Vertex Enumeration and Counting for Certain Classes of Polyhedra

by

Sammani Danwawu Abdullahi

Submitted in accordance with the requirement for the degree of Doctor of Philosophy

The University of Leeds
School of Computing

September 2002

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others

Abstract

Vertex enumeration (VE) algorithms explore the problem of listing some or all solutions that lie at corners of a convex polyhedron defined by a set of linear inequalities. Many algorithms have been developed for general polytopes. The most successful of these, from both an empirical and theoretical viewpoint, are based on pivoting. Dyer [24] gives an algorithm for simple polytopes which runs in time $O(mn^2)$ per vertex. In this thesis we concentrate on the VE problem for certain special classes of polyhedron. We also address the problem of (approximately) counting the vertices without listing them.

Pivoting algorithms rely on the correspondence between vertices and feasible bases and are consequently inefficient in the presence of a high degree of degeneracy such as frequently occurs in network polyhedra. Provan [79] gives a high-level description of a VE algorithm for such polyhedra which has running time that is quadratic in the number of vertices. We describe an implementation of Provan's algorithm, present some computational results on transportation and assignment polytopes and discuss some practical difficulties with the algorithm. We then present an algorithmic description of a VE method via the dual Fourier-Motzkin (*F-M*) elimination method. One of the difficulties with *F-M* is that the number of constraints introduced in eliminating variables grows exponentially; we show that, for linear inequality systems with at most two variables per constraint, denoted *LI(2)*, the growth is exponential in the number of variables but linear in the number of constraints. We go on to prove results which characterize the basis structure for *LI(2)* and dual *LI(2)* systems and hence develop a new pivoting algorithm for enumerating vertices of polyhedra associated with dual *LI(2)* systems.

Counting the vertices of general polyhedra is $\#P$-complete [24] and thus *approximate counting* procedures are of interest. In particular, some *fpras* for counting vertices of polyhedra associated with *0-1* Permanent, Down Sets, Independent Sets, *0-1* Knapsack Problems, $2 \times n$ transportation problems, matroids and matchings in a non-bipartite graph are developed.

Acknowledgements

I start by expressing my special gratitude and appreciation to my supervisors; Professor Martin Dyer (the prestigious *Fulkerson* prize winner and one of the organisers of Computation, Combinatorics and Probability programme at the *Isaac Newton Institute in Cambridge*, July 2002 - December 2002) and Dr. Les Proll for boosting my morale which resulted in increased in self confidence. Especially for their special assistance and support during the difficult moments of my research.

Same goes to Professor J. Ezeolo (former Director National Mathematical Center, Abuja) who recommended me for a Special Federal Government Scholarship Award and approved by the then Nigeria's honorable Minister of Education, a person of Dr. M.T. Liman. It was initially supposed to be undertaken at *Oxford University*, and later changed to School of Computing University of Leeds, which is also one of the *leading international research Institutes of Computing*. Similar appreciation goes to Bayero University, Kano for awarding me a fellowship.

Many thanks are due to Dr. George Feeman for recommending me to work as an Instructor / Lecturer at Information Technology Unit, University General Requirements Unit (UGRU), United Arab Emirates University where Dr. Graham Zelmer gave me a moral support, under the leadership of UGRU directors; Dr. Salah El-Nahwy, Dr. Khalifa Al-Suwaidi and Dr. Abdulla Al-Khanbashi. This has given me an opportunity to sponsor the remaining parts of my studies, and acquired valuable experiences.

Finally, all thanks are due to my *Creator*, Who made it possible for me to complete this thesis which is a **life-time** ambition. Many thanks to my Mum, Hajia Ummul-Khurthum D. Abdullahi for her prayers and moral supports and to my wife Hajia Ummul-Khair S. Abdullahi and daughter Hajia Fatima Sammani Abdullahi for their patience.

Dedication

This work is **dedicated** to my beloved late father **Alhaji Muhammad Danwawu Abdullahi** who had encouraged me to pursue graduate studies, but unfortunately life cut short and is unable to witness its completion.

Declarations

Some parts of the work presented in this thesis have been published in the following articles:

Sammani Abdullahi, Martin Dyer, Les Proll, Enumerating vertices of network polyhedra, "ISMP2000", 17^{th} *International Symposium on Mathematical Programming hosted by the Georgia Institute of Technology*, U.S.A., August 7-11, 2000.

Contents

List of Figures

List of Tables

Chapter 1

The Problem

1.1 Introduction

Optimization is concerned with maximizing a function of certain variables (the objective function) possibly subject to inequality or equality constraints on other functions of the same variables (constraints). A problem of this type is called a mathematical program, and the methods for its solution are called mathematical programming. Such problems have been, and continue to be, extensively studied because they provide useful models for many real-world problems.

Generally, the simplest classes of mathematical programming problems are classified as those in which the variables are subject to only linear equalities (inequalities). If such a problem has n real valued variables, then the constraints define a region in n-space, $\mathbb{R}^n$, which is a convex polyhedron. A characterization of such a region is being that of the convex hull of a certain set of *extreme points* and unbounded half-lines in $\mathbb{R}^n$. The extreme points are also called *vertices* and the half-lines are called *extreme edges* or *rays* of the polyhedron. A polyhedron which is bounded as a subset of $\mathbb{R}^n$ is also known as a polytope, which is simply the convex hull of its

vertex set. Many mathematical programming problems (including LP of course) can be shown to have solutions lying at a vertex of a convex polyhedron defined by some (or all) of the constraints.

For the vertex sets that are finite, methods are suggested for tackling such problems based on partial, or even total, determination of the vertices of the polyhedron. This kind of procedure is called a vertex enumeration (VE) method, and the specific problem of enumerating all vertices is called the VE problem. This is one of the problems studied in the thesis.

The VE problem has a long history and many methods for its solution have been proposed. Two variants of the problem have been identified by Chen et al. [10] who presented on-line and off-line algorithms for enumerating vertices of polyhedron defined by the system (1.1). The on-line problem is particularly important in deterministic algorithms for global optimization [43], while the off-line problem is the one in which P is fully defined *ab initio*. algorithms for the on-line problem can be used to solve the off-line problem and vice-versa.

It is interesting to explore all the means of generating such points, especially when the polyhedron has degenerate vertices. This has practical applications in such areas as computational geometry, dual representations, sensitivity analysis, geometrical optimization and so on. Despite the commendable efforts of Dyer [24, 30] and Chen et al. [10] the problem of degeneracy is still open to competing research in the sense that there is no algorithm for general polytopes which is polynomial in the number of vertices.

The problems caused by degeneracy should not be under-estimated. A vertex v is said to be degenerate if more than n of the binding hyperplanes intersect at it. Many algorithms have been developed for vertex enumeration, notable among which are; [2, 4, 9, 23, 24, 27, 28, 30, 56, 66, 79, 86, 91]. Some of these have good theoretical and empirical efficiencies. However, most of them are inefficient for highly

degenerate polyhedra.

Competing research about solving the problem of degeneracy is as the time of writing open from both theoretical and practical point of view. The theoretical problem is to list the vertices in a time which depends only polynomially on m and n and the number of distinct vertices of the polyhedron. The practical problem is to do this in a feasible computational time.

Geometrically, the above method (VE) can be equivalently stated in dual concept as that of finding the smallest convex set containing a given set of m points in $\mathbb{R}^n$ (the *convex hull*). This has become the central problem in computational geometry. It is a basic intuitive problem and appears frequently as a sub-problem in the solution of other geometrical problems and a number of other important problems such as intersecting half-spaces or constructing *Voronoi Diagrams* have all shown to be convex hull problems in disguise.

Most algorithms for computing the convex hull require maintenance and a description of the whole convex hull or at least of all its facets. Since the number of facets could grow exponentially in higher dimensions, these algorithms, e.g. Seidel [88], Swart [93], appear to exceed the capacity limits of even today's most powerful computers.

In combinatorial optimization one is interested in describing the facets of polytopes whose vertices correspond to combinatorial objects. This opens the possibility of doing optimization over these objects by linear programming (LP) and cutting plane algorithms (Polyhedral Combinatorics). Examples of these are the popular and the very challenging Travelling Salesman polytopes or Cut Polytopes. The polytopes that arise in this area are highly degenerate. Examples of degenerate and non-degenerate polytopes are available in Figure 1.1. Certain highly regular graphs can be described as the skeletons of regular polyhedra. Let us now review some necessary backgrounds and definitions.

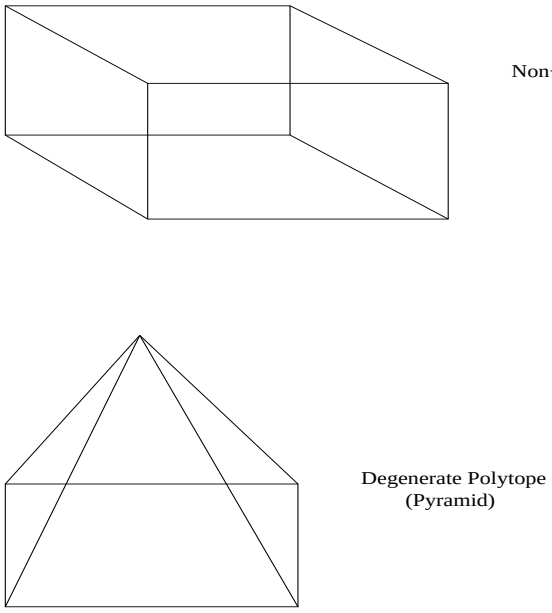


Figure 1.1: **Degenerate and non-degenerate polyhedra**

1.2 Background and Definitions

A polyhedron P is defined to be a set satisfying the following:

$$P = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_{ij}x_j \leq b_i,\ \ i = 1, .., m+n\} \tag{1.1}$$

where a_{ij}, b_i are rational constants and $\mathbb{R}^n$ is real n space. Then, a vertex v of the above polyhedron P is defined to be the unique point of intersection of at least n of the binding hyperplanes H, where H is defined as follows:

$$H_i = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_{ij}x_j = b_i\} \quad i = 1, ...m+n \tag{1.2}$$

A hyperplane divides $\mathbb{R}^n$ into two regions called half-spaces (details of some basic definitions will be available at a later stage in this chapter). A 'typical' case of P is that each vertex is the intersection of exactly n hyperplanes. Such a polytope is called simple, or non-degenerate, and many VE algorithms are directed principally at this class of polyhedra.

