c_{26}}$,

$d_{34}(\pi^o, \mu^o) = 2 \neq c_{34}$, $d_{37}(\pi^o, \mu^o) = \frac{7}{2} \neq c_{37}$, $\boldsymbol{d_{48}(\pi^o, \mu^o) = 3 = c_{48}}$,

$d_{78}(\pi^o, \mu^o) = \frac{9}{2} \neq c_{78}$

Therefore, $A_*(\pi^o, \mu^o) = \{(1,3), (2,6), (4,8)\}$ and $G_*(\pi^o, \mu^o) = (N, A_*(\pi^o, \mu^o))$

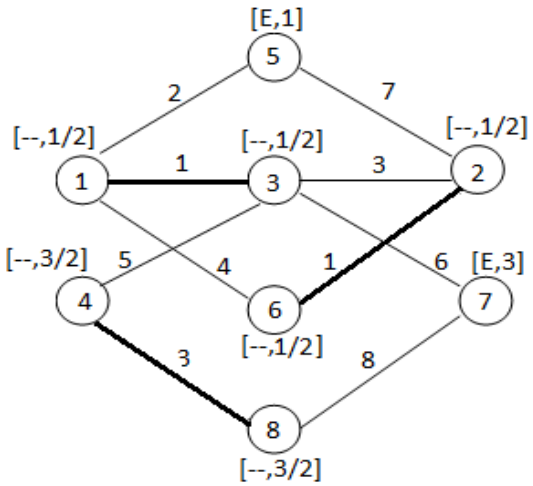
- Construct alternating trees rooted at each unmatched nodes in the equality subnetwork $G_*(\pi^o, \mu^o)$. Then we get augmenting paths;

$P_1 = 1 - 3$, $P_2 = 2 - 6$, $P_3 = 4 - 8$

After augmentation, $M = \{(1,3), (2,6), (4,8)\}$ and $x_{13} = 1, x_{26} = 1, x_{48} = 1$

[Label, π_i]

i



Iteration 2

- At the end of iteration 1, we still have unmatched nodes in the equality subnetwork but augmenting path is not found. Therefore, we have to change dual solution.

$\delta_1 = \text{Min. } \{c_{15} - d_{15}(\pi^o, \mu^o), c_{25} - d_{25}(\pi^o, \mu^o), c_{37} - d_{37}(\pi^o, \mu^o), c_{78} - d_{78}(\pi^o, \mu^o)\}$

$= \frac{1}{2}$

$\delta_2 = +\infty$

$\delta_3 = +\infty$. Therefore, $\delta = \frac{1}{2}$, $\pi^1 = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{3}{2}, \frac{1}{2}, \frac{7}{2}, \frac{3}{2})$ and $\mu^1 = \mu^o = 0$

$\boldsymbol{d_{15}(\pi^1, \mu^1) = 2 = c_{15}}$, $\boldsymbol{d_{13}(\pi^1, \mu^1) = 1 = c_{13}}$, $d_{16}(\pi^1, \mu^1) = 1 \neq c_{16}$,

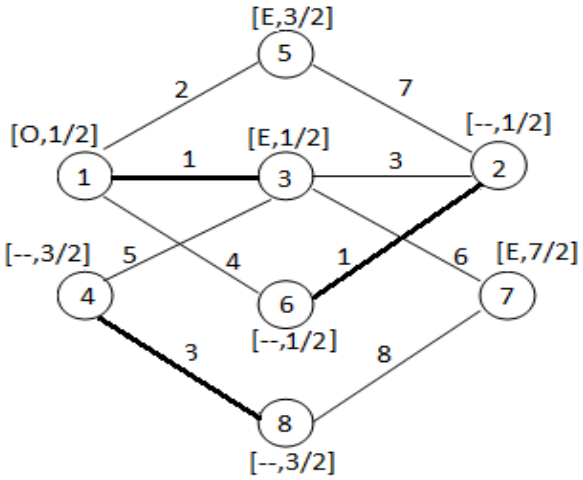
$d_{25}(\pi^1, \mu^1) = 2 \neq c_{25}$, $d_{23}(\pi^1, \mu^1) = 1 \neq c_{23}$, $\boldsymbol{d_{26}(\pi^1, \mu^1) = 1 = c_{26}}$,

$d_{34}(\pi^1, \mu^1) = 2 \neq c_{34}$, $d_{37}(\pi^1, \mu^1) = 4 \neq c_{37}$, $\boldsymbol{d_{48}(\pi^1, \mu^1) = 3 = c_{48}}$,

$d_{78}(\pi^1, \mu^1) = 5 \neq c_{78}$

Therefore, $A_*(\pi^1, \mu^1) = \{(1,3), (1.5), (2,6), (4,8)\}$ and $G_*(\pi^1, \mu^1) = (N, A_*(\pi^1, \mu^1))$.

