# MATCHINGS WITH A SIZE CONSTRAINT

Bo Zhou

B. Sc., Beijing Institute of Technology

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF MATHEMATICS

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September, 1990

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Mathematics

The University of British Columbia
Vancouver, Canada

Date Sept 11, 1990

## Abstract

We study the matching problem and some variants such as $b$-matching and $(g, f)$-factors. This thesis aims at polynomial algorithms which in addition have other properties. In particular, we develop a polynomial algorithm which can find optimal solutions of each possible size for weighted matching problem, and a strongly polynomial algorithm which can find a $(g, f)$-factor of fixed size.

## Table of Contents


**Abstract** ii

**ACKNOWLEDGEMENTS** v

**1 Introduction** 1

1.1 Introduction . . . . . 1

1.2 Definitions . . . . . 1

1.3 Outline of The Thesis . . . . . 4

**2 Fixed Size Weighted Matching** 6

2.1 Preliminaries . . . . . 6

2.2 Variable Size Weighted Matching . . . . . 10

2.3 Fixed Size Weighted Matching . . . . . 17

2.4 A Transformation Approach . . . . . 21

**3 The Fixed Size Weighted $b$-matching Problem** 23

3.1 The Weighted $b$-matching Problem . . . . . 23

3.1.1 Introduction and Definitions . . . . . 23

3.1.2 Characterization of the Optimal Solution . . . . . 25

3.1.3 The Blossom Algorithm . . . . . 27

3.1.4 About Complexity . . . . . 34

3.2 The Fixed Size Wighted $b$-matching . . . . . 35

3.2.1 A Possible Generalization . . . . . 35

3.2.2 A Transformation Approach . . . . . 36

**4 Fixed Size $(g, f)$-Factor Problem** **39**

4.1 Preliminaries . . . . . 39

4.2 An $f$-Factor Algorithm . . . . . 40

4.2.1 Network Flow Formulation . . . . . 40

4.2.2 Symmetrizing a Directed $f$-Factor . . . . . 41

4.2.3 Algorithm for Finding an Alternating Walk . . . . . 44

4.2.4 $f$-Barrier . . . . . 47

4.3 Fixed Size $(g, f)$-Factor . . . . . 48

4.3.1 Network Flow Formulation . . . . . 48

4.3.2 Symmetrizing a Directed Size $2p$ $(g, f)$-Factor: Phase One . . . . 48

4.3.3 Symmetrizing a Directed Size $2p$ $(g, f)$-Factor: Phase Two . . . . 51

4.3.4 Fixed Size $(g, f)$-Barrier . . . . . 54

4.4 Augumenting Walk Theorem . . . . . 57

**Bibliography** **61**

## ACKNOWLEDGEMENTS

I am greatly indebted to my advisor, R.P. Anstee for suggesting this topic, for his guidance and criticism, and his patience with me throughout the preparation of this work. I also thank S. Thomas McCormick for reading the manuscript, and providing much constructive advice.

I express my gratitude to the University of British Columbia and the Natural Sciences and Engineering Research Council of Canada for their generous financial support.

# Chapter 1

# Introduction

## 1.1 Introduction

The renowned weighted matching, weighed b-matching, and $(g, f)$-factor problems are all well solved in literature, and a number of algorithms for each of these problems already exist. The motivation of this thesis is to add a constraint to each of these problems which forces our optimum solutions to have a fixed size. We study how this extra constraint affects the original problems. Algorithms are established or modified to handle these new problems, and transformation techniques are also explored either to solve the new problems or prove some theorem.

## 1.2 Definitions

This section will present notation and definition of many concepts used throughout this thesis.

- A **graph** $G = (V(G), E(G))$ is a finite set of vertices $V(G)$, and a finite set of edges $E(G)$. Each edge $e_{ij}$ in $E(G)$ is a set of two vertices $\{v_i, v_j\}$. Each edge $e_{ij}$ is said to be incident to $v_i$ and $v_j$. Edge $e_{ij}$ is called a **loop** if $v_i = v_j$. Usually we denote a graph by $G = (V, E)$.

- A graph without loops is called a **simple graph**. If we allow more than one edge between any pair of vertices $v_i$ and $v_j$, then a simple graph with such multiple edges is called a **multigraph**. The Figure 1.1 shows a multigraph with a loop.

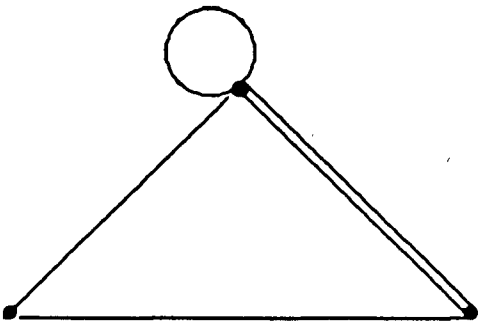

Figure 1.1: A multigraph with a loop

To simplify the exposition, we assume $G$ has no loops throughtout this thesis, this does not fundamentally change the problem.

- A **weighted graph** $G = (V, E, c)$ is a graph with a real number $c_{ij}$ associated with each edge $e_{ij}$, where $c = (c_{ij} : e_{ij} \in E)$ is understood as a vector.

- A **walk** in $G$ is a non-null sequence $w = v_0 e_1 v_1 e_2 v_2 \ldots e_k v_k$, whose terms are alternately vertices and edges, where edge $e_i = \{v_i, v_{i-1}\}$ for each $i$. We say that $w$ is a walk from $v_0$ to $v_k$ of length $k$. Vertex $v_0$ is called the **origin** of $w$, and $v_k$ is called the **terminus** of $w$.

- A walk is called a **trail**, if the edges $e_1, e_2, \ldots, e_k$ of $w$ are distinct. It is called a **path** if, in addition, the vertices $v_0, v_1, v_2, \ldots, v_k$ are also distinct.

- A walk is called a **cycle** if $v_0 = v_k$ and the vertices $v_1, v_2, \ldots, v_k$ are all distinct.

- A graph $H = (V(H), E(H))$ is a **subgraph** of $G = (V, E)$ if $V(H) = V$, and $E(H) \subset E$. For $V_0 \subset V$, a graph with vertex set $V_0$, and whose edge set is the set of edges of $G$ that have both ends in $V_0$ is called the subgraph of $G$ **induced** by $V_0$ and is denoted by $G[V_0]$.

- Two vertices $u$ and $v$ of $G$ are said to be **connected** if there is a path from $u$ to $v$ in $G$. Connection is an equivalence relation on the vertex set $V$. Thus there is a partition of $V$ into nonempty subsets $V_1, V_2, V_3, ..., V_r$ such that two vertices $u$ and $v$ are connected if and only if both $u$ and $v$ belong to the same set $V_i$. The subgraphs $G[V_1], G[V_2], \ldots, G[V_r]$ are called **components** of $G$. If $G$ has exactly one component, $G$ is connected.

- The **degree** $deg_G(v)$ of a vertex $v$ in $G$ is the number of edges of $G$ incident to $v$. It is a fundamental fact that half of the sum over all the vertex degrees is equal to the number of edges in $G$. This fact will be used in the last chapter.

- The **adjacency matrix** of a graph $G = (V, E)$ is the $|V| \times |V|$ matrix $A(G) = (a_{ij})$, in which $a_{ij}$ is the number of edges joining $v_i$ and $v_j$. If $G$ has no loops, then $A(G)$ has zeros entries on the diagonal.

- A subset $M$ of $E$ is called a **matching** in $G = (V, E)$ if no two edges of $M$ are incident to the same vertex.

- A **b-matching** of $G = (V, E)$ is an assignment of integers to the edges of $G$ so that the sum of the weights on the edges incident to a vertex $v$ is at most $b_v$ ($b$ denotes the vectors of $b_v$'s). When $b_v = 1$ for all vertices $v$ in G, then $b$-matchings are the usual matchings.

- An $f$-**factor** is a subgraph of $G$ with degree $f_i$ at the ith vertex for $i = 1, 2, \ldots, n$.

- A $(g, f)$-**factor** is a subgraph of $G$ with degree $d_i$ at the ith vertex, where $g_i \leq d_i \leq f_i$, for $i = 1, 2, \ldots, n$.

- A vertex $v$ is $M$-**matched** if there is some edge of $M$ incident to $v$, otherwise $v$ is $M$-**unmatched**.

- An $M$-**alternating path** in $G$ is a path whose edges are alternately in $E \backslash M$ and in $M$.

- An $M$-**augumenting path** in $G$ is an $M$-alternating path whose origin and terminus are $M$-unmatched. In general, we just say it is an augumenting path if no confusion may occur.

If $M$ is a $b$-matching in $G$, this concept can be generalized to $M$-**augumenting Walk**. If the given graph is not a weighted graph, we usually call a matching (or $b$-matching) $M$ a cardinality matching (or $b$-matching), otherwise we call it a weighted matching (or $b$-matching).

All of the optimization problems discussed here are finite and thus enumeration would always find an optimum solution, but with increasing problem size, enumeration is impractical. At present it is well accepted that a practical algorithm should have its elementary computation steps bounded by a polynomial in term of the problem size. We call such algorithm a **polynomial algorithm**, which were called *good algorithms* by Edmonds in 1965.

For graph related problems, a good measure of problem size is their natural sizes such as the number of vertices, or the number of edges or the size of weighted vectors ... etc. A polynomial algorithm for a graphical problem is called a **strongly polynomial algorithm** if the polynomial is in term of only the number of vertices and the number of edges in graph G. Further details on Complexity Theory can be found in Papadimitriou and Steiglitz [9].

## 1.3 Outline of The Thesis

Algorithms, complexity analysis, and transformation techniques for the Fixed Size Matching problems are presented in this thesis. The contents of each chapters is summarized

below.

- Chapter Two deals with the fixed size weighted matching problem. A primal-dual algorithm is demonstrated to solve the variable size weighted matching problem in polynomial time. The algorithm has the property that at any step, if there are $p$ edges in the matching, then those $p$ edges constitute an optimum matching of size $p$. Thus $p$ is regarded as a variable in the algorithm. Then we show how to use this algorithm to find a fixed size optimum matching, and a small example is worked out. Finally an alternative approach to solve the fixed weighted matching problem is also given.

- Chapter Three tackles the fixed size weighted b-matching problem. First, we rewrite Pulleyblank's [8] blossom algorithm for the weighted $b$-matching problem to obtain experience . Then we transform our fixed size weighted $b$-matching problem into a weighted $b$-matching problem which thus can be solved by the blossom algorithm.

- Chapter Four is devoted to fixed size $(g, f)$-factor problems. We first review Anstee's [1] $f$-factor algorithm, then apply his ideas to our fixed size $(g, f)$-factor problems. We consider our approach a direct one and hence perhaps more practical than a transformation approach. We can either find a fixed size $(g, f)$-factor or display a fixed size $(g, f)$-barrier showing that no fixed size $(g, f)$-factor exists and do this in $O(n^3)$ operations, where $n$ is the number of vertices of a multigraph. At the end we prove an augumenting walk theorem which leads to a conclusion that the size of all feasible $(g, f)$-factors forms an interval, that is, we can start with a $(g, f)$-factor found by any known algorithm and apply our augumenting (or decreasing) walk algorithm to achieve the fixed size $p$ if a $(g, f)$-factor of size $p$ exists. In particular, we can find the largest and smallest cardinality $(g, f)$-factor.

# Chapter 2

# Fixed Size Weighted Matching

## 2.1 Preliminaries

Given a weighted graph $G = (V, E, c)$ with weight vector $c = (c_{ij} : \{i,j\} \in E)$ and $|V| = n$, the general weighted matching problem is to find a matching of $G$ with the largest possible sum of weights. The problem can also be stated as an integer program as below.

$$\text{Max } c^T x$$

subject to:

$$\begin{aligned} \textstyle\sum_{j=1}^{n} x_{ij} &\leq 1, \qquad \forall i \in V \\ x_{ij} &\in \{0, 1\}, \qquad \forall x_{ij} \in E \end{aligned}$$

where $x_{ij} = 1$ iff edge $\{i,j\}$ is in the matching. We cannot ignore these explicit integer constraints $x_{ij} \in \{0,1\}$ and solve the problem via linear program since we may obtain a nonintegral solution which does not correspond to a matching. See Figure 2.2 for an example. If we let $c \equiv 1$ on our pentagon, then the unique maximum weighted solution without integer constraints is $x_{12} = x_{23} = x_{34} = x_{45} = x_{51} = 0.5$, which has weight 2.5. However the weight of a maximum weighted integer matching is clearly 2.

It is well known that the odd cycles of a graph cause this problem. Edmonds [5] found that the following set of linear constraints are sufficient to ensure integrality. For any subset of $2r+1$ vertices there can be at most $r$ matched edges with both ends in the

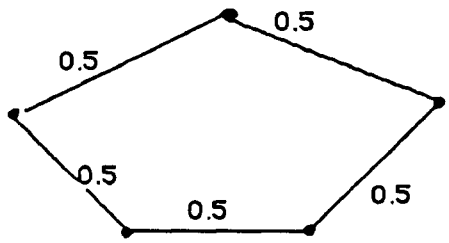


Figure 2.2:

subset. This leads to a linear programming (LP) without explicit integrality constraints whose associated polyhedron has only integer vertices. The LP formulation is:

$$\text{Max } c^T x$$

subject to:

$$\sum_{j=1}^{n} x_{ij} \leq 1, \quad \forall i \in V \tag{1}$$

$$\sum_{i,j \in S_k} x_{ij} \leq s_k, \quad \forall S_k \subset V, |S_k| = 2s_k + 1 \tag{2}$$

$$x_{ij} \geq 0, \quad \forall x_{ij} \in E, \tag{3}$$

We choose $S_k$ to be any subset of $\{1, 2, \ldots, n\}$ which has cardinality that is odd and greater than one, so we have $N = 2^{n-1} - n$ subsets in total. One may not feel comfortable with these exponentially large number of new constraints. However, this causes no trouble for Edmonds' primal-dual algorithm in which all but a polynomial number of the dual variables have zero value throughout the algorithm. Before moving on we need to review some result and concepts about the cardinality matching problem.

**Theorem 2.1** (Berge 1957): A matching $M$ is of maximum cardinality if and only if there exists no augumenting path.

The proof is not hard, but we will skip it because it is a special case of the augumenting walk theorem that we prove in Chapter 4.

**Definitions:**

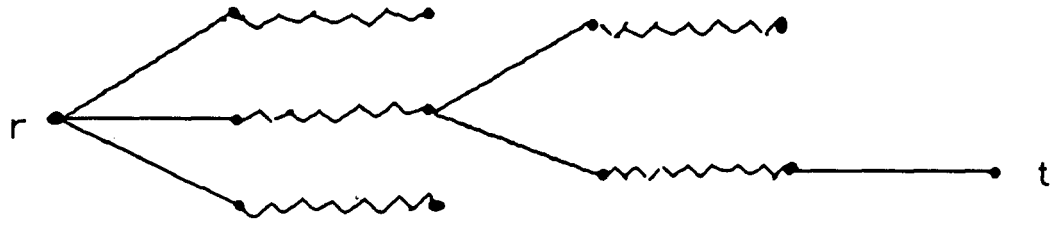


Figure 2.3: An example of augumenting tree

An **alternating tree** w.r.t. a matching $M$ is a tree that satisfies conditions 1, 2 and 3 below.

1. Exactly one vertex of the tree is unmatched, which is called a base (or root) of the tree.

2. All paths from the base are alternating paths.

3. All maximal paths from the the base are of even length.

4. At least one path from the base is an augumenting path.

A tree which satisfies conditions 1, 2 and 4 is called an **augumenting tree** (see Figure 2.3). Throughout this thesis we always use straight lines to denote the edges not in the matching and zigzag lines to denote the edges in the matching. An **inner vertex** of an alternating(or augumenting) tree is a vertex which is at the end of a path of odd length from the root, and an **outer vertex** of an alternating (or augumenting) tree is a vertex which is at the end of a path of even length from the root.

A **blossom** $B$ w.r.t. a given matching $M$ is an odd cycle consisting of $2r + 1$ edges and $2r + 1$ vertices, where $r$ of the edges are in $M$.