Let us review and introduce here some notation and terminology which will be freely used later in this thesis. It is assumed that the reader is familiar with the

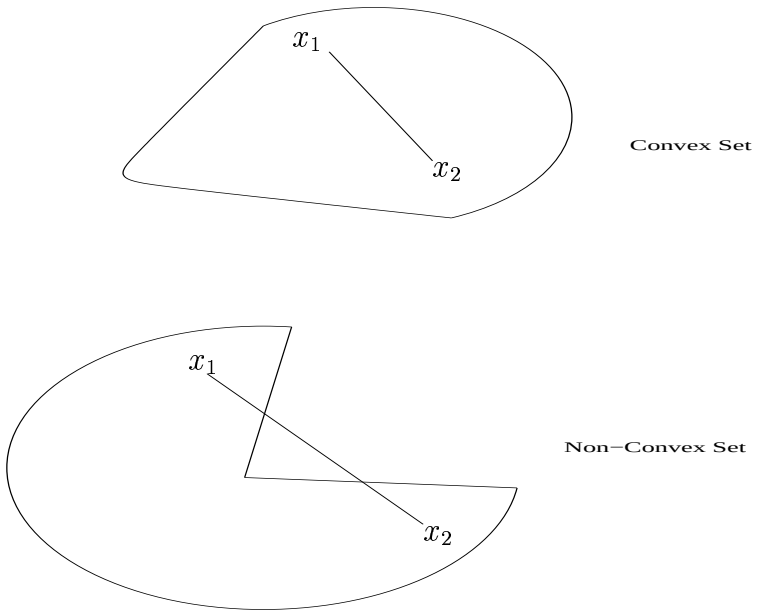


Figure 1.2: **Convex and non-convex sets**

basic theory of linear programming (LP) as is given in such books as: Hadley [41], Jarvis et al. [63] and Williams [99] .

The n-dimensional real space is denoted by $\mathbb{R}^n$. A set X in $\mathbb{R}^n$ is called a *convex set* if given any two points x_1 and x_2 in $\mathbb{R}^n$, then $(\lambda x_1 + (1-\lambda)x_2) \in X$ for each $\lambda \in [0,1]$. It should be noted that $\lambda x_1 + (1-\lambda)x_2$ for λ in the interval $[0,1]$ represents a point on the line segment joining x_1 and x_2. Any point of the form $\lambda x_1 + (1-\lambda)x_2$ where $0 \leq \lambda \leq 1$ is called *convex combination* (or weighted average) of x_1 and x_2. Figure 1.2 is an illustration of the basic concept explained above.

The concept of extreme points or vertices plays an important role in the theory of linear programming (LP). By definition a point x in a convex set C is called an extreme point of C if x cannot be represented as a strict convex combination of two distinct points in C. In another words, $x = \lambda x_1 + (1-\lambda)x_2$ with $\lambda \in (0,1)$ and $x_1, x_2 \in X$ implies $x = x_1 = x_2$. Figure 1.3 is an illustration of this concept explained.

A hyperplane is considered to be a generalization of the idea of a straight line in

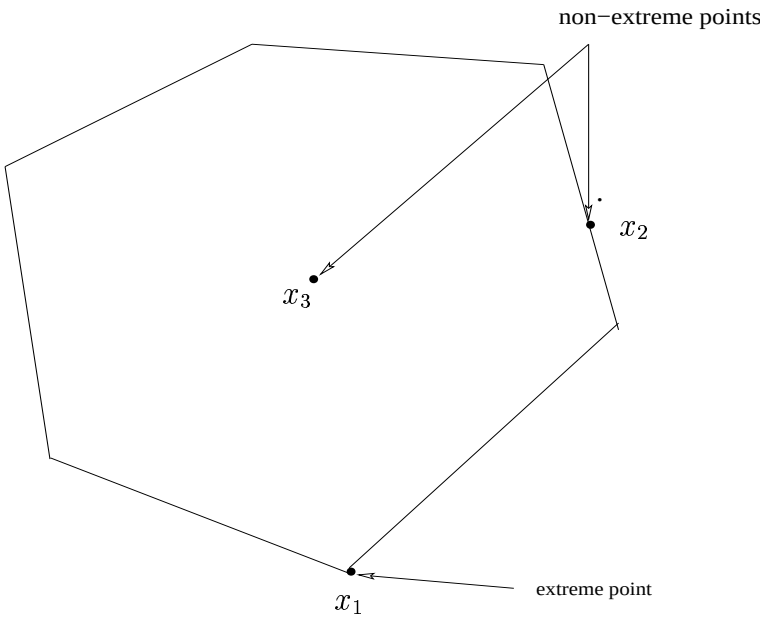


Figure 1.3: **Extreme and non-extreme points**

two dimensional space, $\mathbb{R}^2$ and that of plane in three dimensional space, $\mathbb{R}^3$. A hyperplane $H' \in \mathbb{R}^n$ is a set of the form $\{x : wx = k\}$ where w is a non-zero vector in $\mathbb{R}^n$ and k is a scalar, w is sometimes called the gradient or normal to the hyperplane. Equivalently, a hyperplane consists of all points $x = (x_1, x_2, ..., x_n)$ satisfying the equation $\sum_{j=1}^{n} w_j x_j = k$, that is:

$$H = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} w_j x_j = k\} \tag{1.3}$$

Note that equations (1.2) and (1.3) are analogous. Hence, a half-space is a collection of points of the form $\{x : wx \leq k\}$ that is:

$$H_1 = \{x : wx \geq k\} \tag{1.4}$$

$$H_2 = \{x : wx \leq k\} \tag{1.5}$$

The union of the two half-spaces is the entire space in $\mathbb{R}^n$. This is illustrated in Figure 1.4.

Geometrically, a point $x \in C$ is also said to be an extreme point, or a corner point or a vertex of C if x lies on some n linearly independent defining hyperplanes say,

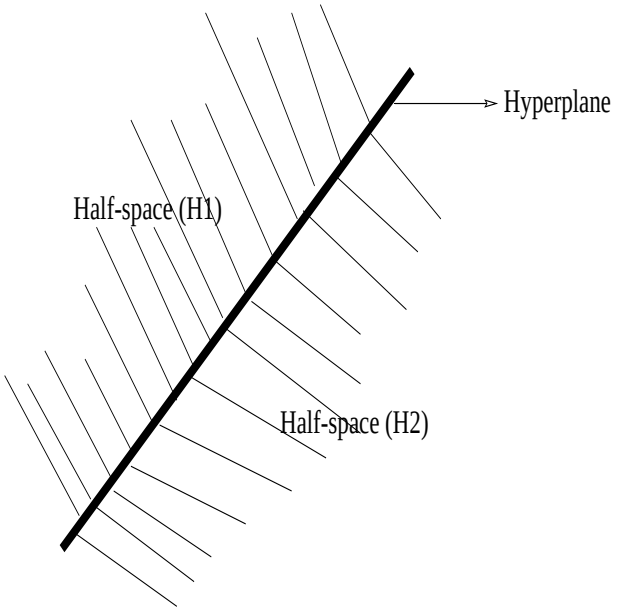


Figure 1.4: **Hyperplane and half-space**

H_i, $i \in \mathbb{N}$ of C. If more than n defining hyperplanes pass through an extreme point (vertex) such a point is called *degenerate extreme point* or *degenerate vertex*. The excess number of planes over n is called its order of degeneracy. A polyhedron with at least one degenerate vertex is said to be a *degenerate polyhedron*.

The practical uses of VE will at least depend on the number of vertices the polyhedron possesses. Indeed, the complexity of the VE problem will obviously depend critically on this. In that respect there are known theoretical bounds. McMullen [68] has proved that the maximum number of vertices a polytope defined by m inequalities in n non-negative variables can possess is given by:

$$V_{max} = \begin{pmatrix} m + \lfloor \frac{1}{2}n \rfloor \\ m \end{pmatrix} + \begin{pmatrix} m + \lceil \frac{1}{2}n \rceil - 1 \\ m \end{pmatrix} \tag{1.6}$$

Note: $\lfloor y \rfloor$ is the largest integer not exceeding y and $\lceil y \rceil$ is the smallest integer not exceeded by y.

On the other hand, Barnette [6] has shown that, at least for non-degenerate poly-

topes, the minimum number of vertices is given by:

$$V_{min} = m(n-1) + 2 \tag{1.7}$$

Note: the assumption here is that all $m + n$ inequalities are non-redundant.

For example, a linear program with $m = 10$ and $n = 20$ will have a minimum number of vertices $V_{min} = 192$, according to (1.7). On the other hand, by (1.6), the same problem will have a maximum number of vertices $V_{max} = 277134$. It can be noted that the gap between the lower and upper bounds is extremely large. Hence, these bounds are not a particularly good guide to the number of vertices to expect in a given polytope.