The set of unmatched nodes is $\{5, 7\}$. Construct alternating trees rooted at each unmatched nodes in the equality subnetwork $G_*(\pi^1, \mu^1)$. Then we get;



Iteration 3

- Dual solution change;

$\delta_1 = \text{Min. } \{c_{25} - d_{25}(\pi^1, \mu^1), c_{23} - d_{23}(\pi^1, \mu^1), c_{34} - d_{34}(\pi^1, \mu^1), c_{78} - d_{78}(\pi^1, \mu^1)\}$

$= 2$

$\delta_2 = \text{Min. } \{\frac{1}{2}(c_{37} - d_{37}(\pi^1, \mu^1))\}$

$= 1$

$\delta_3 = +\infty$. Therefore, $\delta = 1$, $\pi^2 = (-\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{3}{2}, \frac{5}{2}, \frac{1}{2}, \frac{9}{2}, \frac{3}{2})$ and $\mu^2 = 0$.

$\boldsymbol{d_{15}(\pi^2, \mu^2) = 2 = c_{15}}$, $\boldsymbol{d_{13}(\pi^2, \mu^2) = 1 = c_{13}}$, $d_{16}(\pi^2, \mu^2) = 0 \neq c_{16}$,

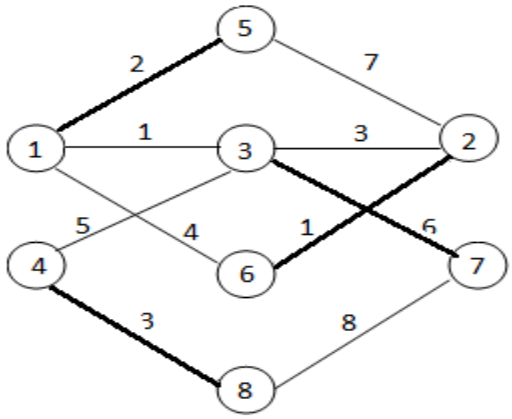
$d_{25}(\pi^2, \mu^2) = 3 \neq c_{25}$, $d_{23}(\pi^2, \mu^2) = 2 \neq c_{23}$, $\boldsymbol{d_{26}(\pi^2, \mu^2) = 1 = c_{26}}$,

$d_{34}(\pi^2, \mu^2) = 3 \neq c_{34}$, $\boldsymbol{d_{37}(\pi^2, \mu^2) = 6 = c_{37}}$, $\boldsymbol{d_{48}(\pi^2, \mu^2) = 3 = c_{48}}$,

$d_{78}(\pi^2, \mu^2) = 6 \neq c_{78}$

Therefore, $A_*(\pi^2, \mu^2) = \{(1,3), (1.5), (2,6), (3,7), (4,8)\}$ and $G_*(\pi^2, \mu^2) = (N, A_*(\pi^2, \mu^2))$.

The unmatched nodes are 5 and 7. Construct alternating trees rooted at each unmatched nodes in the equality subnetwork $G_*(\pi^2, \mu^2)$. Then we get augmenting path $P = 5 - 1 - 3 - 7$. After augmentation we get $M = \{(1,5), (3,7), (2,6), (4,8)\}$ and $x_{15} = 1, x_{26} = 1, x_{37} = 1, x_{48} = 1$.



Optimality: A matching $M = \{(1,5), (3,7), (2,6), (4,8)\}$ is a perfect matching in $G_*(\pi^2, \mu^2)$. Therefore *M* is a minimum cost perfect matching in *G* and the minimum cost is 12.

4. REFERENCE:

[1] D. Fengming, K.K. Meng and T.E. Guan, *introduction to graph theory*, **2007**

[2] Eugene L.Lawler, *combinatorial optimization*: Networks and Matroids, **1976**

[3] J.A. Bondy and U.S.R. Murty, *graph theory with applications*, **1982**

[4] James B. Orlin, Ravindra K. Auja and Thomas L. Magnati, *network flows theory, algorithm, and applications*, **1993**

[5] L.Lovasz and M.D. Plummer, *matching theory annals of discrete mathematics*, **1986**

[6] M. Gondra and M. Minoux, *graphs and algorithm*, **1984**