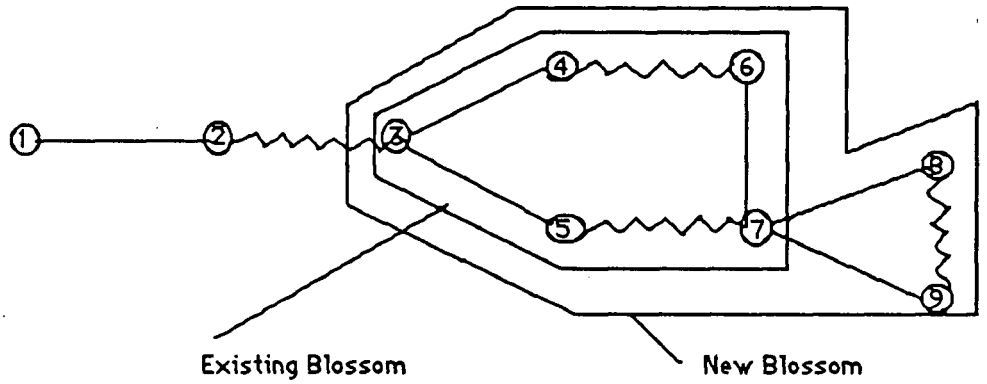


Figure 2.4: An example of blossom construction

In the cardinality matching algorithm, when a blossom is found it is shrunk into one vertex which is called **pesudovertex**. We could have a nest of blossoms, and a blossom not contained within some other blossom is called an **outermost blossom**. An example is given in Figure 2.4.

A **blossom tree** is an alternating tree with one additional edge joining two outer vertices of the tree.

A **Hungarian tree** is an alternating tree for which any edge incident to an outer vertex is also incident to another vertex of the tree.

The cardinality matching algorithm is initialized by specifying an initial matching, possibly empty, and proceeds by searching for an augumenting path. A tree rooted at an unmatched vertex r is grown by adding non-matched and matched edges alternately, shrinking blossoms whenever encountered. Finally we either find an augumenting path or end up with a Hungarian tree. In the first case we can produce a new matching with cardinality increased by one by tracing the augumenting path (expanding a pseudovertex, if discovered, into the original blossom) back to the root and then interchanging the role

of matched and non-matched edges along the path. In the second case, we know that no augumenting paths starting from $r$ will exist any more. Clearly one iteration of the algorithm takes $O(|E|)$ operations since we need only to look at each edge O(1) times. To find a maximum cardinality matching, we only need to grow $n$ trees, so the total complexity is $O(|V||E|)$.

In this chapter we add a size constraint to the weighted matching problem, and we explore the primal-dual procedure with an extra dual variable $\alpha_0$. Our algorithm has the property that at any step if there are $p$ edges in the matching, then these $p$ edges form an optimum matching of $p$ edges. Urquhart [10] (1967) briefly explained a weighted matching algorithm with this property using geometric intuition in his Ph.D. thesis; this algorithm is actually the same as the one in Lawler's book [6], (1976). However another primal-dual version of the weighted matching algorithm given by Papadimitriou and Steiglitz [9] doesn't have this property, and one can find a counterexample with no difficulty.

## 2.2 Variable Size Weighted Matching

We add a size constraint to the weighted matching problem and formulate the new problem into LP form as:

**(P):**

$$\text{Max } c^T x$$

subject to:

$$\alpha_0 : \quad \sum_{i<j} x_{ij} = p \tag{4}$$

$$\alpha_i : \quad \sum_{j=1}^{n} x_{ij} + y_i = 1, \quad \forall i \in V \tag{5}$$

$$r_k : \quad \sum_{i,j \in S_k} x_{ij} + z_k = s_k, \quad \forall S_k \subset V, |S_k| = 2s_k + 1 \tag{6}$$

$$x_{ij} \geq 0, \quad y_i \geq 0, \quad z_k \geq 0, \quad \forall i, j, k.$$

Here $p$ is any positive integer which can be thought as an variable, $y_i, z_k$ are slack variables, and $\alpha_0, \alpha_i, r_k$ are the corresponding dual variables. The dual $(DP)$ of the above $(P)$ is:

**(DP):**

$$\text{Min } p\alpha_0 + \sum \alpha_i + \sum s_k r_k$$

subject to:

$$\forall \{i,j\} \in E : \alpha_0 + \alpha_i + \alpha_j + \textstyle\sum_{i,j \in S_k} r_k \geq c_{ij} \quad (7)$$

$$\alpha_0 \quad \text{free}; \quad \alpha_i \geq 0 \quad \forall i; \quad r_k \geq 0 \quad \forall k.$$

Now we define

$$J_e = \{e = \{i,j\} : \alpha_0 + \alpha_i + \alpha_j + \textstyle\sum_{i,j \in S_k} r_k = c_e\} \quad (8)$$

$$J_b = \{k : r_k = 0\} \quad (9)$$

$$J_m = \{i : \alpha_i = 0\} \quad (10)$$

Here $J_e$ is called a set of **admissible edges,** and $J_b$ is called a set of **admissible odd sets.** We use $\overline{J}_b$ to denote the set of all odd sets not in $J_b$. Our algorithm will solve the problem by starting from the feasible dual solution

$$\pi : \begin{cases} \alpha_0 = max\{c_{ij}\} & \\ \alpha_i = 0 & \forall i \in V \\ r_k = 0 & \forall k \end{cases}$$

Then we consider the following restricted primal (RP).

**(RP):**

$$\text{Max } -x_0^a - \sum x_i^a - \sum x_{n+k}^a$$

subject to:

$$\sum_{i<j} x_{ij} + x_0^a = p \qquad (11)$$

$$\sum_{j=1}^{n} x_{ij} + y_i + x_i^a = 1 \qquad (12)$$

$$\sum_{i,j \in S_k} + z_k + x_{n+k}^a = s_k \qquad (13)$$

$$x_{ij} = 0 \quad \forall \{i,j\} \in \overline{J}_e \qquad (14)$$

$$y_i = 0 \quad \forall i \in \overline{J}_m \qquad (15)$$

$$z_k = 0 \quad \forall S_k \in \overline{J}_b \qquad (16)$$

$$x_{ij} \geq 0 \quad \forall i,j \in V(G) \quad x^a \geq 0$$

Note that (RP) is in fact a Phase I LP, and the $x^a$'s are Phase I artificial variables. We know by the Complentary Slackness (C.S.) that a feasible $x = (x_{ij} : \{i,j\} \in E)$ in (P) is optimal if and only if it satisfies

**(C.S.):**

$$\sum x_{ij} = p \quad \alpha_0 \neq 0 \qquad (17)$$

$$x_{ij} = 0 \quad \forall \{i,j\} \notin J_e \qquad (18)$$

$$y_i = 0 \quad \forall i \notin J_m \qquad (19)$$

$$z_k = 0 \quad \forall S_k \notin J_b \qquad (20)$$

So we attempt to solve (RP); if we find an optimal solution of (RP) with zero value then this solution is feasible in (P) and satisfies the above C.S. conditions (17-20). Thus it must be an optimal solution of (P). If we find the optimal solution of (RP) is strictly negative then we will adjust our initial feasible dual solution $\pi$ to a new feasible dual solution $\pi'$ so that the new (RP) induced by $\pi'$ will have an improved optimal solution. We keep performing this cycling process until we find an optimal solution of (RP) with zero value or we end up with an optimal solution of (RP) with nonzero value which implies that no matchings of size $p$ exist. Note our algorithm always preserves primal and dual feasibility as well as the C.S. conditions, except (4). When the algorithm terminates, if (4) is satisfied then we obtain an optimum matching of size $p$, otherwise we conclude

there doesn't exist a matching of size $p$. Before solving (RP), we need to present its dual as follows.

**(DRP):**

$$\text{Min } p\overline{\alpha}_0 + \sum \overline{\alpha}_i + \sum s_k \overline{r}_k$$

subject to:

$$\forall e = \{i,j\} \in J_e : \quad \overline{\alpha}_0 + \overline{\alpha}_i + \overline{\alpha}_j + \textstyle\sum_{i,j \in S_k} \overline{r}_k \geq 0 \quad (21)$$

$$\overline{\alpha}_0 \geq -1; \quad \overline{\alpha}_i \geq -1; \quad \overline{r}_k \geq -1;$$

$$\overline{\alpha}_0 \geq 0 \quad \forall i \in J_m; \quad \overline{r}_k \geq 0 \quad \forall k \in J_b.$$

We solve (RP) as maximum cardinality matching problem and preserve the following assumptions:

(a): $x_e \in \{0,1\}$, i.e., $x$ corresponds to a matching $M$.

(b): $r_k > 0 \Rightarrow G[S_k]$ has precisely $s_k$ edges of $M$.

(c): $r_i, r_j > 0$ and $S_i \bigcap S_j \neq \emptyset \Rightarrow S_i \subseteq S_j$ or $S_j \subseteq S_i$.

We work in an **admissible graph** $G_J$ corresponding to $J_e$ and $J_b$. Graph $G_J$ consists of the graph (V, $J_e$) after shrinking all odd sets in $\overline{J}_b$. We call a matching of $G_e = (V, J_e)$ **proper** if all odd set in $\overline{J}_b$ are full (assumption (b) holds). A maximum proper matching of $G_e$ need not be a maximum matching of $G_e$. The following lemma ensures that we can find a maximum proper matching in $G_e$ by finding a maximum matching in $G_J$.

**Lemma 2.2:** There is a matching in $G_J$ with $d$ unmatched vertices iff there is a proper matching in $G_e$ with $d$ unmatched vertices.

**Proof:** From a proper matching in $G_e$, we can go to a matching in $G_J$ by simply shrinking maximal odd sets in $\overline{J}_b$; this doesn't change the number of unmatched vertices since we start with a proper matching in which all odd sets in $\overline{J}_b$ are full. Conversely, a matching in $G_J$ can be turned into a proper matching of $G_e$ by expanding odd sets and filling them with matching edges as appropriate. Since each odd set comes from

the shrinking of a nest of blossoms, an odd set is unmatched only if some blossom of those contains an unmatched vertex. An alternating path from the unmatched vertex will allow any specified vertex in the odd set to be unmatched when the blossoms are expanded; the lemma then follows.

**Q.E.D.**

Now we can apply cardinality matching algorithm on $G_J$ to find a maximum cardinality matching, but how is this matching related to the optimal solution of (RP)?

**Theorem 2.3:** An optimal solution $\{x_{ij}\}$ of (RP) is just a maximum proper matching of $G_e$, namely, a maximum cardinality matching of $G_J$.

**Proof:** Suppose we have found a maximum matching of $G_J$ by the cardinality matching algorithm. Thus we have failed to discover an augumenting path in a current graph $G_c$, resulting from $G_J$ by shrinking a number of blossoms (be aware of the sequence $G_e = (V, J_e) \longrightarrow G_J \longrightarrow G_c$). The vertex set of $G_c$ consists of pseudovertices. A pseudovertex can be any of following:

1. A vertex of V.

2. A maximal odd set in $\overline{J}_b$.

3. Several vertices of V and maximal odd sets in $\overline{J}_b$ merged into an outermost blossom.

A pseudovertex of $G_c$ can be either outer or inner or neither. Let O be the vertex set of outer pseudovertices, and I be the vertex set of inner pseudovertices. We then partition the vertices of $G_c$ that are not vertices of V into $\Phi_O$ ( maximal odd sets corresponding outer pseudovertices or blossoms), $\Phi_I$ (maximal odd sets corresponding to inner pseudovertices), and the rest.

LP theory says if a primal feasible solution and a dual feasible solution both have the same objective value, then both are optimal. We are going to prove the optimality of

(RP) by exhibiting a feasible solution of (DRP) that achieves the same cost. We define our solution of (DRP) as:

$\overline{\pi} = (\overline{\alpha}_0, \overline{\alpha}_1, \ldots, \overline{\alpha}_n, \overline{r}_1, \overline{r}_2, \ldots, \overline{r}_N)$ where

$$\overline{\alpha}_0 \equiv -1$$

$$\overline{\alpha}_i = \begin{cases} 0 & i \in O \\ 1 & i \in I \\ 1/2 & \text{else} \end{cases} \qquad \overline{r}_k = \begin{cases} 1 & S_k \in \Phi_O \\ -1 & S_k \in \Phi_I \\ 0 & \text{else.} \end{cases}$$

This definition of $\overline{\pi}$ is different from that in the primal-dual algorithm given in [6] and [9]. Why is this feasible? The constraint (21) can be violated only if $v_i, v_j \in O$, but this means $v_i$ and $v_j$ belong to the same outer pseudovertex (they cannot belong to different outer pseudovertices since then the two pseudovertices would form a blossom), hence $\overline{r}_k = 1$ offsets $\overline{\alpha}_0 = -1$ and salvages its validity. All the other constraints of (RP) are obviously satisfied.

Why is this optimal? With the assumption (a), we may forget about the constraint (12) in (RP). Since if $i \in \overline{J}_m$, vertex $i$ is matched, $y_i = x_i^a = 0$. If $i \in J_m$, we can let $y_i = 1$ to fill the gap and still have $x_i^a = 0$. Thus we can always let $x_i^a = 0$. Similarly, with assumption (b) we may forget about constraint (13) in (RP) and always assume $x_{n+k}^a = 0$. Now consider the objective value of (RP):

$$\begin{aligned} & -x_0^a - \sum x_i^a - \sum x_{n+k}^a = -x_0^a \\ & = -(p - \text{number of edges in the matching}) \\ & = -(p - \sum \overline{\alpha}_i - \sum s_k \overline{r}_k) \\ & = \overline{p}_0 + \sum \overline{\alpha}_i + \sum s_k \overline{r}_k. \end{aligned}$$

Therefore our $\overline{\pi}$ is the optimal solution of (DRP), and the theorem is proved. **Q.E.D.**

**Remark:** If there are $p$ edges in the current matching then $x_0^a = 0$. All feasibility and C.S. conditions are satisfied, so these $p$ edges constitute an optimum matching of size $p$.

What if the number of edges in the current matching is less than $p$? We then need to adjust our dual solution $\pi$ to $\pi'$ such that $\pi' = \pi + \theta\overline{\pi}$ where $\theta$ is determined as follows:

$$\theta = Min(\delta_1, \delta_2, \delta_3)$$

$$\begin{aligned} \delta_1 &= \alpha_0 + \alpha_i + \alpha_j - c_{ij} && \forall i, j \in O \\ \delta_2 &= 2(\alpha_0 + \alpha_i + \alpha_j - c_{ij}) && \forall i \in O, j \notin O \bigcup I \\ \delta_3 &= r_k && \forall S_k \in \Phi_I \end{aligned}$$

Our definition of $\theta$ ensures the feasibility of $\pi'$, we then solve a new (RP) induced by $\pi'$. We repeat the same procedure and each time obtain an improved solution. But how do we know that the algorithm will eventually terminate? Let us examine the complexity of our algorithm.

**Theorem 2.4:** The complexity of our algorithm is $O(pn^3)$.

**Proof:** Let's call each search for an augumenting path a step, and the sequence of steps between two successive augumentations a stage. Clearly there can be at most $p$ stages. Now let's bound the number of steps in each stage.

A step can only be one of the three types, depending on whether $\theta = \delta_1, \theta = \delta_2$, or $\theta = \delta_3$. In the first case, two outer vertices of $G_J$ are joined by a new edge, which means that in the next step we shall discover either an augumenting path or a blossom. Since the shrinking of each blossom decreases the number of vertices of the current graph by at least 2, we can have at most $n/2 + 1$ steps of this case in one stage.

If $\theta = \delta_2$, a new outer vertex is found in the next step, which means that the number of steps of this case is also bounded by $n/2$. Finally if $\theta = \delta_3$, we remove from $\overline{J}_b$ an

odd set $S_k$ that corresponds to an inner pseudovertex of $G_J$, so the number of such steps in a stage is clearly bounded by the number of odd sets in $\overline{J}_b$ at the conclusion of the previous stage. But this number is bounded by $n/2$ by assumption (c).

Now we see we have a total of $O(pn)$ steps, and each step can be carried out in $O(n^2)$ time by the cardinality matching algorithm. The determination of O, I, $\Phi_O$, and $\Phi_I$ can also be done in $O(n^2)$ time, as well as the construction of $G_J$, the calculation of $\theta$ and the variable modification. Therefore the total complexity of our algorithm is $O(pn^3)$.

## 2.3 Fixed Size Weighted Matching

In the primal-dual algorithm given in the previous section, each time we obtain a new dual solution $\pi' = \pi + \theta\overline{\pi}$, we may have one of the following cases before we have to adjust dual variables again.

1. There is no augumenting path found.
2. There is exactly one augumenting path found.
3. More than one augumenting path is found.

When the third case appears, we should pay more attention. If we wants an optimum matching of size p and there are $r < p$ edges in the current matching, suppose we fail to find any more augumenting paths and have to adjust to a new dual solution $\pi^* = \pi + \theta\overline{\pi}$. After doing so, if we suddenly find $q$ augumenting paths simultaneously before we have to adjust dual variables again, and $r + q > p$, then what should we do?

**Theorem 2.5:** We can pick any $p - r$ among these $q$ augumenting paths and still obtain an optimum matching of size $p$.