The dual analogue of VE is the problem of finding the smallest convex set containing a given set of m points in $\mathbb{R}^n$, the convex hull. In this direction it may be worthwhile to review some related definitions: for a set $S \in \mathbb{R}^n$ the *affine hull* is the intersection of all hyperplanes that contain S; the convex hull of S (for short $Conv(S)$) is defined to be the intersection of all half-spaces that contain S; a polytope in this concept is defined to be the convex hull of a finite set; a polytope P is a k-polytope if k is the dimension of the smallest *affine subspace* of $\mathbb{R}^n$ that contains P; a k-simplex is a k-polytope formed by the convex hull of $k+1$ points in $\mathbb{R}^n$ with $n \geq k$; a k-simplex has $k+1$ facets and $\left[\begin{array}{c} k+1 \\ 2 \end{array} \right]$ ridges; a polytope is called *simplicial* if every face of P is a simplex.

The dual equivalent upper bound of McMullen [68] for the maximal number of k-faces for a d-polytope with n number of vertices is given as a theorem known as the *Upper Bound Theorem*: If P is a d-polytope with $n = f_0$ vertices, then for every k it has at most as many k-faces as the corresponding cyclic polytope, i.e: $f_{k-1}(P) \leq f_{k-1}(C_d(n))$, where $f_k = f_k(P)$ denotes the number of k-dimensional faces in P and d is the dimension of polytope, n is the number of vertices, $C_d(n)$ is the cyclic polytope and k is the number of faces with $\lfloor \frac{d}{2} \rfloor \leq k \leq d$ implying P is neighborly. (See [101]).

Barany [60] suggested that the maximal number $f(d)$ of facets of d-dimensional *0-1*-polytope is given by: $2^d \leq f(d) \leq d! + 2d$. Seidel [87] shows that (by asymptotic argument) the number of facets is at most twice the number of k-faces of general polytope P with $k \leq \lfloor \frac{d}{2} \rfloor$ i.e:

$$f_{d-1} \leq 2 \sum_{i=0}^{\lfloor \frac{d}{2} \rfloor} \left(\begin{array}{c} n \\ i \end{array} \right) \tag{1.8}$$

where $i = \frac{d}{2}$ if d is even and $i = \frac{d-1}{2} = \lfloor \frac{d}{2} \rfloor$ if d is odd. Thus, Seidel has given a rough estimate bound f_{d-1}, with a polynomial of degree $\lfloor \frac{d}{2} \rfloor$.

1.3 Algorithms for Vertex Enumeration

A number of algorithms have been proposed for the solution of the basic problem considered here, that of automatically enumerating all vertices of a convex polyhedron formed by the finite system of inequalities given in (1.1).

While much research has been concentrated on convex polytopes (see e.g Gunbaum [40]), it appears that, in general, they have few useful properties that can be exploited in order to enumerate easily their extreme points (vertices).

Dyer [23] was able to classify some of the related published algorithms, examples are available in [2, 4, 6, 7, 9, 10, 11, 12, 23, 24, 27, 28, 30, 38, 42, 56, 66, 67, 79, 86, 91, 93] and so numerous to mention. Dyer [23] divides some of these algorithms into two major categories, and further divided the second category into two main classes in accordance with their computational schemes/methods of *book-keeping*. These are the accounting systems which must be incorporated into the algorithms in order to ensure that vertices are neither omitted nor repeated in the enumeration process.

Algorithms in the first category are classified by Dyer [23] as Constructive Inductive (CI) algorithms. These algorithms are usually motivated by geometrical considerations, as such it may be unimportant whether or not the polyhedra are degenerate.

CI algorithms assume that the vertices for a polytope, say P, are known and subsequent addition of a half space, say H^+, enables one to enumerate the vertices of a new polytope P^i formed by the intersection of the polyhedron P and the half space H^+, that is:

$$P^i = H^+ \cap P \tag{1.9}$$

The first published method appears to be that of Motzkin et al. [71]. Their approach is to add inequalities in a step-by-step manner, recording at each step which 'new' corner points have been created and which 'old' ones excluded. This has certain merits but the chief disadvantage is the case of many redundant constraints. Much effort may be wasted calculating extreme (corner) points that may later be excluded. Examples of CI methods can be found in [12, 36, 39, 59, 71, 76, 85, 89, 93, 94, 95]. The reader is referred to Dyer's thesis [23] for further details.

The second category are called pivoting methods. The main deriving inspiration of these algorithms is the simplex method of linear programming (LP) (see, for example, [19]). These methods exploit the correspondence between the vertices and Basic Feasible Solutions (BFS). This correspondence is one-to-one for non-degenerate polyhedra and one-to-many for degenerate one. One class of this second category comprises methods that use accounting-method based on the defining hyperplanes. These are called Hyperplane Oriented Pivoting (HOP), examples are available in [5, 8, 44]. The second class comprises methods which use book-keeping based on the feasible basis graph, called Basis Oriented Pivoting (BOP). For research in this area, see [2, 64, 65, 82, 91]. Most vertex ranking methods may also be viewed as belonging to this class. Examples are [18, 57, 72, 78].

The problem of determining all the extreme points of simple polytopes is equivalent to that of enumerating all the basic feasible solutions for the set of constraints. In the degenerate case, the polytope may be 'perturbed' by standard methods to restore the *1-1* correspondence. Simplex tableau could be used to represent the extreme point algebraically. Unfortunately, it is well known that (Mattheiss [65]) it is not, in general, possible to generate a complete non-recurrent sequence of basic

feasible solutions by simplex pivots.

The first person to adopt simplex format was Balinski [5]. He devised a flexible search method based on a careful and successive relaxation of the constraints. For this algorithm, the accounting procedure was particularly simple, however the method, like Motzkin's, is inefficient in the presence of large number of redundant constraints. This is because infeasible basic solutions might be visited. Others are Manas and Nedoma [64], who have used the simplex tableau and generate only basic feasible solutions. The 'book-keeping' for their algorithm requires a list of the indices of the basic variables at each basic feasible solution to be held. They attempt to trace an edge-path on the polytope terminating after each extreme point has been visited at least once. Silverman [91] has also devised a method along the lines which he calls the G-Path method. The main difference between the methods of Silverman and that of Manas and Nedoma is in the use of the construction of the G-path. This is an edge path on the polytope such that each extreme point either lies on the path or is adjacent to some point on the path. It can be readily observed that enumerating all extreme points of the polytope suffices to follow such a path. However, such a path may not always exist.

Mattheiss [65] adds a further 'generalised-slack' variable to the constraints and views the original polytope to be a special facet of a polytope in $(n+1)$-dimensions. The search then takes place among the extreme points of the higher dimensional polytope that lie on the special facet. The points of the original polytope are then generated as 'neighbours' of these points. He shows that redundant constraints do not affect the process, but the main reason for the additional complication rests on an unproven conjecture that fewer points will be explicitly generated than in a search of the original polytope. Burdet [8] presents an interesting method using recurrent applications of Phase I linear programming method to construct a kind of branch and bound search for the basic feasible solutions. This method will need more simplex pivoting operations than its competitors in the non-degenerate case, but could be useful in case of significant degeneracy. However, it seems that cer-

tain simplifications which he claims may be made in respect of 2-dimensional and simplicial faces will have little effect in the scheme proposed, and is easily ignorable.

Another interesting approach is that of Chernikova [12], who gives an algorithm for determining the extreme edges of a convex polyhedral cone. This can be adopted in a straightforward manner to find the vertices of a convex polyhedron. The idea is comparable in many respects to that of Motzkin et al. [71] but is presented in a slightly different fashion. This method is also discussed by Rubin [83], who presented its application to cardinality constrained linear programming. Greenberg [39] presents a method for computing all edges of a cone. His work is based on a theorem from Uzawa [95], which is strengthened by incorporating certain exclusion tests on candidate edges. The technique again has similarities with that of Motzkin et al. [71].

Other suggestions have been made by Avis and Fukuda [4], Hadley [41] and Chen et al. [10] and many more. In general little attention was given to the computational efficiency of the algorithms discussed above until Dyer and Proll [27] have presented a method based on the simplex method using the format of the product form revised simplex method [41]. They constructed a spanning tree in the feasible basis graph of the polytope, rather than using single tableau to follow an edge path.

Dyer's algorithm [24] is still the fastest in term of empirical evaluations, especially, since a hashing method was incorporated as 'book-keeping' into his algorithm by Ong et al. [77]. The problem of degeneracy is still an open problem. Although the 'perturbation' technique of Dyer [24] worked satisfactorily for polytopes with a small degree of degeneracy, but it still has difficulty with polytopes that have high degree of degeneracy.

Some of the short-comings of VE algorithms could be summarised as the case may be, that is, CI methods have one advantage over other methods: the ability to directly enumerate vertices of degenerate polyhedra. However, its major drawbacks

are: it is inefficienct for systems with many redundant inequalities; and inappropriate for methods such as multicriteria or multiparametric linear programming. Also, one obvious question associated with CI algorithms is "which order should the constraints be added"? No results appear to exist on this problem, although Rubin [83] has suggested a heuristic approach.

On the other hand, both HOP and BOP methods have two major drawbacks. One is their inability to efficiently enumerate degenerate vertices, and the second is that of potential repetitious enumeration of vertices.

For more details about these procedures, see Dyer's thesis [23].

1.4 Restatement of the Problem

It is possible to re-define the problem (that of automatically enumerating the vertices of polyhedra) as one in which a polyhedron, say P', in $\mathbb{R}^n$ is an input defined by a set of linear equalities and inequalities in the variables $x_1, x_2, ..., x_n$. Then, the input size describing P' is denoted by $|P'|$. If we assume that P' is non-empty (i.e. contains at least one vertex) then P' can be re-described more concisely as follows:

$$P' = \{x \in \mathbb{R}^n : \ Ax = b, \quad x \geq 0\} \tag{1.10}$$

where A is an $m \times n$ matrix with possible rank m and b is an m-vector. P' may be unbounded.

The VE problem with input P' requires as output the set $v(P')$ of extreme points of P' that can be described algebraically as follows:

$$v(P') = \{x \in P' : Ax = 0\} \tag{1.11}$$

That is, the set of corner points $x \in P'$ whose support is a linearly independent set of columns of A (see Hadley [41] or Jarvis et al. [63]).

It has been established (by Dyer, [24]) that there can exist no polynomial time algorithm that can list the elements of $v(P')$, because of the simple fact that there are easily constructible classes of polyhedra where the number of vertices grows exponentially as the description of the polyhedron, P'. Thus, it is natural to allow some algorithms to run in time polynomial in the output size $|v(P')|$.

Dyer [24] analyses the complexities of the different approaches and shows that, of the current methods, only BOP yields algorithms whose time complexity can reasonably be related to the output size. But, BOP is exactly the dual notion of *'gift-wrapping'* and the time bound derived by Dyer [24] for his implementation of a pivoting method is analogous with the $O(mF)$, (where F is number of facets) bound of Swart's [93] implementation of the gift-wrapping method.

As such, general polyhedra have unknown polynomial time solution. The only known polynomial time algorithms make the strong assumption that the polyhedron is simple, meaning that each $x \in P'$ has support of cardinality exactly m. In this case the vertices of P' are in one-to-one correspondence with the feasible bases of the underlying linear system, that is, the non-singular $m \times m$ submatrices B of A with $B^{-1}b \geq 0$.

"The feasible bases in turn can be efficiently enumerated by means of a pivot scheme (by using either HOP or BOP or similar methods as explained earlier). A pivot scheme starts by finding some initial feasible basis of the system (1.1) by solving a linear program and a search is performed of the pivot graph of the polyhedron, identifying the edges adjacent to a given vertex by means of feasible pivots in the associated linear system. Dyer [24] gives detailed description of a pivot scheme which lists the vertices of a simple polyhedron in time $O(mn^2)$ per vertex listed". See Provan [79].

As indicated earlier both HOP and BOP of the pivoting schemes for enumerating vertices have a serious drawback in the case where the polyhedron P' is not simple -

in this case the number of feasible bases can grow exponentially with respect to both $|P'|$ and $|v(P')|$. Then pivot schemes require exponential time to enumerate the distinct vertices. Provan [79] has provided two examples that exhibit such a behaviour.

The contributions of Dyer [24] has served as an improvement over earlier methods. However, despite his perturbation technique [24], the problem of degeneracy is still open from both theoretical point of view and practical viewpoint. The theoretical problem is to list the vertices in a time which depends only on m and n and the number of distinct vertices of the polyhedron. The practical problem is to do this in a feasible computational time.

Therefore, as stated earlier evolving a procedure (be it VE or counting algorithm) which has both theoretical and practical efficiencies and which could explore the possibility of solving problems of degeneracy is what motivates current research in this very interesting and challenging field.

1.5 Research Objectives

In this chapter we have introduced the vertex enumeration problem and briefly discussed various algorithms for its solution. The numerous algorithms cited are aimed at determining the vertices of polyhedra defined by systems of general linear inequalities. It is well known that, in an optimisation context, taking advantage of the structure of particular systems of constraints has led to algorithms having vastly improved performance over the simplex algorithm. The assignment, transportation and maximal flow problems provide obvious examples [19]. Our research is broadly aimed at investigating whether exploiting the properties of particular subclasses of polyhedron can bring any advantage for the vertex enumeration/counting problem. Provan's work on network LPs [79] suggests that this could be a useful line of research. He has developed a VE algorithm for such, frequently degenerate, polyhedra which is quadratic in the number of vertices. As there is not known to be a VE algorithm which is polynomial in the number of vertices for general, non-simple

polyhedra, this is a significant result.

Clearly the computational load of any VE algorithm is heavily dependent on the number of vertices so it would be useful to know this in advance of applying the algorithm. However there is unlikely to be an efficient algorithm for determining this number exactly as the problem is $\#P$-complete [24]. For most polyhedra, McMullen's upper bound on the number of vertices is too loose to be of much use so we turn our attention to ways in which the number of vertices might be approximated.

The purpose of this research therefore can be summarized as:

1. To develop VE algorithms for some special classes of polyhedron;
2. To investigate whether vertices of some special classes of polyhedra can be approximately counted.

1.6 Organisation of the Thesis

As mentioned in the previous section Provan has provided a significant result for VE in a particular class of polyhedra. He describes [79] algorithms for VE in polyhedra arising from network LP's and their duals, which have good theoretical efficiency. The description is at high-level and requires careful implementation. In Chapter 2, we review Provan's algorithm for primal network polyhedra and discuss our implementation, some computational experience with it and make some observations about its empirical performance. A summary of this work was presented at the 17^{th} International Symposium on Mathematical Programming (ISMP), August 2000.

Chapter 3 introduces *LI(2)* systems, i.e. a set of linear inequalities with at most two non-zero coefficients in each inequality, which are related to dual network LP's. We survey several algorithms for the feasibility problem in *LI(2)*, i.e. the problem of finding a solution to the system or showing that no solution exists. It is not obvious how to extend these algorithms to provide a VE algorithm for *LI(2)*, indeed it is not even clear that they guarantee to find a vertex solution. Nevertheless, they

provide some clues on useful avenues to explore.

One such avenue is Fourier-Motzkin (*F-M*) elimination, which is the backbone of Hochbaum and Naor's algorithm [42] for the *LI(2)* feasibility problem. We describe an algorithm for VE using the dual *F-M* method in Chapter 4 and analyse its complexity for dual *LI(2)* systems.

In Chapter 5 we develop propositions concerning the structure of the basis for *LI(2)* and dual *LI(2)* systems. We then exploit one of them to develop a basis oriented pivoting (BOP) algorithm for VE in dual *LI(2)* in the case where the polyhedron is simple and bounded. We show, through a comprehensive example, how the algorithm works and later how it can be extended to deal with unbounded edges and degeneracy issues.

The problem of vertex counting is discussed in Chapter 6. Following a review of relevant background material on approximate counting, we discuss the development of *fpras* for counting the vertices of certain classes of polyhedra. We also raise several related open questions.

A summary of the research and some recommended future work are presented in Chapter 7.

Chapter 2

Implementation of Provan's Algorithm

2.1 Introduction:

A network LP is an LP whose variables are associated with flow in the arcs of a directed graph and whose constraints are associated with conservation of flow at the nodes of the graph. Such problems include the well-known transportation and assignment problems [63]. Algorithms which take advantage of the special structure of the coefficient matrix of network LP's heavily outperform the simplex method, see for example [55]. It is well known that such problems are inherently degenerate and thus enumerating the vertices of network LP's will present difficulties for pivoting methods designed for general polyhedra. For example, the 3 assignment polytope has only 6 vertices, but the corresponding LP has 54 basic feasible solutions (BFS). Presented with an unperturbed problem, Dyer's algorithm does not terminate. The degeneracy-protected variant of Dyer's algorithm does but produces 3 copies of each vertex. Thus it seem certain that, in order to enumerate the vertices of network polyhedra, it will again be necessary to devise algorithm which take advantage of

their special structure. Provan [79] has provided such an algorithm and examines its complexity. He shows that the vertices of the network polyhedron can be listed in time $O(|E||v(P)|^2)$, plus the time needed to find the first vertex of the polyhderon P.

From a theoretical point of view Provan's work is an improvement over some preceding methods discussed in chapter one which may have running time which is exponential in the number of objects (vertices) enumerated for degenerate polyhedra. In contrast, the running time for Provan's algorithms is polynomial and the polyhedra need not be simple or bounded.

Therefore, Provan's algorithm [79] tends to theoretically solve the problems of degeneracy in some transportation and assignment polytopes. But its empirical evaluations are yet unknown. Hence, an implementation is one of the goals of this research. This has been done in four sections, the first one is the outline of Provan's algorithm, second is detail about implementation, and the third is the computational results, and last is the conclusion.

2.2 Basic Definitions and Terminology from Graph Theory

In this section we attempt to give some basic definitions, based on graph theory, for detail information the reader is referred to Jarvis et al. [63]. A directed network (or *digraph*) G is a graph that consists of *nodes* $V = \{1, 2, 3, ..., m\}$ and a set of directed *arcs* $E = \{(i, j), (k, l), ..., (s, t)\}$ joining pairs of nodes in V. Arc (i, j) is said to be *incident* at nodes i and j and is directed from node i to node j. The digraph $G(V, E)$ is called *proper* if the cardinalities of V and E satisfy $|V| \geq 2$ and $|E| \geq 1$. Two nodes in the graph are said to be *adjacent* if they are directly connected by some arc, the node i in the directed arc (i, j) is called its *from-node* and j its *to-node*. A *path* from node i_o to i_p is a sequence of arcs $P = \{(i_o, i_1), (i_1, i_2), ..., (i_{p-1}, i_p)\}$. A *chain* is a similar structure to a path except that not all arcs are necessarily directed toward node i_p. A *circuit* is a path from some node i_o to i_p plus the arc (i_p, i_o), i.e

a circuit is *closed path*. Similarly, a *cycle* is a *closed chain*.