**Proof:** At the beginning we take the initial dual variable on each vertex to be zero, i.e. $\forall i \in V\ \alpha_i = 0$. In any later steps, each time we look for an augumenting path on

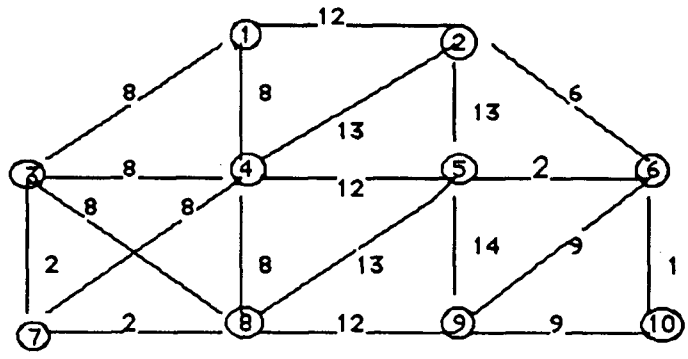


Figure 2.5:

$G_J$ we start from an unmatched vertex $i$ with $\alpha_i = 0$; if an augumenting path can be found, the path must terminate at an outer vertex $j$ also with $\alpha_j = 0$. If any one of $i$ and $j$ is a pseudovertex corresponding some maximal blossom $S_k$, then $S_k$ contains at least one vertex $r$ such that $\alpha_r = 0$. So all of our augumenting paths both start from and end at a vertex with zero dual variable. Therefore, if we pick any $p - r$ among these $q$ augumenting paths we still preserve all the feasibility and C.S. conditions, i.e., we are still at optimality. **Q.E.D.**

We are going to work out an example which may help in understanding our primal-dual algorithm in Section 2.2

**Example:** Find a maximum weighted matching of size 4 in the graph in Figure 2.5.
**Solution:** At each step, a nonzero value of $\alpha_i$ will appear on the corresponding vertex, $\overline{\alpha}_0 \equiv 0$ throughout the algorithm, and admissible graph $G_J$ is presented at every step.

**Step 1:** $\overline{\alpha}_i = 0$ for $i \neq 5, 9$; $\overline{\alpha}_i = 1/2$ for $i = 5, 9$. Also, $\delta_1 = 1$ because of edge {2, 4}; $\delta_2 = 2$ because of edge {5, 8}; $\delta_3 = \infty$ because of $\Phi_I = \emptyset$, and $\theta = 1$. Edge $\{5, 9\}$ is an optimal matching of size 1.

**Step 2:** $\overline{\alpha}_i = 0$ for $i \neq 2, 4, 5, 9$; $\overline{\alpha}_i = 1/2$ for $i = 2, 4, 5, 9$. Also, $\delta_1 = 5$ because of edge {3, 8}; $\delta_2 = 1$ because of edge {5, 8}; $\delta_3 = \infty$ because of $\Phi_I = \emptyset$, and $\theta = 1$. An

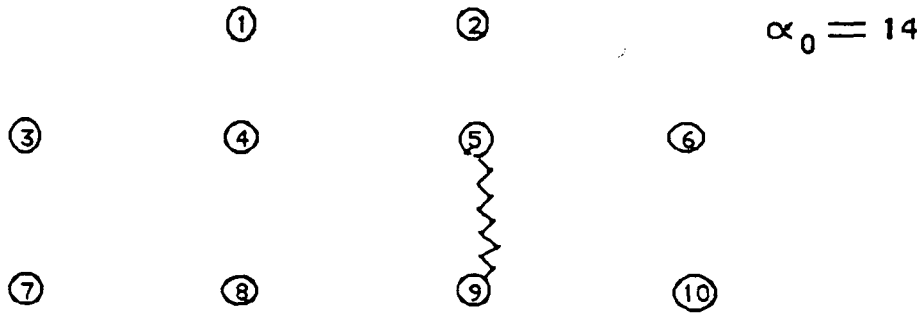


Figure 2.6: Step 1

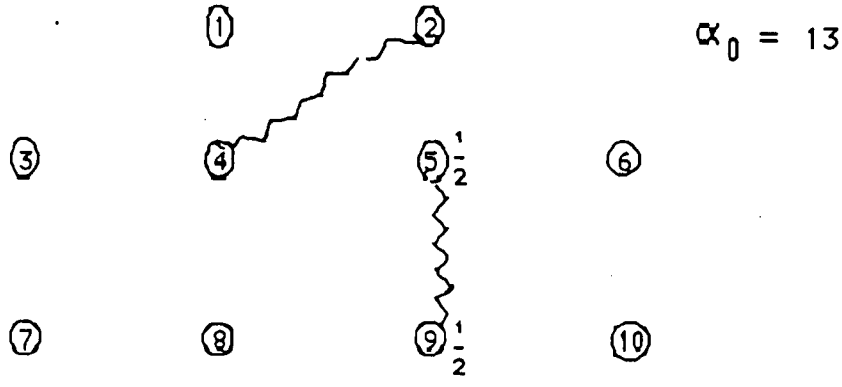


Figure 2.7: Step 2

optimal matching of size 2 is $\{\{2,4\},\{5,9\}\}$.

**Step 3:** $\overline{\alpha}_i = 0$ for $i \neq 2,4,5$; $\overline{\alpha}_i = 1/2$ for $i = 2,4$; $\overline{\alpha}_5 = 1$. Also, $\delta_1 = 1$ because of edge {8, 9}; $\delta_2 = 1$ because of edge {1, 2}; $\delta_3 = \infty$ because of $\Phi_I = \emptyset$, and $\theta = 1$.

**Step 4:** $\overline{\alpha}_i = 0$ for $i \neq 2$; $\overline{\alpha}_2 = 1$; $\overline{r}_{\{5,8,9\}} = 1$. Also, $\delta_1 = 2$ because of edge {4, 5}; $\delta_2 = \infty$ because of $I = \emptyset$; $\delta_3 = \infty$ because of $\Phi_I = \emptyset$, and $\theta = 2$.

**Step 5:** $\overline{\alpha}_i = 0$ for $i = 3,6,7,10$; $\overline{\alpha}_i = 1/2$ for the other vertices. Also, $\delta_1 = 7$ because of edge {3, 7}; $\delta_2 = 2$ because of edge {1, 3}, {6, 9} and {9, 10}; $\delta_3 = \infty$ because of $\Phi_I = \emptyset$, and $\theta = 2$. An optimal matching of size 3 is $\{\{1,2\},\{4,5\},\{8,9\}\}$.

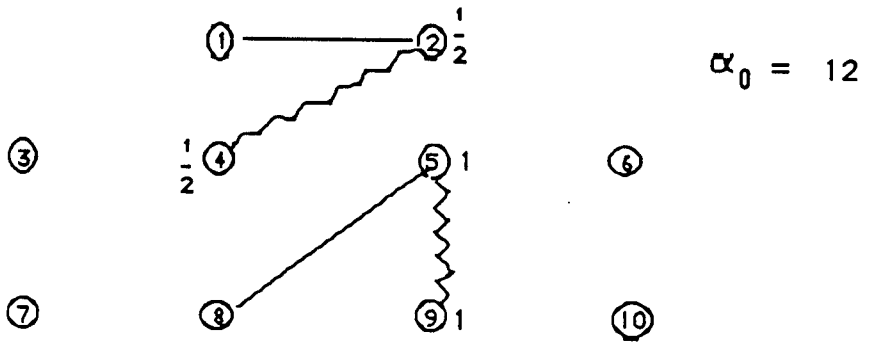


Figure 2.8: Step 3

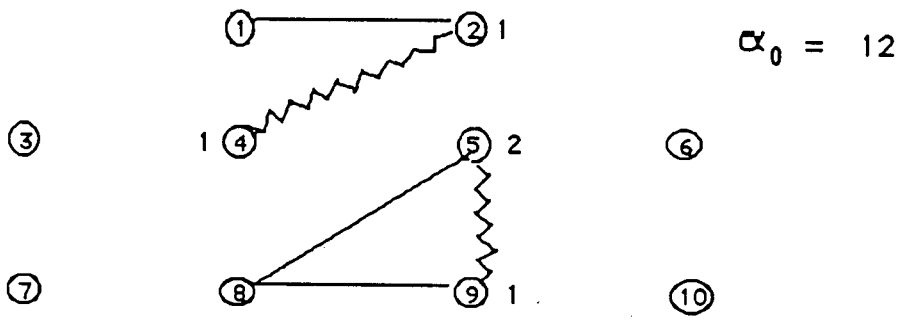


Figure 2.9: Step 4

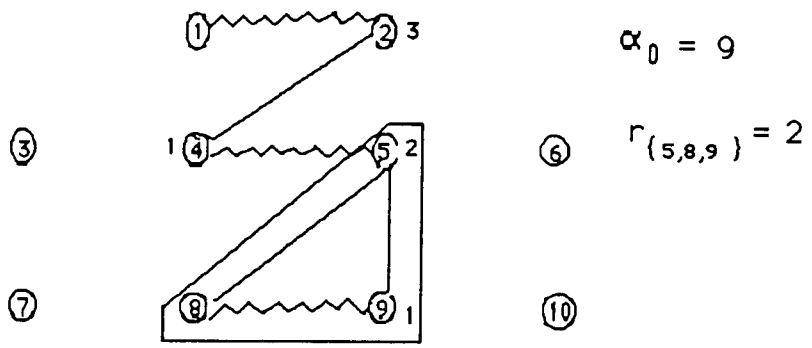


Figure 2.10: Step 5

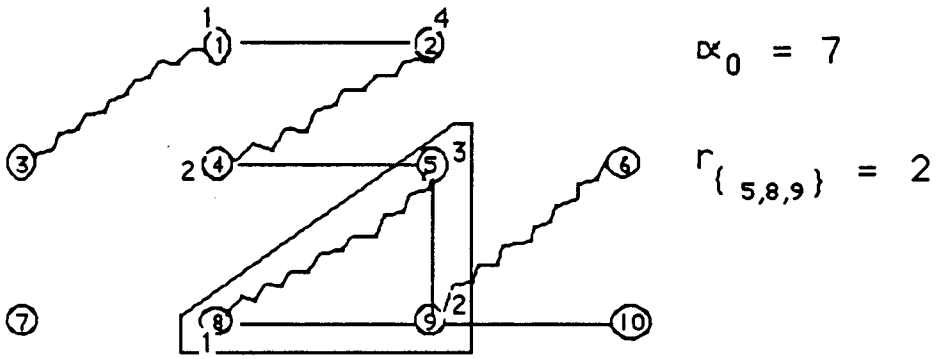


Figure 2.11: Step 6

**Step 6:** $\overline{\alpha}_i = 0$ for $i = 6,7,10$; $\overline{\alpha}_i = 1$ for $i = 5,8,9$; $\overline{\alpha}_i = 1/2$ for $i = 1,2,3,4$; $\overline{r}_{\{5,8,9\}} = -1$. Also, $\delta_1 = 6$ because of edge $\{6, 10\}$; $\delta_2 = 2$ because of edge $\{2, 6\}$; $\delta_3 = 2$ because of $S_k \in \Phi_I$, and $\theta = 2$. We now have a Hungarian tree, so the algorithm stops.

A maximum weighted matching of size 4 is $\{\{\mathbf{1, 3}\},\{\mathbf{2, 4}\},\{\mathbf{5, 8}\},\{\mathbf{6, 9}\}\}$.

## 2.4 A Transformation Approach

In this section we provide an approach which transforms the fixed size weighted matching problem into a general matching problem.

Let $G = (V, E, c)$ be a weighted graph with $|V| = n$. We create a new graph $\overline{G} = (\overline{V}, \overline{E}, \overline{c})$ such that

$$\overline{V} = V \bigcup \{n - 2p \text{ new vertices}\}$$
$$\overline{E} = E \bigcup \{\{s, i\} : i \in V, \text{all new verticess}\}$$
$$\overline{c} = \{\overline{c}_{ij} : \{i,j\} \in \overline{E}\}, \text{ where}$$
$$c_0 = max\{c_{ij} : \{i,j\} \in E\} + 1, \text{ and}$$
$$\overline{c}_{ij} = \begin{cases} c_{ij} & \{i,j\} \in E \\ c_0 & \text{otherwise.} \end{cases}$$

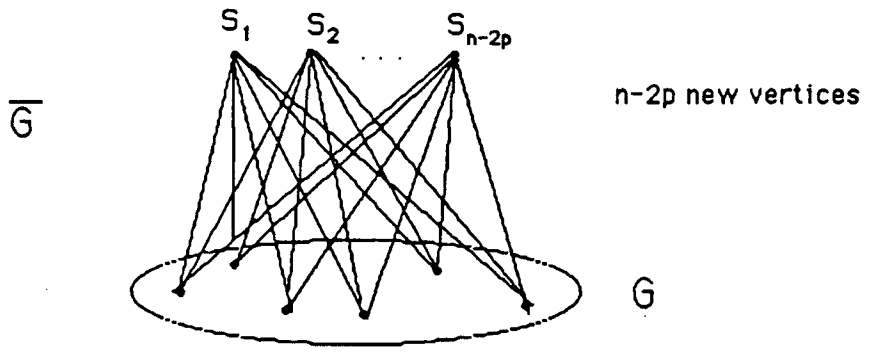


Figure 2.12: An example of graph $\overline{G}$

See Figure 2.12 for an example. Then we can apply any known algorithm to find a full size maximum weighted matching on $\overline{G}$. When this optimum matching is restricted to G, it induces an optimum matching of size $p$ on $G$.

**Remark:** $\overline{G}$ has $n(n-2p)$ more edges than $G$. So $\overline{G}$ will be become very dense for small $p$; we can no longer take advantage of the sparsity (if any) of $G$. This is particularly relevant if the complexity of Theorem 2.4 is more carefullly calculated with respect to the number of edges as well as the number of vertices.

# Chapter 3

# The Fixed Size Weighted $b$-matching Problem

## 3.1 The Weighted $b$-matching Problem

In this section, we introduce the blossom algorithm for the weighted $b$-matching problem, which is due to Pulleyblank [8].

### 3.1.1 Introduction and Definitions

Let $G = (V, E, c)$ be a weighted graph, and $b = (b_i : i \in V)$ be a vector of positive integers. The general weighted $b$-matching problem can be formulated in the following LP form:

**(P):**

$$\text{Max } c^T x$$

subject to:

$$\begin{array}{lll} (3.1)\ \alpha_i: & \sum_{j=1}^{n} x_{ij} \leq b_i & \forall i \in V \\ (3.2)\ r_k: & \sum_{i,j \in R_k} x_{ij} \leq s_k & \forall R_k \subseteq V, \quad \sum_{i \in R_k} b_i = 2s_k + 1 \\ (3.3) & x_{ij} \geq 0 & \forall \{i,j\} \in E. \end{array}$$

Where $\alpha_i$'s and $r_k$'s are the corresponding dual variables.

**Definition 3.1** *A feasible solution $x = (x_{ij} : \{i,j\} \in E)$ of the above (P) is called a* **$b$-matching**.

**Definition 3.2** *Given a $b$-matching $x$, the* **deficiency of a vertex** *$i$ is $b_i - \sum_{j=1}^{n} x_{ij}$ which is denoted by $def_x(i)$. Vertex $i$ is called a* **deficient vertex** *if $def_x(i) > 0$.*

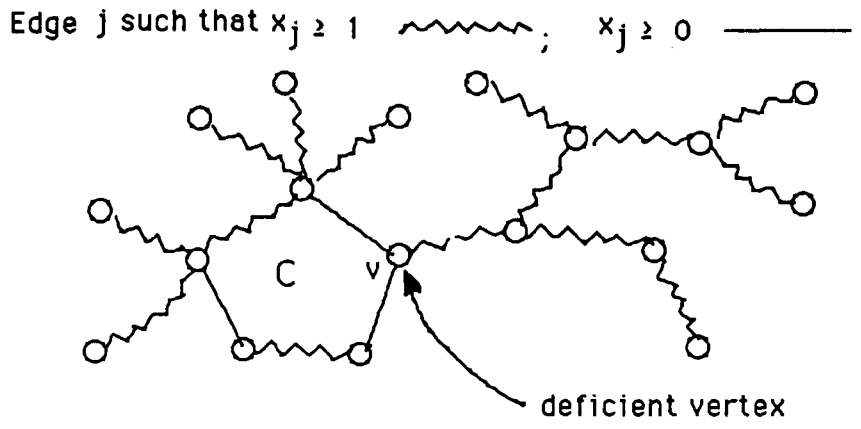


Figure 3.13: Sample Blossom

**Definition 3.3** *A walk* $w = v_0e_1v_1e_2v_2 \ldots e_{2k-1}v_{2k-1}$, *where* $e_i = \{v_{i-1}, v_i\}$, *is an* **augmenting walk** *w.r.t. a b-matching* $x$ *if* $v_0$ *and* $v_{2k-1}$ *are both deficient vertices and* $x_k \geq 1$ *(*$\geq 2$ *if occur twice) if* $k$ *is an even edge of* $w$.

**Definition 3.4** *The* **deficiency of a graph** $G$ *w.r.t. a b-matching* $x$ *and its corresponding dual solution* $y = (\alpha, r)$ *is* $\sum_{y_i>0} def_x(i)$ *which is denoted by* $\Delta(G; x, y)$.

**Definition 3.5** *A* **blossom** *(see Figure 3.13), w.r.t. a b-matching* $x$, *is a connected graph* $B = (V(B), E(B))$ *containing no even cycles, exactly one odd cycle* $C$ *and for which the degree constraints satisfy the following conditions. Let* $v \in V(C)$.

$$(3.4)\quad \sum_{j=1}^{n} x_{ij} = b_i \qquad \forall i \in V(B) - \{v\}$$

$$(3.5)\quad \sum_{j=1}^{n} x_{vj} = b_v - 1$$

$$(3.6)\quad x_j \geq 1 \qquad \forall j \in E(B) - E(C)$$

$$(3.7)\quad x_j \geq 1 \qquad \forall j \in E(C), \text{ and } j \text{ is the even edge of a path in } C \text{ rooted from } v$$

By **shrinking** a blossom $B$ in the graph $G$, we mean that all the edges of $G$ which have both ends in $B$ are contracted. We call the resulting vertex $s$ a **pseudovertex** and define $b_s = 1$.

There is an important property of a blossom, which ensures we can do the shrinking in our blossom algorithm. See the following proposition.

**Proposition 3.1** Let $B$ be a blossom w.r.t. a $b$-matching $x$. Then for any $i \in V(B)$, there exists a new $b$-matching $x'$ that can be obtained from $x$ by an alternating walk such that $B$ is also a blossom w.r.t. $x'$ and $def_{x'}(i) = 1$.

During the course of the blossom algorithm we construct forests having special properties. Let $T$ be a tree contained in G, and $r \in V(T)$ be the root of $T$. Then an edge $\{i,j\} \in E(T)$ is **even** or **odd** according as $\{i,j\}$ is the last edge of the path in $T$ rooted from $r$ to an outer or inner vertex of $T$.

**Definition 3.6** *We call $T$ an* **alternating tree** *w.r.t. a $b$-matching $x$ (see Figure 3.14) if*

$$\begin{aligned}
&(3.8) \quad \textstyle\sum_{j=1}^{n} x_{rj} < b_r \\
&(3.9) \quad \textstyle\sum_{j=1}^{n} x_{ij} = b_i \quad \forall i \in V(T) - \{r\} \\
&(3.10) \quad \{i,j\} \in E(T) \quad \forall x_{ij} > 0 \text{ and } i \text{ or } j \in V(T) \\
&(3.11) \quad x_j > 0 \qquad\quad \forall j \text{ is an even edge of } T.
\end{aligned}$$

We call a nonempty collection of alternating trees an **alternating forest**.

**Definition 3.7** *Let $k$ be an edge of a tree $T$ with root $r$. If we delete $k$ from $T$ then the resulting graph will consist of two trees, one of which, $T'$, will not contain $r$. We call $T'$ the* **portion** *of $T$ above $k$.*

**Definition 3.8** *A* **Hungarian Forest** *is an alternating forest that can not grow any more.*

### 3.1.2 Characterization of the Optimal Solution

First let's consider the dual problem (D) of the primal $b$-matching problem (P).

(D):

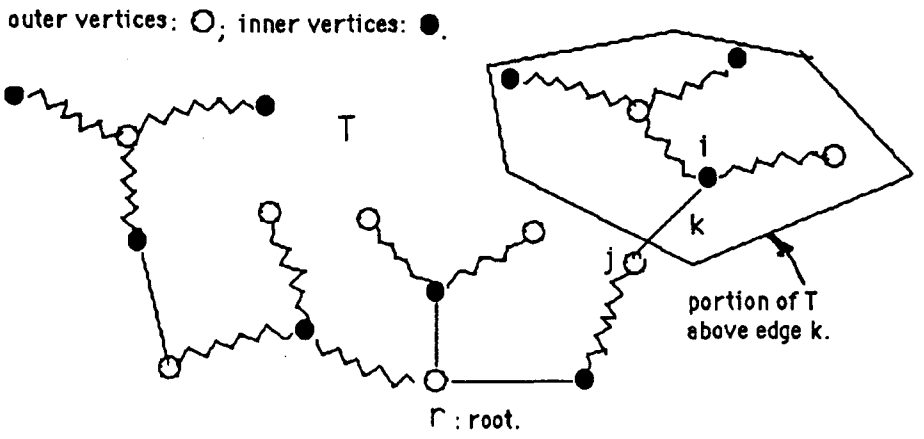


Figure 3.14: Alternating Tree

$$\text{Min } \sum b_i\alpha_i + \sum s_k r_k$$

subject to:

$$(3.12)\quad \alpha_i + \alpha_j + \textstyle\sum_{i,j\in R_k} r_k \geq c_{ij} \qquad \forall\{i,j\} \in E(G)$$

$$(3.13)\quad \alpha_i \geq 0 \qquad \forall i \in V$$

$$(3.14)\quad r_k \geq 0 \qquad \forall k.$$

The blossom algorithm is also a primal dual procedure: We work on an admissible graph $G_J$ (see Chapter 2 for the definitions of $G_J$, $J_e$, $J_m$, $J_b$, etc.) and find a maximum cardinality $b$-matching $x$ on G. Then we check if $\Delta(G; x, y) = 0$. If so, we are at optimality; if not, we adjust our dual solution $y = (\alpha_i, r_k)$ to $y'$ and obtain a new $G_J$ on which we can find an improved maximum cardinality $b$-matching $x'$. We continue this cycling process until we terminate with $\Delta(G; x, y) = 0$.

We state the C.S. conditions of our problem as follows:

$$(3.15)\quad x_{ij} > 0 \Longrightarrow \alpha_i + \alpha_j + \textstyle\sum_{i,j\in R_k} r_k = c_{ij} \qquad \forall\{i,j\} \in E$$

$$(3.16)\quad \alpha_i > 0 \Longrightarrow \textstyle\sum_{j=1}^{n} x_{ij} = b_i \qquad \forall i \in V$$

$$(3.17)\quad r_k > 0 \Longrightarrow \textstyle\sum_{i,j\in R_k} x_{ij} = s_k \qquad \forall R_k$$

The blossom algorithm always preserves all the primal and dual feasibilities, as well as C.S. conditions except (3.16). At each stage of the algorithm, we have a $b$-matching $x = (x_{ij} : \{i,j\} \in E)$ and a feasible dual solution $y = (\alpha_i, r_k : \forall i, R_k)$.

Define $G^+(x) = (V(G), E^+(x))$ where $E^+(x) = \{\{i,j\} \in E(G_J) : x_{ij} > 0\}$, and let $H$ be any component of $G^+(x)$. Then $H$ has the following properties:

(3.18) $H$ contains no even cycles;

(3.19) $H$ contains at most one odd cycle;

(3.20) if $H$ contains an odd cycle, then $H$ contains no deficient vertices;

(3.21) if $H$ contains no cycles, then $H$ has at most one deficient vertex.

We also have an alternating forest $F$ contained in $G_J$ such that

(3.22) Each $i \in V$ such that $\sum_{j=1}^{n} x_{ij} < b_i$ is the root of a tree in $F$.

Forest $F$ is partitioned into $F^0$ and $F^1$: $F^0$ consists of all those trees in $F$ such that $\alpha_r = 0$ if the root $r \in V$ or $\alpha_i = 0$ if the root $r \subset V$ is a pseudovertex and $i \in r$; and $F^1$ consists of all other trees of $F$. Then

(3.23) $\Delta(G; x, y) = \sum_{\alpha_i > 0} def_x(i) = \sum(def_x(i) : i \text{ is the root of a tree of } F^1)$

It will be seen in the algorithm that as long as there are vertices in $F^1$, we are not at optimality, and as soon as $V(F^1) = \emptyset$, namely $\Delta(G; x, y) = 0$, we implictly have an optimal solution.

### 3.1.3 The Blossom Algorithm

(3.24) Initially we may take $x_{ij} = 0 \quad \forall \{i,j\} \in E$, $\alpha_i = 1/2 \max \{c_{ij} : \{i,j\} \in E\}$, and $r_k = 0 \ \forall R_k$. $F$ will be the spanning forest of $G$ in which every tree consists of a single outer vertex.

**Part One: Cardinality Matching Algorithm.**

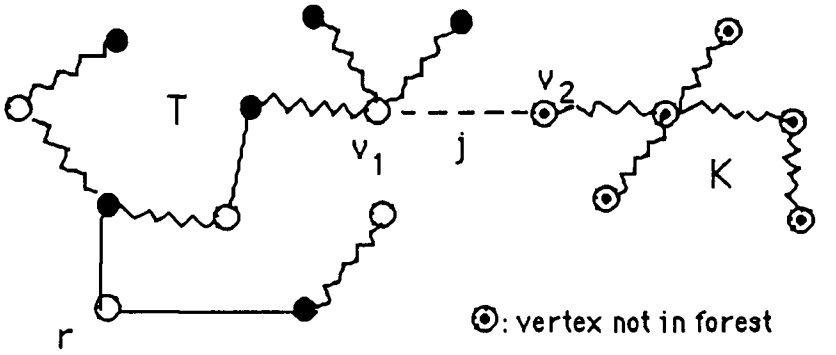


Figure 3.15: Forest Growth

**Step 1:** Scan $E(G_J)$ to find an edge $j = \{v_1, v_2\}$ joining an outer vertex $v_1$ of $F^1$ to a vertex $v_2$ that is not an inner vertex of $F^1$. If no such edge exists, then the current forest is Hungarian; go to Step 8. Otherwise, go to Step 2.

**Step 2:** Examining Vertex $v_2$.

If $v_2$ is in a component of $G^+(x)$ which is not contained in $F$, go to Step 3.

If $v_2$ is an outer vertex of a tree in $F$ which is different from the tree containing $v_1$ then go to Step 4.

If $v_1$ and $v_2$ belong to the same tree of $F$, go to Step 5.

If $v_2$ is an inner vertex of a component of $F^0$, go to Step 7.

**Step 3** (Grow Forest $F$): Let $K$ be the component of $G^+(x)$ containing $v_2$. If $K$ contains a cycle, go to Step 3b.

**Step 3a** (see Figure 3.15): If $K$ contains no cycle, we grow the alternating tree $T$ containing $v_1$ by attaching $v_2$ and $K$ by means of the edge $j$. Go to Step 1.

**Step 3b** (see Figure 3.16): $K$ contains an odd cycle $C$. Let $w_1$ be a vertex of $C$ which is an odd distance from $v_2$ in $K$ and for which this distance is as short as possible. Let $w_2$ be a vertex of $C$ adjacent to $w_1$ in $C$ which is no closer to $v_2$ in $C$ than $w_1$. Let $k$ be the edge of $C$ joining $w_1$ and $w_2$, and $K'$ be the tree obtained from $K$ by removing

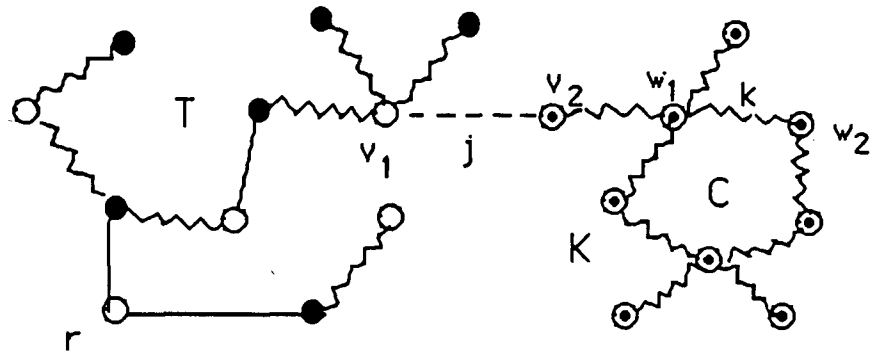


Figure 3.16: Addition of Cycle to Forest

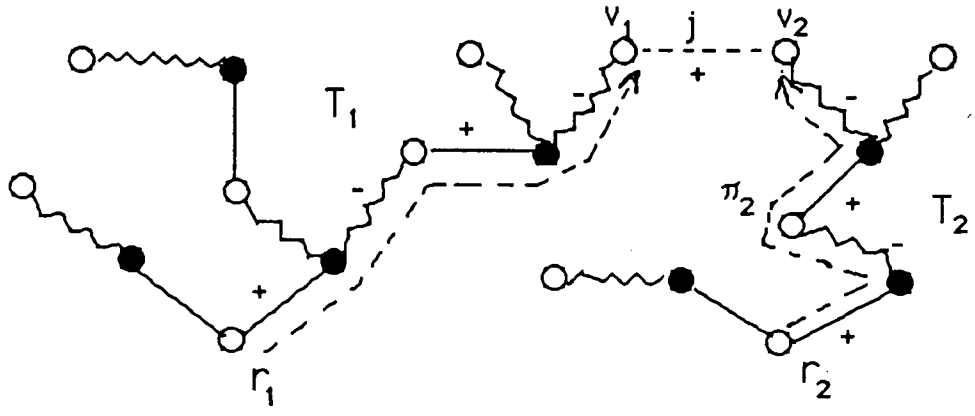


Figure 3.17: Two Tree Augmentation

the edge $k$. Add $K'$ to the forest by using edge $j$ as described in Step 3a. Go to Step 5.

**Step 4** (see Figure 3.17): Augmentation (Two Trees).

**Step 4a:** Calculation of $\sigma$.

Let $r_1$ ($r_2$) be the root of the tree $T_1$ ($T_2$) of $F^1$ ($F$) containing $v_1$ ($v_2$). Let $\sigma_1 = \min \{x_k\}$ where $k$ is an even edge of the path $\pi_1$ in $T_1$ from $r_1$ to $v_1$; let $\sigma_2$ and $\pi_2$ be analogously defined for $T_2$, $v_2$ and $r_2$. By (3.11) $\sigma_1, \sigma_2 \geq 1$. Let $\sigma \equiv \min \{\sigma_1, \sigma_2, def_x(r_1), def_x(r_2)\}$. By (3.8), $\sigma \geq 1$.

**Step 4b:** Augmentation.

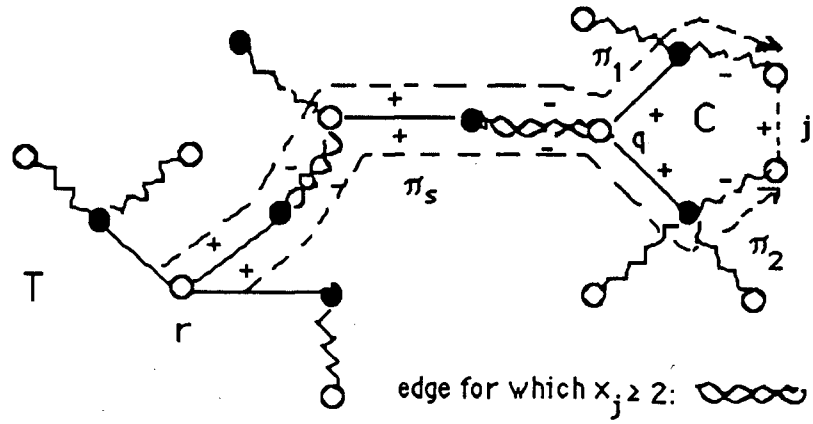


Figure 3.18: One Tree Augmentation

Define $x'$ to be the new $b$-matching obtained from $x$ by subtracting $\sigma$ from $x_k$ if $k$ is an even edge of $\pi_1$ or $\pi_2$ and adding $\sigma$ to $x_k$ if $k$ is an odd edge of $\pi_1$ or $\pi_2$, or $k = j$. Then

$$\Delta(G_J; x', y) \leq \Delta(G_J; x, y) - 1.$$

**Step 4c:** Computation of new $F$.

If $def_{x'}(r_1) = 0$ (resp. $def_{x'}(r_2) = 0$) then remove $T_1$ (resp. $T_2$) from $F$. If $k$ is an even edge of $\pi_1$ or $\pi_2$ for which $x'_k = 0$ then remove $k$ and the portion of the tree above it from $F$. Go to Step 1.