Graph $G(V, E)$ is said to be *connected* if there exists a chain between every pair of nodes in G. This property is also known as *weakly connected* graph. A *strongly connected* graph is one where there is a directed path from each node to every other node. A *complete* graph is one where each node is connected by an arc to every other node. A *subgraph* $G'(V', E')$ of a graph $G(V, E)$ is one that satisfies $V' \subseteq V$ and $E' \subseteq E$, with the understanding that $(i, j) \in E'$ if both i and j are in V'. If $G' \neq G$ then $G'(V', E')$ is called *proper subgraph* of $G(V, E)$. If $V' = V$ then G' is called a *spanning subgraph* of G. A *component* of a graph $G(V, E)$ is subgraph that is connected and that is not a proper subgraph of another connected subgraph, i.e they are *maximal connected subgraphs* and are also called *separable pieces* of a graph.

A *tree* is a connected graph with no cycles. A *spanning tree* with respect to the graph $G(V, E)$ is a tree that includes every node of the graph. The *degree* of a node is the number of arcs incident at it.

2.3 The Algorithm

An attempt has been made to differentiate Provan's algorithm as it appears in [79] and some of the vague steps not mentioned in [79] that are necessary for the computational implementation of the algorithm. These are categorized in the following sections, but first the algorithm. Provan presents two algorithms that use a characterization of adjacent vertices in network and dual network LP in order to perform a traversal search of the edge graph of the polyhedron.

The general network LP has as parameters a directed loopless graph $G(V, E)$, vertex supplies/demands b_i, $i \in V$ and edges costs a_{uv}, lower/upper bounds, $l_{uv} < c_{uv}$ (infinite lower and upper bounds, but not both are allowed). It is a minimization problem in the variables x_{uv}, $(u, v) \in E$ in the form of equations: (2.1) is the objective function, (2.2) is known as flow conservation or nodal balance or Kirchhoff

equations and (2.3) are the bounds.

$$Minimize \sum_{(u,v)\in E} a_{uv}x_{uv} \tag{2.1}$$

subject to:

$$\sum_{u:(u,v)\in E} x_{uv} - \sum_{u:(v,u)\in E} x_{vu} = b_v \tag{2.2}$$

$$v \in V, l_{uv} \leq x_{uv} \leq c_{uv}, (u,v) \in E \tag{2.3}$$

In order to perform the edge-graph search for the network LP algorithms, a procedure has been established by Provan [79] that list the adjacent vertices of each vertex v of polyhedron P. The edge-graph search is a special case of pivoting. It starts at some initial vertex of P found by solving the appropriate linear program (in this case network LP). It is noted that at each stage a list of *vertices* rather than bases is kept together with a sublist of those vertices left to be processed. In a given processing step the procedure chooses a vertex v to be processed, but first identifies the edges of P adjacent to v rather than simply the feasible bases adjacent to the basis chosen to represent v, so that the second vertex if any is identified. An accounting procedure is incorporated to check which of these vertices has been processed (note that in this case Provan [79] did not give detail about the accounting procedure), and the new vertices are added to the list of vertices to be processed. When the list of vertices is exhausted the procedure terminates with all of the vertices listed.

Since adjacent vertices rather than adjacent bases are identified at each stage, such a procedure will operate in time polynomial in the number of edges of P plus the time to enumerate the adjacent edges at each processing step.

The backbone of Provan's algorithm [79] is the following proposition.

Proposition 2.1(*Provan*): The edges of the polyhedron P adjacent to a vertex v are in one-to-one correspondence with simple cycles of a directed graph G_v (explained later).

The algorithm is described in the following formats which have been extracted from Provan [79]:

(a) Characterize the set of feasible bases of the linear systems representing the vertex v. In this case vertex v which is a basic feasible solution to the linear system (2.2) and (2.3) and represented or partitioned as follows: E_B - basic edges, E_L-edges at the lower bound, and E_U - edges at the upper bound. E_B^* represent the edges whose flows or component x_{uv} are neither at lower nor at upper bounds.

(b) Determine the pivot directions with respect to each basic feasible solution, that is the directions taken from that basic feasible solution in an attempt to bring non basic variables into the basis. If we let (E_B, E_L, E_U) represent a vertex v, and (u, v) an edge in $E_L \cup E_U$, then inserting edge (u, v) into the spanning tree formed by E_B creates a unique cycle C. Thus, the pivot direction corresponds to a polyhedral edge of P in so far as the respective change in each edge of $E_B \setminus E_B^*$ is also in the direction away from its respective bounds.

(c) Characterize the polyhedral edges adjacent to v, by determining in steps (a) and (b) above which of the basis of v admit non-degenerate pivot direction. This is based on the earlier proposition 2.1 mentioned above.

(d) Give an efficient enumeration routine that will list these edges direction over all possible basis representation for v.

(e) Modify step (c) above to list only bounded polyhedral edges.

The description of the polyhedral edges of network polyhedron P adjacent to a vertex v can be facilitated by the following construction. First, we can contract the edges in E_B^* in the original graph G, and let the result be $G' = (V', E')$, with edges in E' associated with E, edges of the original graph in the obvious way. That is to say, if the vertex under consideration is non-degenerate, contraction of E_B^* will

yields one single component (which we call a *pseudo-vertex*). But, if the vertex is degenerate we will get different pseudo-vertices, each with composition of vertices in the original graph attached to E_B^* edges. We then let G_v be the graph with vertex set $V_v = V'$ and edges set be:

$$E_v = \{(u,v) : (u,v) \in E', x_{uv} = l_{uv}\} \cup \{(u,v) : (u,v) \in E', x_{uv} = c_{uv}\} \qquad (2.4)$$

The edges of the contracted graph G_v now describe the allowable direction of change in the value of an edge E' in the description of pivot direction of step (b) above. The edges E_v are more or less associated with non- E_B^* edges.

In order to avoid listing unbounded edges of P, it is necessary to ensure in the listing of cycles of the contracted graph G_v that each cycle of the original graph G has at least one edge that is bounded in the direction defined by the direction of cycle in G_v [79].

2.4 Tasks Necessary for Computer Implementation

Provan [79] provides a high-level description of his algorithm intended to support the theoretical analysis rather than computer implementation. There are several vague steps that are not detailed in the algorithm and which are essential for its computer implementation. We need to specify how to:

1. Contract the original graph into a multi-graph.
2. Convert the multi-graph into a simple one.
3. Convert the contracted E_B^* edges in pseudo-vertices into trees whose vertices point toward a root in the tree.
4. Find the cycles of the contracted graph G_v.
5. Translate simple cycles of the contracted graph into cycles of the multi-graph.
6. Find the adjacent vertices from the corresponding cycles.

7. Design accounting procedure to ensure that vertices are neither omitted nor replicated.

2.5 Implementation

In an attempt to implement any computer algorithm a high level computer language is ultimately needed. In our work Fortran 90 [81] was selected as it supports the recursive procedures which are used in [79].

There are basically two algorithms developed in [79], one is primarily concerned with primal network LP and the second, which uses similar ideas, with its dual. In our work concentration is given to the implementation of the primal network LP algorithm which involved the following steps:

- Code for getting an initial vertex
- Code that transforms network data into a form appropriate for Provan's algorithm
- Code for the contraction of graphs into simple graphs
- Code that finds cycles of simple graphs
- Code that finds cycles of multi-graphs and adjacent vertices
- Code that chooses a new vertex
- Code that performs the accounting procedure

The first stage of our implementation involved coding a routine capable of enumerating simple cycles of simple graphs. This has became a vital subroutine of the main program and is invoked at a later stage of the program.

2.5.1 Data structures

The data structures used are linear adjacency lists of directed graphs, with a pointer for each node to the element of the list at which its edges begin. In the main program some codes are developed that read in the following:

- Number of nodes n
- Out-degree of each node
- Adjacency list for each node
- Lower bounds on each arc (edge)
- Upper bounds on each arc
- Initial flows on each arc

Nodes	Adjacency	LUP Truth Values
1	4	T
	5	T
	6	F
	7	F
2	4	F
	5	F
	6	T
	7	T
3	4	F
	5	F
	6	F
	7	T

Figure 2.1: **Data structure of an initial vertex**

The data for the reversed graph, i.e. the graph in which the direction of each arc in the original graph is reversed, are also stored in a linear adjacency list. The reversed graph is required because, at some stages of the implementation, especially in the construction of E_B^* trees in pseudo-vertices, directed trees are used while at others undirected ones are used. See Figures 2.1, 2.2.

Note that LUP(j) is a binary function, which is true if an edge j is an E_B^* edge and false otherwise. They are read in as follows:

- Out-degree of each node

Nodes	Adjacency	LUP Truth Values
4	1	T
	2	F
	3	F
5	1	T
	2	F
	3	F
6	1	F
	2	T
	3	F
7	1	F
	2	T
	3	T

Figure 2.2: **Data structure of the reversed graph of Figure 2.1**

- Adjacency list for each node
- The correspondence between each edge in the reversed graph and its mirror edge in the original graph, i.e. link(i) = j means that the ith element of the adjacency list for the reverse graph is the jth element of the adjacency list for the original graph

2.5.2 Initial vertex

Any basic feasible solution to a linear program (in this case network LP) will provide an initial vertex. In our case an existing minimum cost network flow code, modified to output the data described in 2.5.1, was used. Figure 2.3 is an example of a transportation problem with 3 sources and 4 sinks that is used for illustration. It was also one of the data sets used to test our program.