**Step 5** (see Figure 3.18): Augmentation (One Tree).

**Step 5a:** Calculation of $\sigma$ and Blossom Test.

Let $r$ be the root of the tree of $F^1$ containing $v_1$ and $v_2$. Let $\pi_1$ ($\pi_2$) be the path in T from $r$ to $v_1$ ($v_2$). Let $\pi_s$ be the common position of $\pi_1$ and $\pi_2$. Then $E(\pi_1) \cup E(\pi_2) \cup \{j\} \setminus E(\pi_s)$ are the edges of an odd cycle $C$.

Define $\sigma_1 = \min \{x_k$: $k$ is an even edge of $\pi_s\}$, and $\sigma_2 = \min \{x_k$: $k$ is an even edge of $\pi_1$ or $\pi_2$ and $k \notin E(\pi_s)\}$. Then $\sigma_1, \sigma_2 \geq 1$. Let

$$\sigma = \min \{[\sigma_1/2], \sigma_2, [def_x(r)/2]\}.$$

Where $[\alpha]$ denotes the largest integer no greater than $\alpha$. If $\sigma \geq 1$ then go to Step 5b where we augment; otherwise, go to Step 6 where we shrink a portion of $G_J$.

**Step 5b:** Augmentation.

Define $x'$ to be the new $b$-matching obtained from $x$ by subtracting (adding) $\sigma$ from (to) $x_k$ if $k$ is an even (odd) edge of $\pi_1$ or $\pi_2$ not belonging to $\pi_s$, and subtracting (adding) $2\sigma$ from (to) $x_k$ if $k$ is an even (odd) edge of $\pi_s$. Then

$$\triangle(G_J; x', y) \leq \triangle(G_J; x, y) - 2.$$

**Step 5c:** Computation of new $F$.

If $def_{x'}(r) = 0$ then remove $T$ from $F$ and go to Step 1.

If $def_{x'}(r) > 0$, but there are $l \in E(\pi_s)$ such that $x'_l = 0$, let $k$ be the first such edge in $\pi_s$, and $T'$ be the portion of T above $k$. Remove $T'$ and $k$ from $F$ and go to Step 1.

If $def_{x'}(r) > 0$ and $x'_l > 0$ for all $l \in E(\pi_s)$, but $x'_k = 0$ for some edge $k$ of $C$, remove all such edges $k$ from $F$ and go to Step 1.

Finally, if $def_{x'}(r) > 0$ and $x'_l > 0$ for all $l \in E(\pi_1) \bigcup E(\pi_2) \bigcup \{j\}$, then there must be an even edge $k$ of $\pi_s$ for which $x'_k = 1$ or $def_{x'}(r) = 1$. Go to Step 6.

**Step 6:** Shrinking Step (see Figure 3.19).

We now identify a blossom in $G_J$. $T$ is the tree of $F^1$ containing $v_1$ and $v_2$, and $\pi_s$ is the path in $T$ from its root $r$ to the nearest vertex $q$ of $C$, the odd cycle formed by adding $j$ to $T$. Let $w$ be the first outer vertex of $\pi_s$ such that the path $\pi'$ in $T$ from $w$ to $q$ contains no even edge $k$ for which $x_k < 2$. Note $\pi'$ could be empty. Let $C' = \pi' \bigcup C$; we regard $C'$ as an odd cycle in the sense that we split each vertex of $\pi'$, except for $w$, into two vertices and split the edges of $\pi'$ properly. Then the blossom $B$ consists of $C'$ and any component $H$ of $G^+(x)$ such that $V(H) \bigcap V(C') \neq \emptyset$ except for the even edge of $T$ incident with $w$ if it exists. Note $\sum_{j \in V(B)} x_{wj} = b_w - 1$. Shrink blossom $B$ and go to Step 1.

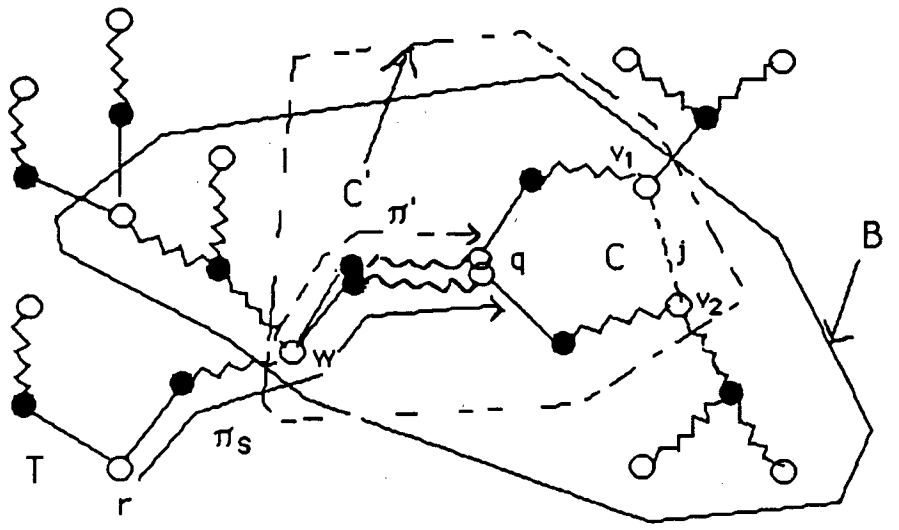


Figure 3.19: Shrinking Step

**Step 7** (see Figure 3.20): Grow Forest $F^1$ (pseudo forest growth).

Edge $j$ joins an outer vertex $v_1$ of a tree $T_1$ rooted at $r_1$ in $F^1$ to an inner vertex $v_2$ of a tree $T_0$ rooted at $r_0$ in $F^0$. Let $k$ be the first edge of the path from $v_2$ to $r_0$ in $T_0$, and $\overline{T}$ be the portion of $T_0$ above $k$. We adjoin $\overline{T}$ and the component H of $G^+(x)$ containing $v_2$ to $v_1$ by means of the edge $j$ thereby obtaining a larger tree $T_1'$ and a smaller tree $T_0'$ as indicated in Figure 3.20.

If $r_0 \notin V(T_1')$, then replace $T_1$ by $T_1'$ and $T_0$ by $T_0'$ in $F$. Go to Step 1.

If $r_0 \in V(T_1')$, then remove $T_0$ from $F^0$. Let $T$ denote $T_1'$ and perform the following steps.

**Step 7a:** Pseudo Augmentation.

Let $\pi$ be the path in $T$ from $r_1$ to $r_0$. Observe that both $r_0$ and $r_1$ are outer vertices of $T$. Let $\sigma_1 = \min\{x_j : j \text{ is an even edge of } \pi\}$. Let $\sigma = \min\{\sigma_1, def_x(r_1)\}$. Then $\sigma \geq 1$. Let $x'$ be the new $b$-matching obtained from $x$ by adding (subtracting) $\sigma$ to (from) $x_k$ if $k$ is an odd (even) edge of $\pi$. Then

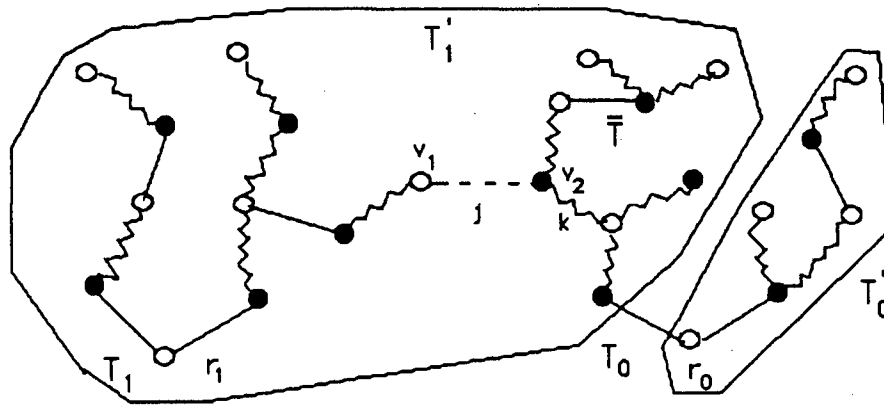


Figure 3.20: Pseudo Forest Growth

$$\Delta(G_J; x', y) \leq \Delta(G_J; x, y) - 1.$$

**Step 7b:** Computation of new $F$.

If $def'_x(r_1) = 0$, then remove $T$ from $F^1$, reroot $T$ at $r_0$, and add $T$ to $F^0$. Go to Step 1.

If $def'_x(r_1) > 0$, then we must have $x'_l = 0$ for some even edge of $\pi$; let $k$ be the first such edge, and $\overline{T}$ be the portion of $T$ above $k$. Remove $\overline{T}$ and $k$ from $T$, reroot $T$ at $r_0$ and add it to $F^0$. Go to Step 1.

Part Two: Optimality Test and Dual Variable Change.

**Step 8:** If $\Delta(G_J; x, y) = 0$, stop: the current $b$-matching is optimal.

**Step 9:** Dual Variable Change.

**Step 9a:** Calculation of $\theta$.

Let $\delta_1 = \min \{\alpha_i + \alpha_j - c_{ij}\}$ where $\{i, j\}$ joins an outer vertex of $F^1$ to a vertex not in $F^1$.

Let $\delta_2 = \min 1/2\{\alpha_i + \alpha_j - c_{ij}\}$ where $\{i, j\}$ joins two outer vertices of $F^1$.

Let $\delta_3 = \min \{r_k/2\}$ where $R_k$ is an inner pseudovertex of $F^1$.

Let $\delta_4 = \min \{\alpha_i\}$ where $i$ is an outer vertex of $F^1$.

Finally, let $\theta = \min\{\delta_1, \delta_2, \delta_3, \delta_4\}$.

**Step 9b:** Change of Dual Variable.

We define new dual solution $y' = (\alpha_i', r_k')$ as follows:

$$\alpha_i' = \begin{cases} \alpha_i - \theta & i \text{ is an outer vertex of } F^1. \\ \alpha_i + \theta & i \text{ is an inner vertex of } F^1. \\ \alpha_i & \text{otherwise.} \end{cases}$$

$$r_k' = \begin{cases} r_k + 2\theta & R_k \text{ is an outer pseudovertex of } F^1. \\ r_k - 2\theta & R_k \text{ is an inner pseudovertex of } F^1. \\ r_k & \text{otherwise.} \end{cases}$$

Then update edge set $J_e$ by this new dual solution.

**Step 9c:** If $\theta \in \{\delta_1, \delta_2\}$, we will get new edges in $G_J$. Go to Step 1.

If $\theta = \delta_3$, we must properly expand an inner pseudovertex of $F^1$ for which $r_k = 0$ and properly update the new alternating forest $F$. Then go to Step 1.

If $\theta = \delta_4$, let $I = \{i : \alpha_i' = 0\}$ where $i$ is an outer vertex of $F^1$. For each $i \in I$ such that $i$ is the root of a tree $T_i$ in $F^1$, remove $T_i$ from $F^1$ and add it to $F^0$. Then go to Step 1. Note that

$$\triangle(G_J; x, y') \leq \triangle(G_J; x, y) - 1.$$

If there is no $i \in I$ such that $i$ is the root of a tree in $F^1$, then choose any $r_0 \in I$, and let $r_1$ be the root of $T$. Go to Step 7a.

### 3.1.4 About Complexity

An upper bound on the amount of work required by the blossom algorithm to solve a weighted b- matching problem is of the order

$$\triangle(G; x^0, y^0).|V|.|E|$$

where $x^0$ and $y^0$ are the initial $b$-matching and dual solution. The details of the proof are available in Pulleyblank [8].

If we start with an initial $b$-matching and its corresponding dual solution as described in (3.24), the blossom algorithm will take $O(|b|.|V|.|E|)$ operations which is dependent on vector $b$. So this is not a strongly polynomial algorithm nor indeed a polynomial algorithm in the seting of a multigraph; a strongly polynomial algorithm for weighted $b$-matching problem was given by Anstee [2] (1983).

## 3.2 The Fixed Size Wighted $b$-matching

Let G = (V, E, c) be a weighted graph, $V^{\leq} \bigcup V^{=}$ be a partition of V, and b = $(b_i : i \in V)$ be a vector of positive integers. Then the fixed size weighted $b$-matching problem can be formulated in the following LP form: **(P)**:

$$\text{Max } c^T x$$

subject to:

$$\begin{aligned}
&\textstyle\sum_{i<j} x_{ij} = p && \\
&\textstyle\sum_{j=1}^{n} x_{ij} \leq b_i && \forall i \in V^{\leq} \\
&\textstyle\sum_{j=1}^{n} x_{ij} = b_i && \forall i \in V^{=} \\
&\textstyle\sum_{i,j \in R_k} x_{ij} \leq s_k && \forall R_k \subseteq V, \quad \textstyle\sum_{i \in R_k} b_i = 2s_k + 1 \\
&x_{ij} \geq 0 && \forall \{i,j\} \in E.
\end{aligned}$$

Note in the previous section we have taken $V^{=} = \emptyset$ for brevity's sake.

### 3.2.1 A Possible Generalization

**Conjecture:** The fixed size weighted $b$-matching problem can be solved by the primal dual algorithm in Section 2.2 also.

This conjecture is very likely to be true since there is no essential difference between these two problems. One can exactly follow the same procedure as in Section 2.2: Apply Part One of the blossom algorithm on $G_J$ instead of using the cardinality matching algorithm for the general matching problem, and pay some care to the computation of the dual variable change parameter $\theta$.

### 3.2.2 A Transformation Approach

We explore the same idea used in the previous chapter, namely, we want to transform the fixed size weighted $b$-matching problem into a weighted $b$-matching problem so that we can solve it by the blossom algorithm in Section 3.1.

We create a new graph $\overline{G} = (\overline{V}, \overline{E}, \overline{c})$ such that

$$\overline{V} = V \bigcup \{v_0\}$$
$$\overline{E} = E \bigcup \{\{v_0, v_j\} : v_j \in V^{\leq}\}$$
$$b_0 = \sum_{i \in V} b_i - 2p$$
$$\overline{b} = (\overline{b}_i : i \in V)$$
$$\overline{c} = (\overline{c}_{ij} : \{\mathrm{i, j}\} \in \overline{E}), \text{where}$$

$$\overline{b}_i = \begin{cases} b_i & \forall i \in V \\ b_0 & i = v_0 \end{cases}, \text{ and } \quad \overline{c}_{ij} = \begin{cases} c_{ij} & \{\mathrm{i, j}\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

We create a new problem $(\overline{P})$ on $\overline{G}$ as follow:

$(\overline{P})$:

$$\text{Max } c^T x$$

subject to:

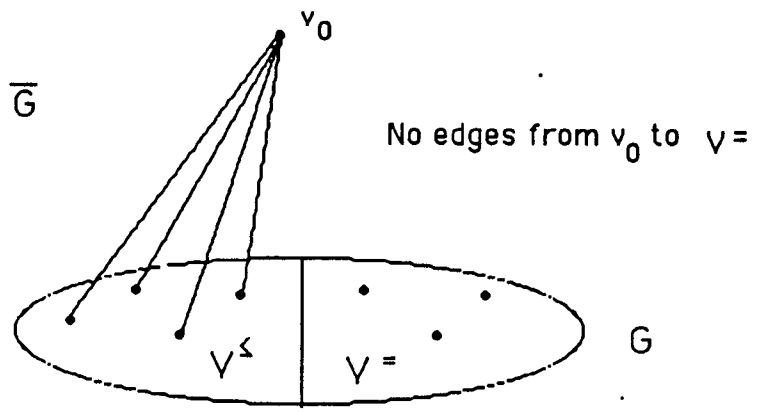


Figure 3.21: The construction of $\overline{G}$

$$\begin{aligned} &\textstyle\sum_{j=1}^{n} x_{ij} = \overline{b}_i && \forall i \in \overline{V} \\ &\textstyle\sum_{i,j \in R_k} x_{ij} \leq s_k && \forall R_k \subseteq \overline{V} \quad \text{s.t. } \textstyle\sum_{i \in R_k} \overline{b}_i = 2s_k + 1 \\ &x_{ij} \geq 0 && \{\text{i, j}\} \in \overline{E}. \end{aligned}$$

Note we have partitioned $\overline{V}$ into $\overline{V}^{=} \bigcup \overline{V}^{\leq}$, where $\overline{V}^{\leq} = \emptyset$ (see Figure 3.21). In fact $(\overline{P})$ is also a weighted $f$-factor problem.

**Theorem 3.2:** (P) has a feasible $b$-matching iff $(\overline{P})$ has one; an optimal $b$-matching on $(\overline{P})$ induces an optimal $b$-matching of (P); conversely, an optimum solution of (P) can be extended to an optimum solution of $(\overline{P})$.