The adjacency data structure (used as lists for directed graphs) are used as follows: the nodes of the sources have their corresponding adjacent nodes of the sinks. In the reversed case, the nodes of the sinks have nodes of the sources as their adjacency, the reversed graph is needed in the construction of a contracted graph, which is

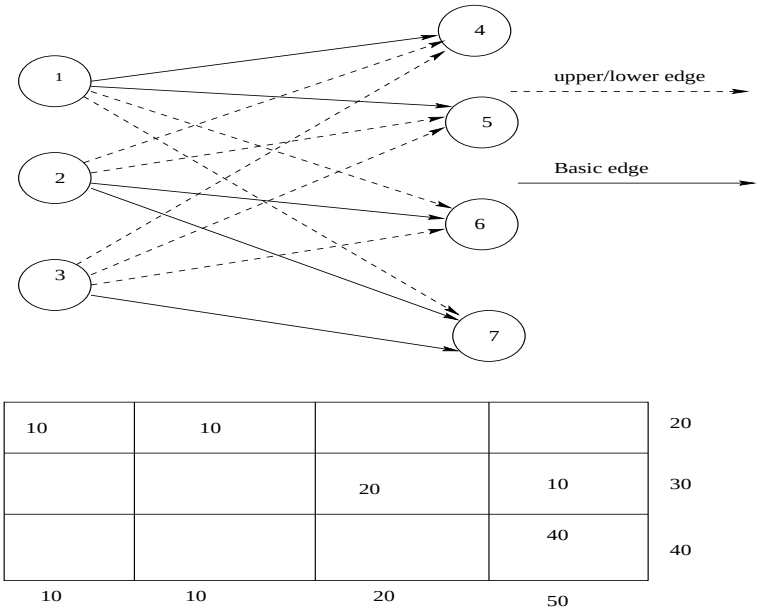
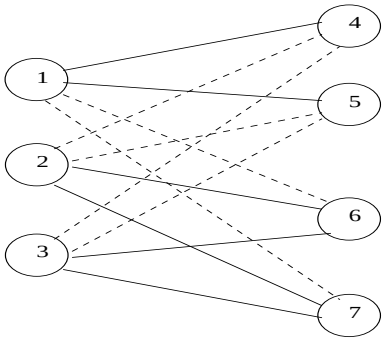


Figure 2.3: **A network LP and its initial vertex**

explained below. See also Figures 2.2, 2.4 and 2.5 for visualizations of the concept explained (note that 'label' is used to indicate the number of pseudo-vertices, and which of the vertices are in the same pseudo-vertex i.e. if vertices have the same 'label' value, then they are in the same pseudo-vertex, else in different ones). Also, note that 'label' is the same function as 'Name' used below.

2.5.3 Contracting graphs into simple graphs

One of the most important steps needing to be efficiently implemented is the contraction of both E_B^* and non-E_B^* edges in order to get the required simple graphs from which cycles and hence adjacent vertices are determined. This has been achieved with the aid of the **Union-Find** algorithm from Aho et al. [1]. This algorithm has good computational and theoretical efficiencies. It has been modified to suit our particular need. Another modification is in the *Union* subroutine, with the aid of the integer function $FIND$ we find vertices say i and j and try to compare which one of them has greater ordering number, this is not presented in the original version. Also the last two steps of the original algorithm from Aho et al. [1] are replaced by $Count(i) \leftarrow 0$ if i is a root. Another reason for the modification



Nodes	Adjacency	LUP Values
1	4	T
	5	T
	6	F
	7	F
2	4	F
	5	F
	6	T
	7	T
3	4	F
	5	F
	6	T
	7	T

Figure 2.4: **Data structure for a vertex**

is because of the data structure used in the original graph is a linked list, while our data structure for the modified Union-Find is a doubly linked list.

2.5.3.1 Modified Union-Find algorithm

(1) INITIALIZATION:

for $i \leftarrow 1$ to n **do**

$Count(i) \leftarrow 1$

$Father(i) \leftarrow 0$

(2) FIND-as integer function:

Make list empty

$v \leftarrow i$

while $Father(v) \neq 0$ **do**

Add v to the list

$v \leftarrow Father(v)$

Vertices in EB* that have T as their LUP truth value		Label
Vertex	Adjacency	
1	4, 5	1
2	6, 7	2
3	6, 7	2
4	1	1
5	1	1
6	2	2
7	2, 3	2

Figure 2.5: **Data structure of contracted degenerate vertex of Figure 2.3**

$Find \leftarrow v$ N.B: v is the root of the tree containing i.

Path compression is as follows:

for each w in the list **do**

$Father(w) \leftarrow v$

(3) UNION:

N.B:- Union set containing i and j

$f_i \leftarrow Find(i)$

$f_j \leftarrow Find(j)$

if $Count(f_i) \leq Count(f_j)$ **then**

$small \leftarrow f_i$

$large \leftarrow f_j$

$Count(large) \leftarrow Count(large) + Count(small)$

$Count(small) \leftarrow 0$

$Father(small) \leftarrow large$

N.B:- If Count(i) $= 0$, then i is the root

How the Union-Find algorithm is implemented can be explained with aid of an

example, see Figures 2.6 and 2.7, as follows: **Find** is used as an integer function, it starts with, say, vertex 1 and finds those vertices adjacent to it that are joined (linked) by E_B^* edges. Thus, from Figure 2.4, it finds vertices 4 and 5 and unions their respective E_B^* edges. These three vertices are now merged into a single pseudo-vertex. This is recorded in the array *Name* so that cycles in the original graph can be recovered from the cycles which we find in the contracted graph. Note that $Name(1) = Name(4) = Name(5)$, since they are all in pseudo-vertex I (Figure 2.7).

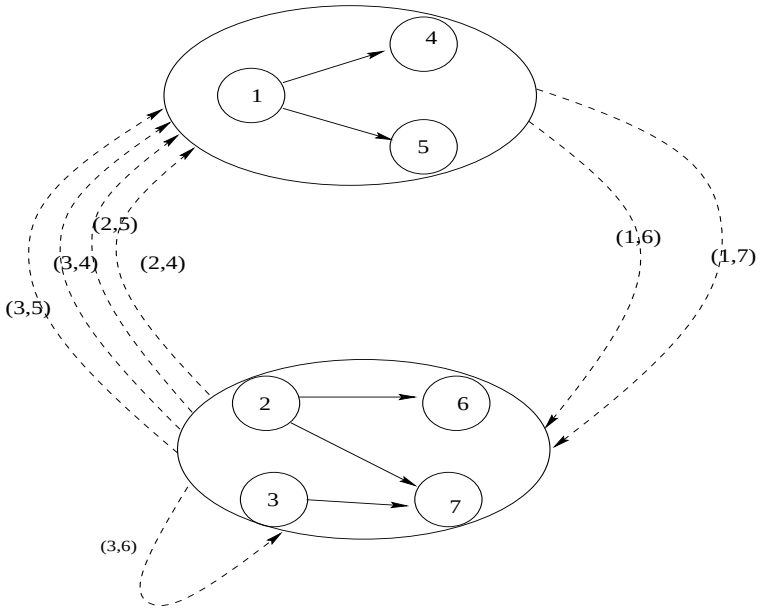


Figure 2.6: **A contracted multi-graph**

The Union-Find algorithm then finds vertex 2 and the vertices adjacent to it that are joined by E_B^* edges. In this case vertices 6, 7 and 3 are found and their corresponding E_B^* edges are merged together into pseudo-vertex II. Here also, Name(2) = Name(6) = Name(7) = Name(3).

In each of the constructed pseudo-vertices, E_B^* edges are transformed from directed into undirected trees, the reversed data structures are used in this case. This is to ensure while tracing the path in each pseudo-vertex for the cycles of the original graph appropriate paths are followed. The root of the tree in this case is vertex

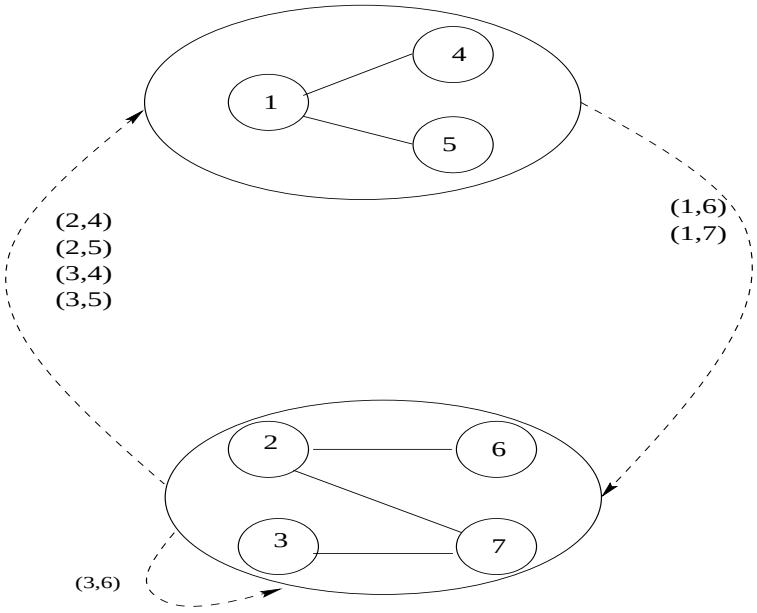


Figure 2.7: **A contracted simple graph**

numbered 2, all other vertices are pointed toward the root. It can be noted that vertices 6 and 7 are at the same level with respect to the position of the root and vertex 3 is at a lower level. See Figure 2.8 for an illustration.