**Proof:**

1. Assume x is a feasible matching of (P), and define $\overline{x}$ as follow:

$$\overline{x} = (\overline{x}_{ij} : \{\text{i, j}\} \in \overline{E}) \text{ where}$$

$$\overline{x}_{ij} = \begin{cases} x_{ij} & \{\text{i, j}\} \in E \\ b_i - \sum_{r \in V} x_{ir} & j = v_0, i \in V^{\leq}. \end{cases}$$

Then $\sum_{i<j} \overline{x}_{ij} \equiv \overline{b}_i, \forall i \in \overline{V}$, so $\overline{x}$ is a feasible $b$-matching of $(\overline{P})$; Conversely, if $\overline{x}$ is a feasible $b$-matching in $(\overline{P})$, then $(\overline{x}|_G)$ is obviously a feasible $b$-matching of (P).

2. Let x be a maximum weighted matching of (P), and define $\overline{x}$ as above. Then $\overline{x}$ is a maximum weighted matching of $(\overline{P})$. If not, assume $\overline{x}'$ is a maximum weighted matching of $(\overline{P})$ such that

$$\overline{c}^T\overline{x}' > \overline{c}^T\overline{x}$$

but $\overline{c}^T\overline{x}' = c^T(\overline{x}'|_G) > c^T x$, so x is not a maximum weighted matching, a contradiction!

Conversely if $\overline{x}$ is a maximum weighted matching of $(\overline{P})$, then $(\overline{x}|_G)$ is a maximum weighted matching of (P). This is also because of

$$\overline{c}^T\overline{x} = c^T(\overline{x}|_G).$$

**Q.E.D.**

**Complexity:** Our transformation doesn't increase complexity significantly since we have only added one vertex and at most $n$ edges.

**Remark:** In (P), if we set b $\equiv 1$ (i.e. $b_i = 1 : \forall i \in V$), we come to a general fixed size weighted matching problem; when we create the new problem $(\overline{P})$, we have $b_0 = n - 2p$. This transformation is better than the one given the previous chapter if we want to solve this special case weighted $b$-matching problem $(\overline{P})$ by Pulleyblank's algorithm, since in the previous one, we added $n - 2p$ vertices and $n(n - 2p)$ edges.

# Chapter 4

# Fixed Size $(g, f)$-Factor Problem

## 4.1 Preliminaries

Let $G = (V(G), E(G))$ be a multigraph where for each $e \in E(G)$, we let $u_e$ denote the multiplicity of an edge $e$ in $G$. We may understand $G$ as a simple graph with a capacity vector $u = (u_e : e \in E(G))$.

Let $g = (g_v : v \in V(G))$, $f = (f_v : v \in V(G))$ be vectors of nonnegative integers satisfying

$$\forall v \in V(G): \qquad 0 \leq g_v \leq f_v \leq deg_G(V),$$

where $deg_G(v)$ measures the degree of $v$ in $G$ counting multiplicities. A $(g, f)$-factor is a subgraph D of G with

$$\forall v \in V(G): \qquad g_v \leq deg_D(v) \leq f_v.$$

This might be called a capacitated $(g, f)$-factor since an edge can be used at most $u_e$ times.

A $b$-matching is a $(g, f)$-factor with $g \equiv 0$, and an $f$-factor is a $(g, f)$-factor with $f \equiv g$.

It is convenient to state the problem in matrix terms. Let $G = (V(G), E(G))$ have the adjacency matrix $u = (u_{ij})$. For the brevity's sake, we do not allow loops in our graph. We wish the row and column sums of an adjacency matrix $A$ to correspond to the degrees of the associated vertices. Then a $(g, f)$-factor corresponds to a symmetric integral matrix $A$ with zero entries on the diagonal, $i$th row and column sum is bounded

by $(g_i, f_i)$ for $i = 1, 2, \ldots, n$ and satisfying $0 \leq A \leq u$. When $g \equiv f$, a $(g, f)$-factor reduces to an $f$-factor for which we require the $i$th row and column sum to be $f_i$ for $i = 1, 2, \ldots, n$.

## 4.2 An $f$-Factor Algorithm

A number of algorithms for solving $f$-factor problems are known. One restriction of the problem is to take the G to be a simple graph with capacity vector $u \equiv 1$, in which case the $f_i$'s are bounded by $n - 1$ which is a polynomial in $n$. However in the general case the $f_i$'s are not bounded by a polynomial in $n$ and so this technique will not yield a polynomial algorithm.

The algorithm introduced in this section is due to Anstee [1] which solves the problem by either finding an $f$-factor or showing one does not exist and does this in $O(n^3)$ operations. This complexity bound is independent of the size of $u$ or the size of $f_i$'s, and so is strongly polynomial.

### 4.2.1 Network Flow Formulation

Our search for an $f$-factor is split into two parts.

First, network flows is employed to find an integral matrix $A$ with $i$th row and column sums $f_i$ for $i = 1, 2, \ldots, n$, and satisfying $0 \leq A \leq u$. We call such a matrix a **directed $f$-factor**. If no directed $f$-factor exists, then clearly no $f$-factor exists.

Second, we are left only with the problem of making $A$ symmetric since our network flows will, because of the lack of loops, automatically give zero entries on the diagonal. We compute an half integral symmetric matrix

$$x = (A + A^T)/2$$

where $A^T$ is the transpose of $A$. Then we eliminate the halves by an alternating walk

algorithm (see Section 4.2.3). If the alternating walk algorithm can not delete all these halves, then we will prove that no $f$-factor exists by displaying an $f$-barrier (see Section 4.2.4).

A directed $f$-factor correspond to an integral flow in the following network. There is a source $s$, a sink $t$, and nodes $R_1, R_2, \ldots, R_n, S_1, S_2, \ldots, S_n$. There are directed edges from $s$ to $R_i$ and $S_i$ to $t$ both with capacity $f_i$ for $i = 1, 2, \ldots, n$. There are directed edges from $R_i$ to $S_j$ with capacity $u_{ij}$ for $i, j = 1, 2, \ldots, n$. Assuming there is an integral maximal flow of size $f_1 + f_2 + \ldots + f_n$, form a matrix $A = (a_{ij})$ from the flow by letting $a_{ij}$ be the flow from $R_i$ to $S_j$. We deduce that A has $i$th row and column sum $f_i$ and satisfies $0 \leq A \leq u$ and thus is a directed $f$-factor.

Note that our network has $|V| = 2n+2$ vertices, so a directed $f$-factor can be obtained in $O(|V|^3) = O(n^3)$ operations or $O(|V||E|log|V|)$ if you wish to take advantage of the sparsity of $G$ and hence of our network.

### 4.2.2 Symmetrizing a Directed $f$-Factor

Let $A$ be a directed f-factor, and form a matrix

$$x = (A + A^T)/2.$$

Now $x$ is symmetric, has the the desired row and column sums and satisfies $0 \leq x \leq u$. Unfortunately, it need not be integral, merely half integral off the diagonal and zero entries on the diagonal. We find that the remaining problem of removing the halves is bounded in that it does not involve the $f_i$'s and only some of the edges of $G$. One may view the directed $f$-factor as the bulk of the solution.

These are two phases in making $A$ into an $f$-factor. Define a graph $H$ on the vertices of $G$ with edge $\{i, j\}$ in $H$ whenever $x_{ij}$ is not integral. Thus we can add or subtract $1/2$ from these entries and keep $0 \leq x \leq u$. Since the row sums of $x$ are integral, we deduce

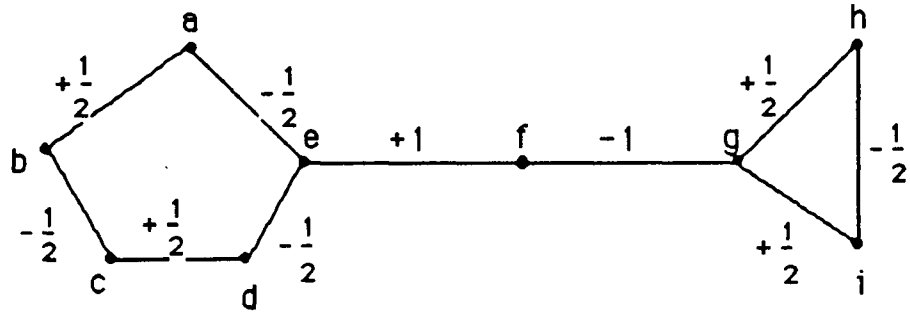


Figure 4.22: Two cycles in $H$ joined by an alternating walk in $G$

that the degrees of $H$ are even, and so $H$ can be decomposed into closed trails. If $H$ has an even length closed trail, then alternately adding and subtracting $1/2$ from the entries of $x$ corresponding to the edges of the closed trail leaves the row and column sums of $x$ unchanged with $0 \leq x \leq u$. Thus we wish to remove all the even length closed trails from $H$, which can be done by the following simple algorithm. We call it Eliminating Algorithm One.

**Eliminating Algorithm One:** Start with any path and extend it using the fact that the degrees are even. Eventually some vertex appears twice, yielding a cycle. If the cycle is even, then eliminate it as described; if the the cycle is odd, remove it from the graph but save it. Continue this process, starting over with as much of the original path as was left. If at any point two odd cycles have been found with any vertices in common, then the edges of the cycles can be decomposed into one or more even length closed trails which can be eliminated from $H$. Thus the algorithm terminates with $H$ consisting of only vertex disjoint odd cycles.

The algorithm takes $O(|E|)$ operations, since each edge of $H$ can be investigated at most twice.

With $H$ consisting of vertex disjoint odd cycles, we start our second phase which is to eliminate the odd cycles from $H$ in pairs as in Figure 4.22. The cycle *abcde* and *ghi* are in

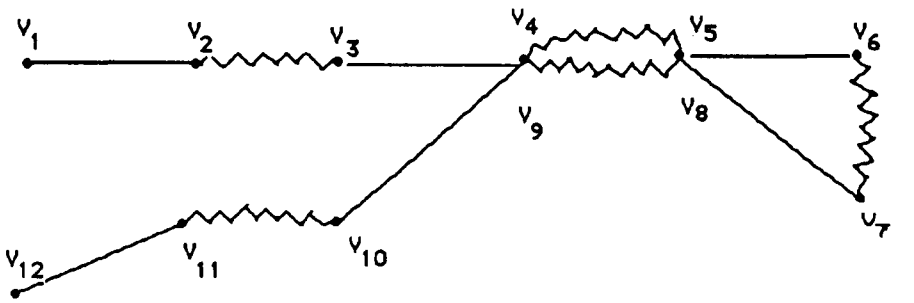


Figure 4.23: A key example of alternating walk

$H$, but edges $\{e, f\}, \{f, g\}$ are not (they are edges of $G$). The numbers next to the edges are added to the corresponding entries in $x$. Row and column sums are preserved, and the operations on the edge $\{e, f\}$ and $\{f, g\}$ must be chosen so that after the changes, we still have $0 \leq x \leq u$. In performing these changes, the two odd cycles of $H$ are removed as described. Note we are always left with even number of odd cycles.

Thus we search for walks of edges not in $H$ which always allow us to alternately add and subtract 1 and join two odd cycles of $H$.

Let us define these walks precisely. Consider a walk $v_1 e_1 v_2 e_2 v_3 \ldots e_{n-1} v_n$, where no edges of $H$ are allowed since then we could terminate sooner. The walk will be called an **alternating walk** starting at $v_1$ and ending at $v_n$ if we may perform the following changes to x and still keep $0 \leq x \leq u$: add 1 to entry $e_1$, subtract 1 from entry $e_2$, add 1 to entry $e_3, \ldots$, or the same series starting with subtraction. We say that the walk ends in addition (resp. subtraction) if we are adding to (resp. subtracting from) entry $e_{n-1}$. We require our alternating walk to be minimal (not necessarily minimum) subject to these conditions. Thus an edge may be used twice but only if both times we are adding to it or subtracting from it and the second time it appears with its ends in reverse order. See Figure 4.23. Our definition ensures that we can eliminate two odd cycles if they are joined by an alternating walk.

### 4.2.3 Algorithm for Finding an Alternating Walk

Let $H$ consist of vertex disjoint odd cycles $C_1, C_2, \ldots, C_t$. We start at any odd cycle of $H$, say $C_1$, and try to find an alternating walk from any vertex of $C_1$ to any vertex of any $C_k$ for $k > 1$. This takes $O(n^2)$ operations and there are at most $n/3$ cycles in $H$, so we need only find $n/6$ such walks. Thus assuming we are successful in finding the alternating walks, the algorithm will take $O(n^3)$ operations to remove all the odd cycles from $H$ and arrive at the desired f-factor.

In the algorithm we form a multigraph $G^*$ on the vertices of $G$ with edges in two classes $M$, $\overline{M}$. Because an alternating walk may use an edge at most twice, we will allow up to two edges in each class to join the same pair of vertices. The edges in $M$ correspond to entries for which we may subtract 1 (two edges if we may subtract 2) and edges in $\overline{M}$ correspond to entries for which we may add 1 (two edges if we add 2) and yet all the while keeping $0 \leq x \leq u$.

We form a directed tree directed out from a base vertex which corresponds to $C_1$. The tree grows vertex by vertex (Step 5). Using the notation $p(v) = w$ to denote that $w$ is the precursor of $v$ in the tree, the directed paths in the tree correspond to alternating walks. The edges of the tree are stored separately from $G$ to keep them from being used again. A vertex is given a label $S$ (resp. $T$) if there is an alternating walk from a vertex of $C_1$ to the given vertex ending in addition (resp. subtraction).

When an edge is added to the tree that creates a cycle it is called a blossom. We shrink the vertices of the blossoms to a single pseudovertex, and there may be nesting of blossoms. An outermost blossom or pesudovertex in this inductive structure is called **exposed.**

For each blossom, one vertex (possible a pseudovertex) is distinguished as the base vertex, being the vertex of the blossom closest to the base (root) of the tree. If the base

of a blossom is not a pseudovertex then it is called a true base, otherwise the true base is inductively defined as the true base of the base of the blossom.

In the shrinking operation, edges joining vertices on the blossom may be deleted as redundant. The edges of the blossom are saved. Further multiple edges may arise in the shrinking process, but for a given pair of vertices only 2 from $M$ and 2 from $\overline{M}$ could be used, the others could be deleted if desired. We keep track of the original ends of an edge as well as its new ends, which is useful in finding the alternating walk in $G$ from a directed path in the tree. The tree also shrinks but we end up with a tree. We give the pseudovertices labels $S$,$T$, and these labels apply to all vertices of $G$ contained in them.

Throughout the algorithm, we use the term **dual** to refer to the replacement of $x_e$ by $u_e - x_e$, label $S$ by label $T$, addition by subtraction, $M$ by $\overline{M}$, and vice versa.

**Algorithm for Finding an Alternating Walk**

- **Step 1:** Shrink the vertices of $C_1$ in G into a single vertex which forms the base of the tree and gets labels $S$,$T$. The vertex is put on the list of **unscanned vertices** $Y$. Form a multigraph $G^*$ whose edges are divided into two classes $M$, $\overline{M}$. The edges of $M$ consist of $\text{Min}(x_e, 2)$ copies of edge $e$. We ignore the edges of the odd cycles of $H$. The edges of $\overline{M}$ are given by the dual definitions.

- **Step 2:** If $Y$ is empty, then stop, there is an $f$-barrier (refer to next section), else, pick a $v \in Y$ where $u$ is automatically a vertex of the tree.

- **Step 3:** If $v$ has label $S$, do Steps 4 and 5.

- **Step 4:** (searching for blossoms) Search for any edge $\{v, w\} \in M$, where $w$ is in the tree and has label $S$. If there is such an edge, it forms a blossom in the tree which we shrink to a pseudovertex. The pseudovertex is given labels $S$,$T$ and is added to

$Y$. The vertices contained in the pseudovertex are deleted from $Y$. Return to Step 2. If there is no such edge, continue.

- **Step 5:** (grow tree). For all vertices $w$ not in the tree, do the following. If there is an edge $\{v, w\} \in M$, then put $w$ in the tree with $p(w) = v$, and label $T$. The chosen edge $\{v, w\}$ is deleted and $w$ is added to $Y$. If in addition $w \in C_k$ for some $k > 1$, then stop. There is an alternating walk from a vertex of $C_1$ to a vertex of $C_k$.

- **Step 6:** If $v$ has the label $T$ do the duals of steps 4 and 5.

- **Step 7:** Consider $v$ to be scanned and delete $v$ from $Y$. Return to Step 2.

The algorithm will certainly terminate. The assertions made in Steps 2 and 5 are verified by the following lemmas (for the proofs refer to Anstee [1]).

**Lemma 4.1:** Consider a pseudovertex $u$ with true base $b$. Then there is an alternating walk in the original $G^*$ from $b$ to any vertex of $G$ (not in $C_1$) of the pseudovertex ending in addition and another ending in subtraction. Both these walks start with addition (resp. subtraction) if the label of $p(b)$ is $S$ (resp. $T$). If there is no $p(u)$, then we do not care how the walks start since $b$ will corrspond to $C_1$.

**Lemma 4.2:** If a vertex $v$ of $G$ has a label $S$, then there is an alternating walk from a vertex of $C_1$ to $v$ ending in addition. The dual also holds.

**Lemma 4.3:** If a vertex $v$ is reachable from a vertex of $C_1$ by an alternating walk ending in addition then $v$ will have the label $S$ if the algorithm terminates at Step 2. The dual also holds.

### 4.2.4 $f$-Barrier

We will only characterize the structure of our $f$-barrier; for the rigorous proof one should see Anstee [1].

Suppose at any stage we can not find an alternating walk, then we can find an $f$-barrier as follows. Define a vertex $y$ as **reachable** from a vertex $x$ if there is an alternating walk from $x$ to $y$. A vertex is considered to reachable from itself by an alternating walk of no edges which can be considered to end in either in addition or subtraction.

Assume we were unable to elnminate $C_1$. Define a partition of the vertices of $G$ into sets $S$, $T$, $U$ as follow. Let $S$ be the vertices reachable from vertices in $C_1$ only by alternating walks ending in addition, and let $T$ be the vertices reachable from vertices in $C_1$ only by alternating walks ending in subtraction. The remaining vertices $U$ decompose into two sets $W$, $L$ where $W$ is the set of vertices each one of which is reachable from a vertex of $C_1$ by an alternating walk ending in addition and from a vertex of $C_1$ by an alternating walk ending in subtraction. The set $L$ consists of the vertices not reachable from a vertex of $C_1$ and so includes the vertices of $C_2, C_3, \ldots, C_t$.

We define an edge $\{i, j\}$ to be empty (resp. full) if $x_{ij} = 0$ (resp. $u_{ij} - x_{ij} = 0$). Then each edge joining a vertex of $S$ to a vertex of $S \bigcup L$ is empty. Each edge joining a vertex of $T$ to a vertex of $T \bigcup L$ is full. There are no edges in $G$ joining vertices of $L$ to vertices of $W$. In the subgraph of $G$ induced by the vertices of $W$, there are components $U_1, U_2, \ldots, U_l$. Let $C_1 \subseteq U_1$, then every edge joining $U_1$ to $S$ (resp. $T$) will be empty (resp. full). The same will be true for each $U_k$ $(k > 1)$ with one exceptional edge for each $U_k$ as follows. Either there is an edge $\{i, j\}$ with $i \in U_k$, $j \in S$, and $x_{ij} = 1$ or there is an edge $\{i, j\}$ with $i \in U_k, j \in T$, and $x_{ij} = 14$. All these properties plus some network results and computations constitute the proof that no $f$-factor exists. One can consult Anstee [1] for the details. Therefore, our partition of the vertices of $G$: $S$, $T$, $L$,

$U_1, U_2, \ldots, U_l$ is called an **$f$-barrier.**

## 4.3 Fixed Size $(g, f)$-Factor

Necessary and sufficient condition on the existence of a $(g, f)$-factor are given by Lovász [7]. An $O(n^3)$ algorithm for finding a $(g, f)$-factor on a multigraph is due to Anstee [3]. This section is mainly my work; I am going to explore the same idea used in the $f$-factor problem to find a fixed size $(g, f)$-factor or displaying a fixed size $(g, f)$-barrier and do this in $O(n^3)$ operations.

### 4.3.1 Network Flow Formulation

Form a network with a source $s_0$, a sink $t$, and nodes $s, R_1, R_2, \ldots, R_n, S_1, S_2, \ldots, S_n$. There is a directed edge from $s_0$ to $s$ with upper bound = lower bound = $2p$. There are directed edges from $s$ to $R_i$, and $S_i$ to $t$ both with upper bound $f_i$ and lower bound $g_i$ for $i = 1, 2, 3, \ldots, n$. There are directed edges from $R_i$ to $S_j$ with capacity $u_{ij}$ for $i, j = 1, 2, \ldots, n$. (see Figure 4.24). Assume there is a feasible flow of size $2p$. Form a matrix $A = (a_{ij})$ from the flow by letting $a_{ij}$ be the flow from $R_i$ to $S_j$. We deduce that $A$ has $i$th row and column sum between $g_i$ and $f_i$ and satisfying $0 \leq A \leq u$. We call $A$ a directed size $2p$ $(g, f)$-factor. If no directed size $2p$ $(g, f)$-factor exists, then clearly no size $p$ $(g, f)$-factor exists.

### 4.3.2 Symmetrizing a Directed Size $2p$ $(g, f)$-Factor: Phase One

Symmetrization is much more involved than that in the $f$-factor problem, since we will have to preserve the fixed size $2p$ on the network (i.e., size $p$ on graph $G$).

Let $A$ be a directed size $2p$ $(g, f)$-factor. Form a matrix

$$x = (A + A^T)/2$$

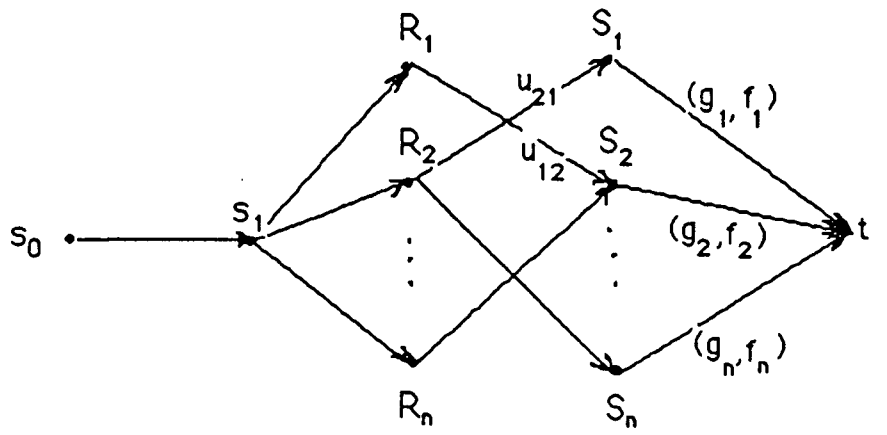


Figure 4.24: Network For Size $p$ $(g, f)$-Factor Problem

so that $x$ is symmetric and half integral. We can view $x$ as a fractional subgraph of $G$, with $0 \leq x \leq u$, and $g_i \leq deg_x(i) \leq f_i$ for $i = 1, 2, \ldots, n$. We still define $H$ as the subgraph of $G$ whose edges are half integral. This time the row sums of $x$ need not be integral, so some vertices of $H$ may have odd degrees. We need some definitions before going further.

Less Than Upper Bound (**LTUB**) vertex: A vertex $v \in V$ is called an LTUB if $g_v = deg_x(v) < f_v$.

Greater Than Lower Bound (**GTLB**) vertex: A vertex $v \in V$ is called a GTLB if $g_v < deg_x(v) = f_v$.

Strictly in Between (**SB**) vertex: A vertex $v \in V$ is called a SB if $g_v < deg_x(v) < f_v$.

Note $V$ can only have even number of odd degree vertices, and these vertices are all SB's. We can decompose $H$ into

1. Even length closed trails.

2. Vertex disjoint odd cycles.

3. Even length paths between two odd degree vertices.

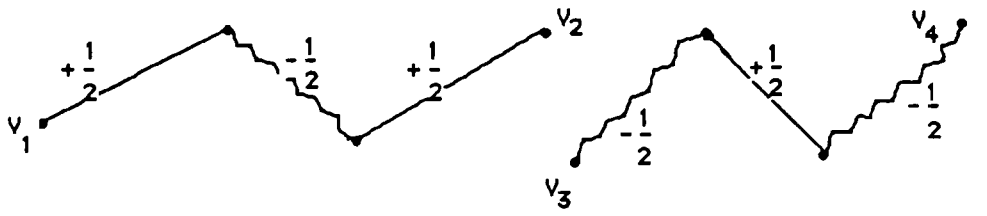


Figure 4.25: A pair of odd length paths

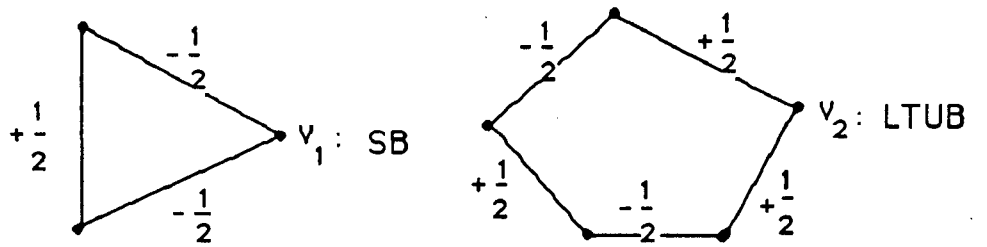


Figure 4.26: A pair of odd degree cycles

4. Odd length paths between two odd degree vertices.

For 1. and 3. we can eliminate them by alternately adding 1/2 and subtracting 1/2 on these edges and we still have a directed size $2p$ $(g, f)$-factor. For 4. we can eliminate any pair of such odd length paths by the similiar change (see Figure 4.25).

Note $v_1, v_2, v_3, v_4$ all have odd degrees in $H$, so they are all SB's so that we can on one path start with addition and on another start with subtraction. Let us call this kind of change a **double change**. After enough double changes we are left with only one such odd length path or none. For 2. we can also eliminate a pair of odd cycles by a similiar double change if there is an LTUB on one cycle and a GTLB on another. Here a SB can be used either as an LTUB or a GTLB. (see Figure 4.26)

We need an algorithm to accomplish the above eliminations.

**Eliminating Algorithm Two:**

- Step 1: If $H$ has odd degree vertices go to Step 2, else, go to Step 4.
- Step 2: Start with any path starting with an odd degree vertex and extend it if possible. When some vertex appears twice, a cycle is found. If the cycle is even, then eliminate it as described; if it is odd remove it from the graph but save it. Continue this process, starting over with as much of original path as was left. If at any point, two odd cycles have been found with any vertices in common, then the edges of the cycles can be decomposed into one or more even length closed trails which can be eliminated from $H$. When we are stuck we must end with an odd degree vertex. If this is same as the initial vertex we get a cycle which can be handled as above, otherwise we are left with a path between two odd degree vertices, then go to Step 3.
- Step 3: If the path between two odd degree vertices has even length, eliminate it as described. If odd length, remove it, but save it. Whenever we have two such paths, we eliminate them as a pair by the double change, go to Step 1.
- Step 4: All vertices have even degree now. Run the Eliminating Algorithm One given in the Section 4.2.2.

The above algorithm also takes $O(|E|)$ operations which is the same as the Eliminating Algorithm One. When the above algorithm stops we are left with either an even number of vertex disjoint odd cycles or one odd length path plus an odd number of vertex disjoint odd cycles.

### 4.3.3 Symmetrizing a Directed Size $2p$ $(g, f)$-Factor: Phase Two

An odd length path can be regarded as an odd cycle by pasting both end vertices into one, we call the resulting cycle a **pesudocycle**. Later on, one will see that an odd length

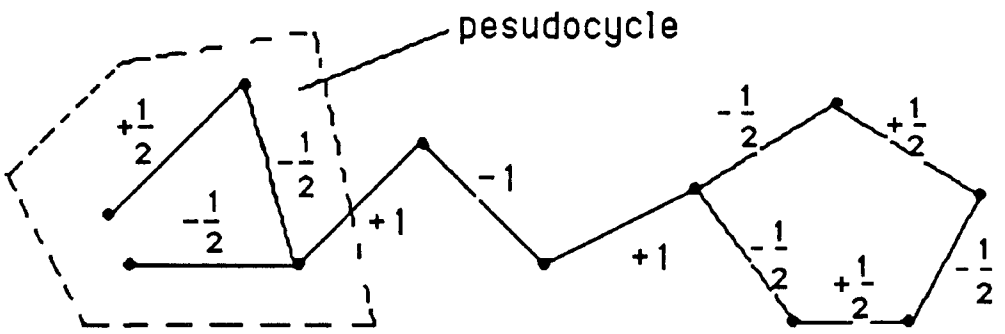


Figure 4.27: Type 1 example

path plays the same role as an odd cycle: When it appears in Type 1, it can be eliminated with another odd cycle as in Figure 4.27. In Type 2, it can be used as a cycle joined to either a LTUB or a GTLB by an alternating walk of zero length ending in either addition or subtraction; this is because the both end vertices of an odd length path are SB's, and thus the pasted vertex in a pesudocycle is a SB. It is obvious that a pesudocycle can never occur in Type 3. Thus we can simply assume when Eliminating Algorithm Two stops we are left with only vertex disjoint odd cycles $C_1, C_2, \ldots C_{2k}$. In order to keep the same size we can only eliminate these cycles in pairs if possible.

Before going further let's check the vertices of these cycles. Whenever we find a LTUB on one cycle, and a GTLB on another, we can eliminate them as in Figure 4.26 by a double change. Here a SB can still be used as either an LTUB or a GTLB. To eliminate more cycles we have to employ alternating walks again, we first examine how we can eliminate these cycles.

- Type 1: Two cycles are joined by an alternating walk, we can perform the following changes as illustrated in Figure 4.27.

- Type 2: One cycle is joined to an LTUB by an alternating walk ending in addition

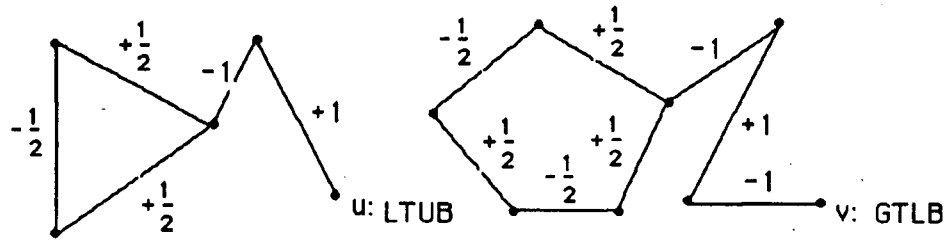


Figure 4.28: Type 2 example

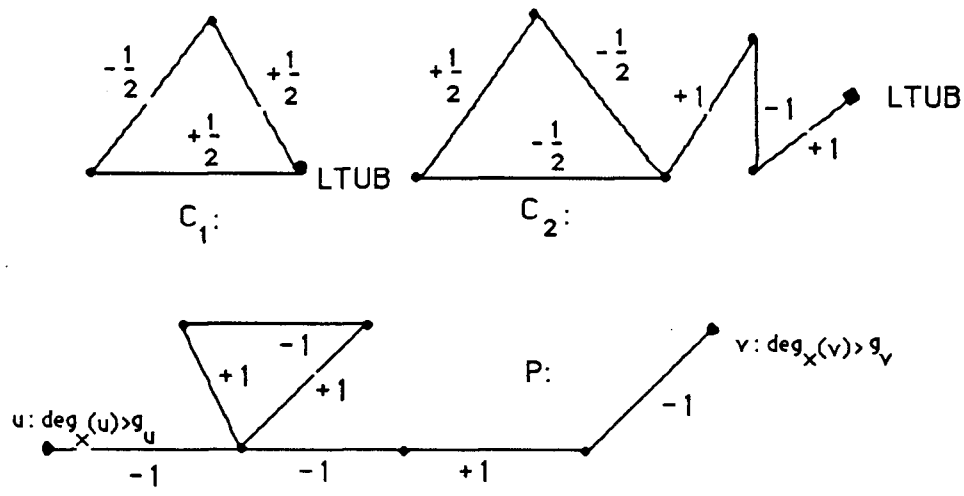


Figure 4.29: Type 3 example

and another cycle is joined to a GTLB by an alternating walk ending in subtraction, then these two cycles can be eliminated (see Figure 4.28). Note a SB could be used as either an LTUB or a GTLB, $u$ could be on $C_1$ and $v$ also could be on $C_2$.

- Type 3: Each of the cycles is joined to an LTUB (resp. GTLB) by an alternating walk ending in addition (resp. subtraction) and an odd length alternating walk between two GTLB's (resp. LTUB's) or SB's or a GTLB (resp. LTUB) and a SB (see Figure 4.29).

Note: Changes on $C_1$ and $C_2$ totally increase size by 1 and changes on the walk $P$ decrease

size by 1 (We are actually considering the edge size on G; the size should doubled if on the network). Now we give the the following algorithm to deal with these cycles of Types 1, 2, and 3.