Note that multi-graphs have to be transformed into simple ones. Therefore, non-E_B^* edges are aliased into a single edges as can be seen in Figure 2.7. It has been noted that each of the pseudo-vertex-trees has been transformed from directed to undirected tree, a code has been established that was able to convert the directed contracted E_B^* edges tree structures into undirected one so that the paths passing through cycles involved with those E_B^* edges trees in their respective pseudo-vertices are easily traced. This is done on the contracted graph.

The aliases of the non-E_B^* edges of the contracted multi-graph is aided by a special computed function called 'Dangi(j)'. That is defined as 0 if the edge is an E_B^*, $-j$ if j is in a simple loop, and k if edge j in the original graph is merged into edge k of the contracted simple graph.

Figure 2.7 illustrates a degenerate vertex. On the other hand contraction of a non-

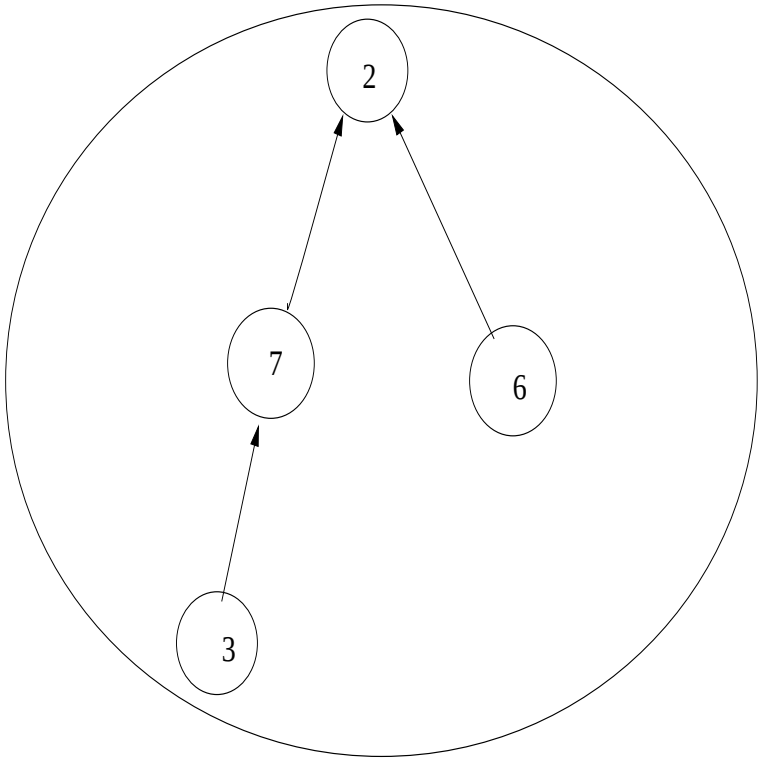


Figure 2.8: **Transformations of E* edges of a pseudo-vertex**

degenerate vertex always yields a single component pseudo-vertex that is a loop identical to end point. These comprise 1-edge cycles and therefore must be counted in the cycles that create edges. After a simple graph is obtained successfully, the next step is to find or enumerate its simple cycles.

2.5.4 Finding cycles of simple graphs

Another vital routine which is able to enumerate efficiently simple cycles of simple graphs has been successfully implemented. This has been achieved with the aid of Read and Tarjan's [80] algorithm for enumerating simple cycles of directed graphs. Implementation of [80] required firstly implementing a Depth First Search (DFS) algorithm from Aho et al. [1] which was used to find the strongly connected components of directed simple graphs.

Read and Tarjan's algorithm [80] was implemented as follows:

(a) The DFS algorithm from [1] was implemented to firstly find the strongly con-

nected components of a directed graph. It uses a Last in-First Out (LIFO) stack; *Push* was used to insert vertices onto the stack and *Pop* was used to get the vertices out of the stack. The vertices are numbered by a search number in accordance with the order they are visited. At each stage of DFS, the stack represents a path from the root to the current vertex. If the current vertex has an edge leading to a vertex that is already on the stack then the vertex pointed to must be a *cycle-start-vertex*. It should be noted that there could be more than one cycle start vertex per strongly connected component. It may not matter which of the starting vertex is found. The edges that join together strongly connected components are deleted as proposed in [80], this was done by using a binary function, which set these edges to False.

The different types of edges encountered in the algorithm are defined below :
(i) Tree edges - are edges leading to new nodes during a search.
(ii) Forward edges - are edges that go from ancestors to proper descendants, but are not tree edges.
(iii) Back edges - are edges that go from descendants to ancestors (possibly from a vertex to itself)
(iv) Cross edges - are edges that go between vertices that are neither ancestors nor descendants of one another.

The visualization of the DFS algorithm implemented for generating strongly connected components of directed graphs can be seen with the aid of Figure 2.9. Strongly connected components are transformed into individual (single) components by deleting edges connecting the vertices of the components (this has been explained above).

(b) There is also another vague step in the *Backtracking* procedure of [80] for enumerating simple cycles, for example a path needs to be constructed that reaches the cycle-start vertex. This path has to be different from the current path. This procedure was not detailed in [80] or in [79]. However, with the assistance of a Breadth First Search (BFS) algorithm from Cormen et al. [14, page 471-472] this

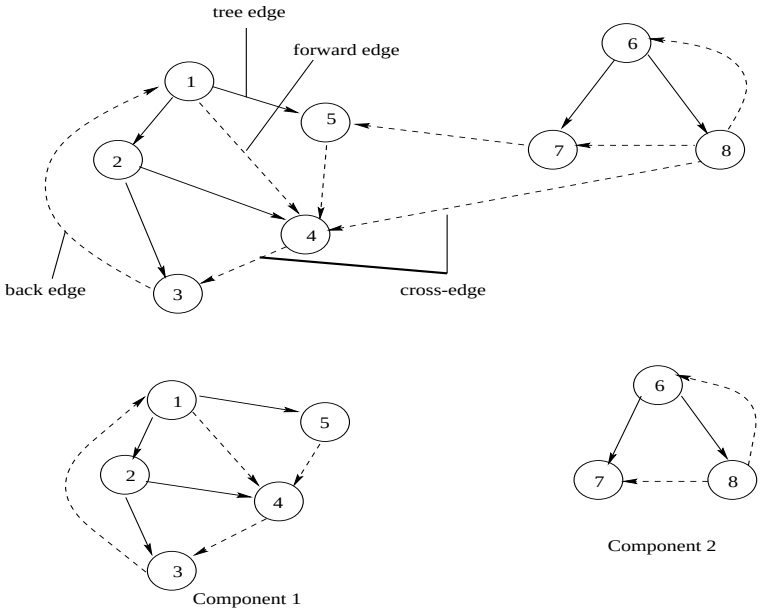


Figure 2.9: **DFS procedure for strongly connected components of a directed graph**

has been successfully accomplished. BFS uses a First in First out (FIFO) queue; in this case also the *Push* and *Pop* routines are invoked. Figure 2.10 is an illustration of how BFS algorithm was implemented.

2.5.5 Finding cycles of multi-graphs and adjacent vertices

The simple cycles of the contracted graph (obtained in term of the pseudo-vertices) have to be translated into cycles of the original graph so that flow adjustment can be performed round the cycle in order to generate a vertex of the network polytope. Each cycle edge in the contracted graph is aliased to several non-E_B^* edges of the network. Consequently each cycle in the contracted graph has to be expanded using all combinations of the aliased edges. This is achieved using a depth-first search method which results in *pseudo-cycles*, i.e. structures of the form $(i_1, j_1), (i_2, j_2), ..., (i_p, j_p)$ where i_k, j_k are associated with successive pseudo-vertices in a cycle in the contracted graph and j_k, i_{k+1} are associated with the same pseudo-vertex. A cycle of the original graph is completed by tracing the path between each

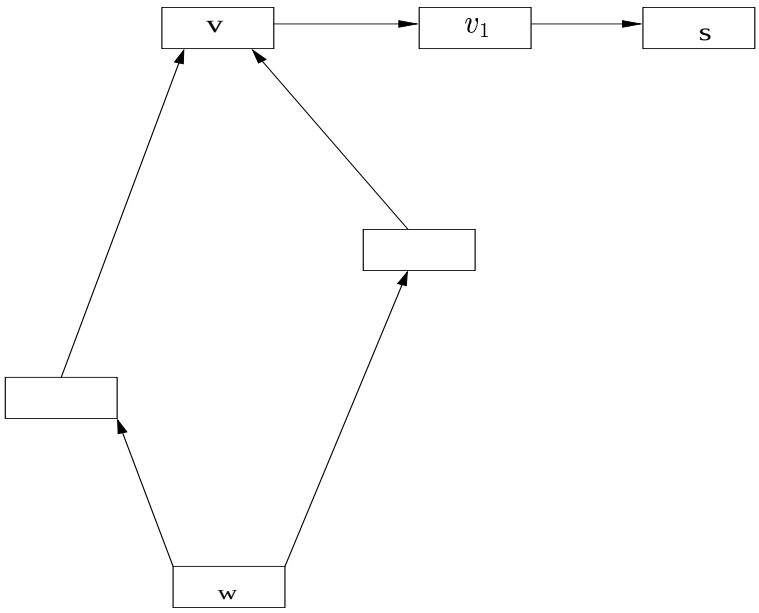


Figure 2.10: **BFS procedure for finding an alternative path**

pair j_k, i_{k+1} using the appropriate pseudo-vertex tree.