**Eliminating Algorithm Three (E.A.T.):**

- Step 1: Apply the alternating algorithm on each cycle. If at any stage, a cycle is joined by an alternating walk to another cycle, then eliminate them as in Type 1.
- Step 2: For each $i$, grow tree $T_i$ with $C_i$ as base. Whenever we encounter Type 2 with an LTUB (or a SB) on one tree and a GTLB (or a SB) on the other, then the two cycles can be eliminated as in Type 2.
- Step 3: If two $T_i$'s each have at least one LTUB (resp. GTLB) or SB vertex reachable from base ending in addition (resp. subtraction), then seek an alternating walk joining two GTLB (resp. LTUB) or SB vertices starting and ending in subtraction (resp. addition). If we succeed, we can eliminate $C_1$ and $C_2$ as in Type 3.

**Remark:** We apply the alternating walk algorithm at most $n$ times, so the total complexity of steps 1, 2, and 3 is still $O(n^3)$.

We will prove in next section if we still have cycles left after using elimination of Types 1, 2, and 3, then there does not exist a size $p$ $(g, f)$-factor.

### 4.3.4 Fixed Size $(g, f)$-Barrier

First we present an alternate method to solve our fixed size $(g, f)$-factor problem by transforming it into an $f$-factor problem. Create a new graph $\overline{G} = (\overline{V}, \overline{E})$, such that

$$\overline{V} = V \bigcup \{s\}, \qquad \overline{E} = E \bigcup \{\{s, i\} : i \in V\},$$

with the multiplicity $\overline{u}$ defined as:

$$\forall i \in V: \qquad \overline{u}_{si} = f_i - g_i; \quad \forall e \in E: \qquad \overline{u}_e = u_e,$$

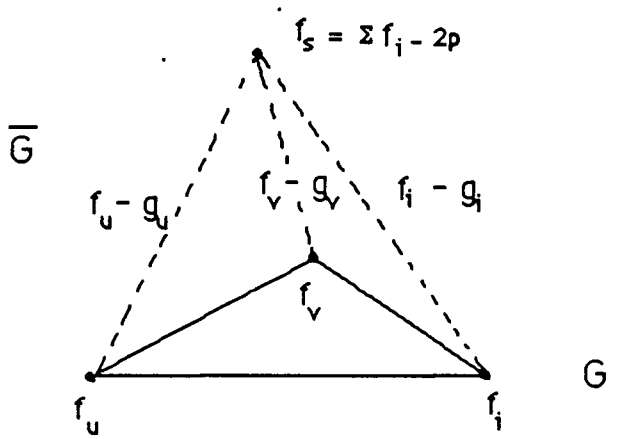


Figure 4.30: An transformation from G to $\overline{G}$.

and $\overline{f}$ is defined as:

$$\overline{f}_i = f_i, \qquad \forall i \in V; \qquad \text{and } \overline{f}_s = \sum_{i \in V} f_i - 2p.$$

**Remark:** Assume each $g_i$ of $g = (g_i : i \in V)$ can only be 0 or $f_i$. Define $V^{\leq} = \{i \in V : g_i = 0\}$ and $V^{=} = \{i \in V : g_i = f_i\}$, then this transformation is exactly same as the one in Figure 2.21. Thus this one is in fact a generalization of the one given for fixed size weighted $b$-matching problem.

**Theorem 4.4:** There is a size $p$ $(g, f)$-factor on G iff there is an $\overline{f}$-factor on $\overline{G}$; an $\overline{f}$-factor on $\overline{G}$ induces a size $p$ $(g, f)$-factor on G (this transformation also solves the weighted case fixed size $(g, f)$-factor problem, see Figure 4.30).

**Proof:** Assume $x$ is a size $p$ $(g, f)$-factor of $G$. Define $\overline{x}$ as follows:

$$\overline{x} = (\overline{x}_{ij} : \{\mathrm{i}, \mathrm{j}\} \in \overline{E}) \text{ where}$$

$$\overline{x}_{ij} = \begin{cases} x_{ij} & \{\mathrm{i}, \mathrm{j}\} \in E \\ f_i - \sum_{r \in V} x_{ir} & j = s, i \in V. \end{cases}$$

Then $\overline{x}_{si} \leq f_i - g_i \ \forall i \in V$ since $\sum_{r \in V} x_{ir} \geq g_i$; $\sum_{j \in V} x_{sj} = f_s = \sum f_i - 2p$. The other

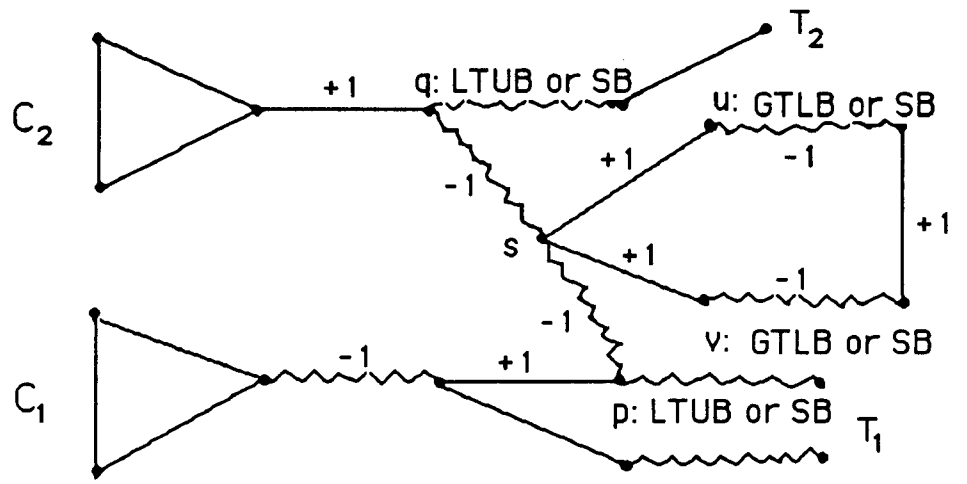


Figure 4.31: Figure for Theorem 4.6.

vertices of $V$ are clearly all saturated by $\overline{x}$, thus $\overline{x}$ is an $\overline{f}$-factor on $\overline{G}$. Conversely, if $\overline{x}$ is an $\overline{f}$-factor on $\overline{G}$, then it easy to see that $\overline{x}|_G$ is a size $p$ $(g, f)$-factor.

Secondly, by using this transformation we will deduce a fixed size $(g, f)$-barrier on G via an $\overline{f}$-barrier on $\overline{G}$.

**Lemma 4.5:** If in $\overline{G}$ there is no alternating walk joining any pair of odd cycles, then no $\overline{f}$-factor exists in $\overline{G}$ which implies no size $p$ $(g, f)$-factor in G. An $\overline{f}$-barrier in $\overline{G}$ induces a fixed size $(g, f)$-barrier in G.

**Theorem 4.6:** If Eliminating Algorithm Three fails to eliminate all the odd cycles, then there does not exist a fixed size $(g, f)$-factor in G.

**Proof** (see Figure 4.31): Consider an alternating walk $w$, which must pass through $s$ in view of Step 1 of E.A.T., from $C_1$ to $C_2$ in $\overline{G}$. Without loss of generality, assume $w$ first reaches $s$ from $T_1$ with subtraction from a vertex $p$. We deduce by definition of $\overline{G}$ and the chosen subgraph that $p \in$ LTUB or SB. Now the reversal of $w$ goes from $C_2$ to $C_1$, and so first reaches $s$ from a vertex $q$ by subtraction with $q \in$ LTUB or SB (if by addition then $q \in$ GTLB or SB, violating Step 2 of E.A.T.). But since $w$ is an alternating

walk, there is an alternating walk from $s$ to $s$ starting with addition to a vertex $u$ (which then is a GTLB or SB) and also ending in addition from a vertex $v$ (which must also be a GTLB or SB), and so there is an alternating walk joining $u$ and $v$ starting and ending in subtraction; but now Step 3 of E.A.T. would have been used. So we have a contradiction. **Q.E.D.**.

Now we can conclude there does not exist an $\overline{f}$-factor on $\overline{G}$. If we define $S$, $T$, $L$, $U_1, U_2, \ldots, U_l$ for $\overline{G}$ as in the previous section, our new vertex $s$ will be in $L$ for the first two cases, and in $T$ for last two cases. When we restrict this partition to $G$, we can call it a **fixed size $(g, f)$-barrier.**

## 4.4 Augumenting Walk Theorem

Assume $G = (V, E)$ is a simple graph with capacity vector $u \equiv 1$. This assumption is only for brevity's sake; our result actually holds for multigraphs. Let a subgraph $M$ be a $(g, f)$-factor found by any known algorithm. Then an augumenting walk w.r.t. $M$ is defined as below.

**Definition:** A walk $P = v_0 e_1 v_1 e_2 v_2 \ldots v_{2n} e_{2n+1} v_{2n+1}$ is called an **augumenting walk** if we can add 1 on $e_1$, subtract 1 on $e_2$, add 1 on $e_3, \ldots$, subtract 1 on $e_{2n}$, add 1 on $e_{2n+1}$ and still get a $(g, f)$-factor.

Note if $P$ is an augumenting walk, both $v_0$ and $v_{2n+1}$ must be either an LTUB or a SB (see Figure 4.32). An augumenting walk can help get another $(g, f)$-factor with size increased by 1. By proving an augumenting walk theorem, we ensure that a maximum size $(g, f)$-factor can always be found. We are going to prove this theorem by developing a walk growing algorithm. Note this algorithm can also be used to find an augumenting walk if one exists.

**Theorem 4.7:** If $M$ is a $(g, f)$-factor such that no augumenting walks w.r.t. $M$

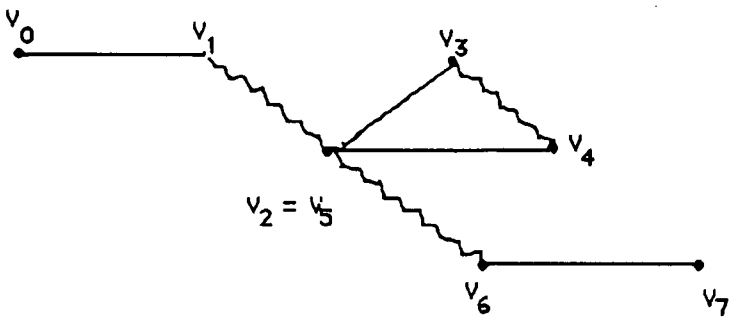


Figure 4.32: An example of augumenting walk

exist, then $M$ is a maximum size $(g, f)$-factor.

**Proof:** Suppose $\overline{M}$ is a maximum size $(g, f)$-factor, we only need to show $|\overline{M}| \leq |M|$. Consider

$$H = M \triangle \overline{M}$$

For any component $D$ of $H$ if we can show $|\overline{M}_D| = |\overline{M} \cap D| \leq |M_D| = |M \cap D|$, then we can conclude that $|\overline{M}| \leq |M|$. We show this by the following algorithm.

- Step 1: If $H$ is empty: stop; else: pick any component $D$ of $H$ and go to Step 2.
- Step 2: If for all $v_i \in V(D)$ : $deg_{\overline{M}}(v_i) \leq deg_M(v_i)$ then replace $H$ by $H \setminus D$ (since in this case $|\overline{M}_D| \leq |M_D|$). Else: pick an vertex $v_0 \in V(D)$ such that $deg_{\overline{M}}(v_0) > deg_M(v_0)$ and go to Step 3.
- Step 3: Find a vertex $v_1 \in V(D)$, s.t. $\{v_0, v_1\} \in \overline{M}$. Vertex $v_1$ cannot be an LTUB or a SB (w.r.t. $M$), otherwise an augumenting walk is found; so there must be a vertex $v_2 \in V(D)$, s.t. $\{v_1, v_2\} \in M$. We keep visiting edges of $\overline{M}$ and of $M$ alternately until we are forced to stop. Finally we obtain an alternating walk $P = v_0 e_1 v_1 e_2 v_2 e_3 v_3 \ldots e_{2k} v_{2k}$ and go to Step 4.

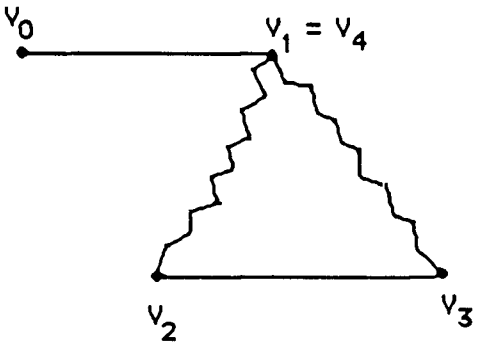


Figure 4.33: An example showing terminal vertex $v_4$

Note: $P$ must be of even length and none of $v_1, v_3, ..., v_{2k-1}$ could be either an LTUB or a SB, otherwise an augumenting walk will be found. Moreover we must have $deg_{\overline{M}}(v_{2k}) < deg_M(v_{2k})$, otherwise $v_{2k}$ can not be the terminal vertex. In Figure 4.33 $v_4$ is a terminal vertex where a straight line denotes an edge of $|\overline{M}|$ and a zigzag line denotes an edge of $M$.

- Step 4: First, do $f_i = f_i - 1$ for i = 1 to 2k. Second, delete $e_1, e_3, ..., e_{2k-1}$ from $\overline{M}$ and delete $e_2, e_4, \ldots, e_{2k}$ from $M$. Namely we replace $H$ by $H \setminus P$. Go to Step 1.

Decreasing $f_i$'s by 1 except $f_0$ makes each $M$-matched vertex have the same deficiency as before deleting $P$ from $H$. Since edges of $H$ decrease montonically, our algorithm only takes $O(n^3)$ operations. When the algorithm stops we can deduce $|\overline{M}| \leq |M|$ because each time we delete $P$ from $H$ we are deleting edges of $\overline{M}$ and of $M$ in pairs. **Q.E.D.**

**Remark 1:** We can define a decreasing walk similarly (see Figure 4.34). A decreasing walk can help get another $(g, f)$-factor with size decreased by 1. A theorem can also be proved that if $M$ is a $(g, f)$-factor without any decreasing walks, then $M$ is a minimum size $(g, f)$-factor.

**Remark 2:** The size of all feasible $(g, f)$-factors forms an interval, i.e., a $(g, f)$-factor

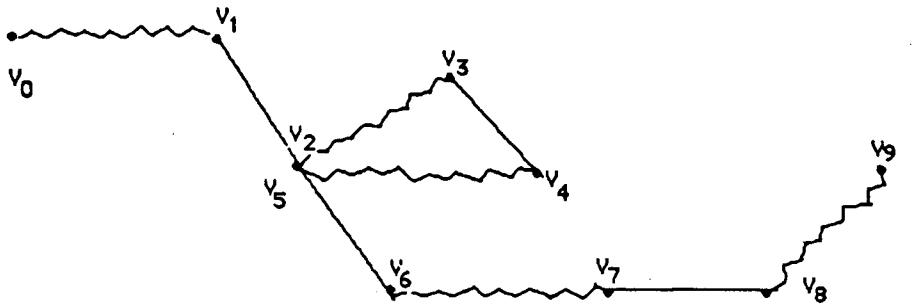


Figure 4.34: An example of decreasing walk

of any integral size between $a$ and $b$ exists where $a$ is the size of a maximum size $(g, f)$-factor and $b$ is the size of a minimum size $(g, f)$-factor. If we want a $(g, f)$-factor of some fixed size $p$, we could start with any feasible $(g, f)$-factor $M$ and apply the augumenting (or decreasing) walk algorithm to achieve this size; if we fail we know that no $(g, f)$-factor of size $p$ exists.

Note that we could use this result and binary search on size using the fixed size $(g, f)$-factor algorithm as an alternative approach to determine the interval is a specific instance. This approach would often be more efficient.

## Bibliography


[1] R.P. Anstee, An algorithmic proof of Tutte's $f$-factor theorem, J. Algorithms 6(1985)112-131

[2] R.P. Anstee, A polynomial algorithm for b-matchings: An alternative approach, Information Processing Letters 24(1987) 153-157,North-Holland.

[3] R.P. Anstee, Simplified existence theorems for $(g, f)$-factors, Discrete Applied Mathematics 27(1990)29-38, North-Holland.

[4] J.A. Bondy and U.S.R. Murty, *Graph Theory With Applications*, Elsevier, New York, 1976.

[5] J. Edmonds and E.L. Johnson, Matching: A well solved class of integer programs, Summary in: Combinatorial Structures and Their Applications (Gordon & Breach, New York, 1970)89-92.

[6] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York, 1976,

[7] L. Lovász, Subgraphs with prescribed valencies, J. Combin. Theory 8(1970)391-416.

[8] W.R. Pulleyblank, Faces of matching polyhedra, Ph.D. thesis, Dept. of Combinatorics and Optimization, Univ. of Waterloo, 1973.

[9] Christos H. Papadimitriou, Kennneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall Inc, Englewood Cliffs, NJ 07632, 1984.

[10] R.L. Urquhart, Degree constrained subgraphs of linear graphs, Ph.D. Thesis, The Univ. of Michigan,1967.