Figure 2.11 illustrates the process for the contracted simple graph of Figure 2.7. The cycle **i-ii** can be translated into 9 cycles of the original graph. As an example, cycle $1-6-2-4$ is found as follows, in the simple cycle $i-ii$, edge $(1,6)$ is aliased to edge $i-ii$ and edge $(2,4)$ is aliased to edge $ii-i$. We now need to trace the path from 6 to 2 inside pseudo-vertex ii, which is simply $(6,2)$, and from 4 to 1 inside pseudo-vertex i, which is simply $(4,1)$. The cycle is now complete and we now look for the cycle which includes $(1,6)$, $(2,5)$. The edge $(3,6)$, which is a simple loop on pseudo-vertex ii, is also expanded into a cycle of the original graph by tracing a path from 3 to 6 on the pseudo-vertex tree.

2.5.6 Finding adjacent vertices

A code has been developed which is able to determine adjacent vertices of the current vertex. This can be explained using the simple cycle $1-6-2-4-1$ in Figure 2.12. The first edge in the cycle must be a non-E_B^* edge and thus the flow in this edge must be at either the lower or upper bound. Here the edge $(1,6)$

Initially, contracted cycle i-ii is enumerated which is then translated into simple cycles of the original graph as follows:

(1) 1-6-2-4

(2) 1-6-2-5

(3) 1-6-2-7-3-4

(4) 1-6-2-7-3-5

(5) 1-7-2-4

(6) 1-7-2-5

(7) 1-7-3-4

(8) 1-7-3-5

(9) 3-6-2-7 (simple loop)

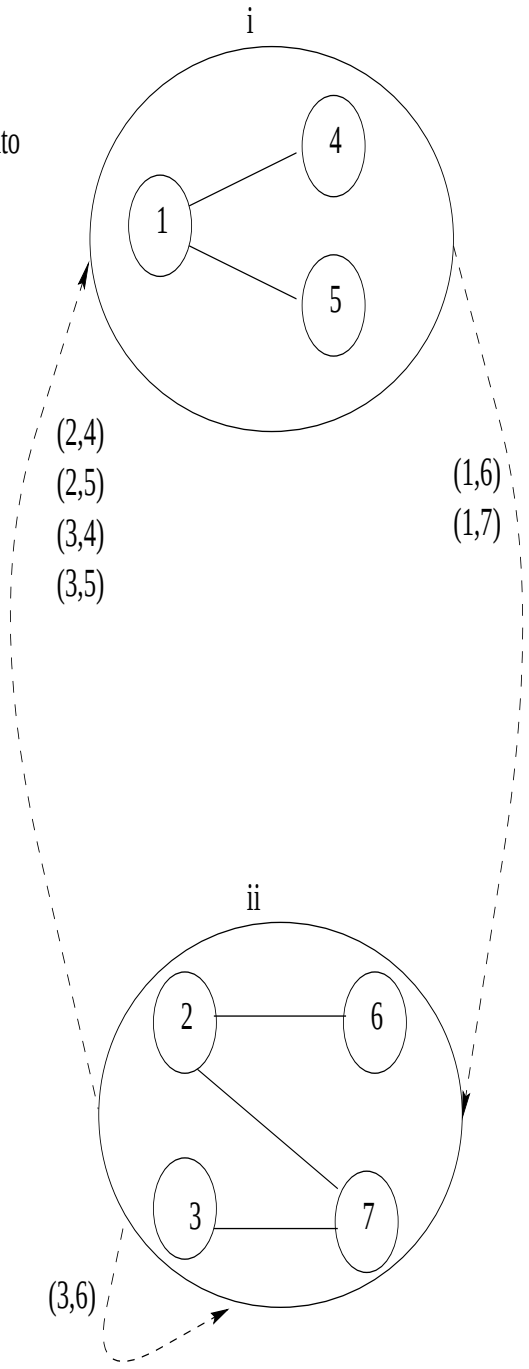


Figure 2.11: **Simple cycles are obtained from the contracted graph**

is at its lower bound and so the flow must increase in this edge, decrease in the successor edge $(6, 2)$ and so on round the cycle As none of the edges have finite upper bounds, the flow change is determined by the edges on which flow decreases, so $Q = min\{10, 20\} = 10$. After substituting the value of Q into the cycle edges we get the new vertex shown in Figure 2.12.

2.5.7 Finding simple loops

Another routine is coded that was able to enumerate simple loops and the subsequent adjacent vertices. Simple loops are found when a non-E_B^* edge of a degenerate vertex is associated with a single pseudo-vertex, see, for example, Figure 2.12. Also, it is possible to get a (single component-pseudo-vertex) loop when the E_B^* edges of a non-degenerate vertex are contracted, as shown in Figure 2.13. In this case loops are formed by considering each non-E_B^* edge in turn. Flow adjustment, leading to the discovery of adjacent vertices, is performed in a similar manner to that for cycles.

2.6 Accounting Procedure (Hashing)

In most vertex enumeration algorithms, an accounting procedure has to be incorporated in order to ensure that vertices are neither omitted nor repeated in the enumeration procedure. In our case *Hashing* was selected because it has been shown to be more effective in the context of Dyer's algorithm than the more elegant AVL tree method [77]. Hashing has also been used to manage the accounting procedure within the Chen et al. [10] algorithm. In this method, a hash function is used to map a vertex into a cell in a hash table. If a generated vertex hashes to an empty cell in the table, it is newly discovered. If it hashes to an occupied cell, it is said to collide with another vertex. The two vertices are then compared on an arc by arc basis to determine whether the generated vertex is new or a copy of the existing vertex. Several distinct vertices may collide; the number of collisions clearly has an effect on performance and can be influenced by the choice of hash function. Many methods of managing collisions are suggested in Knuth [58] e.g.

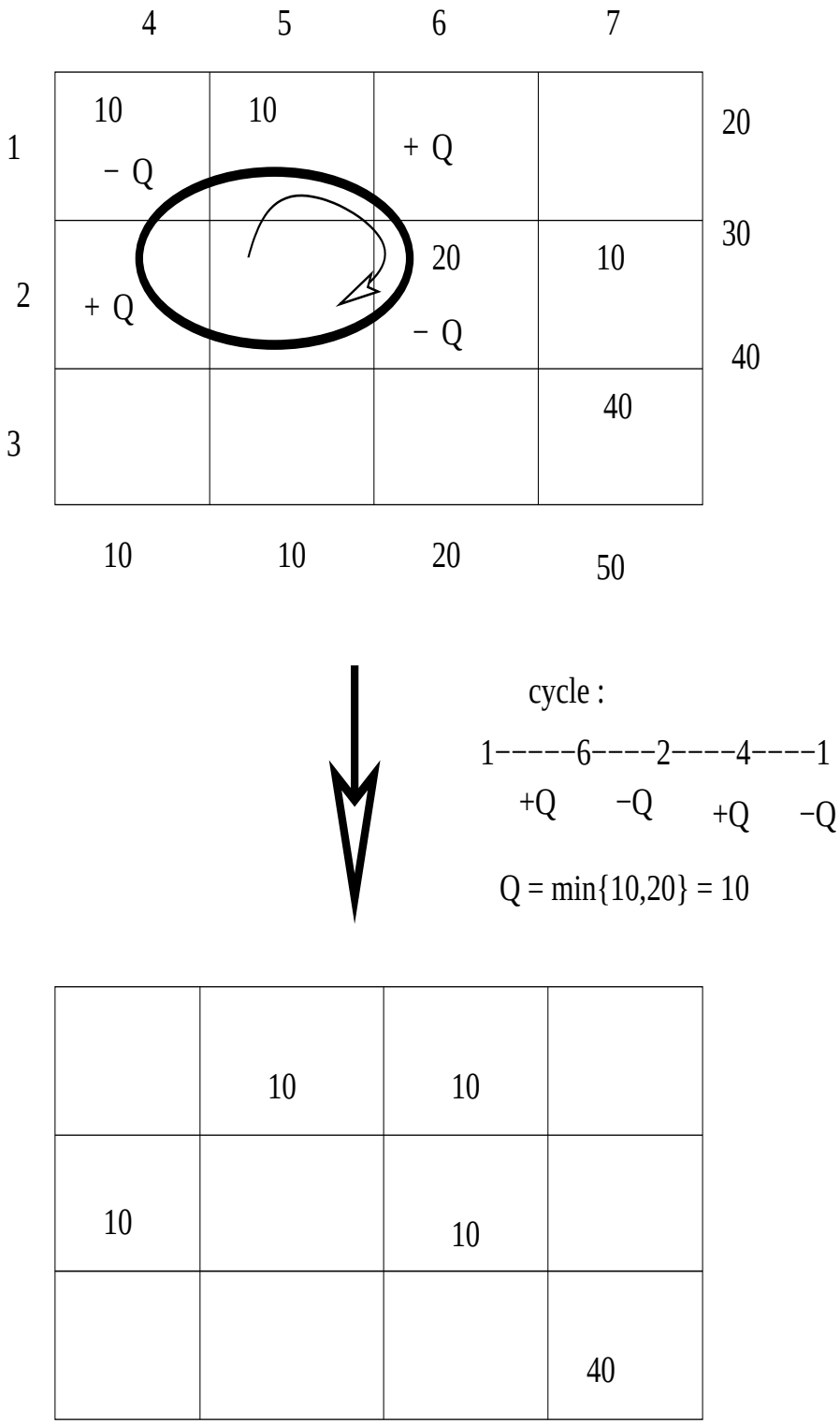


Figure 2.12: **Finding a new adjacent vertex using a simple cycle**

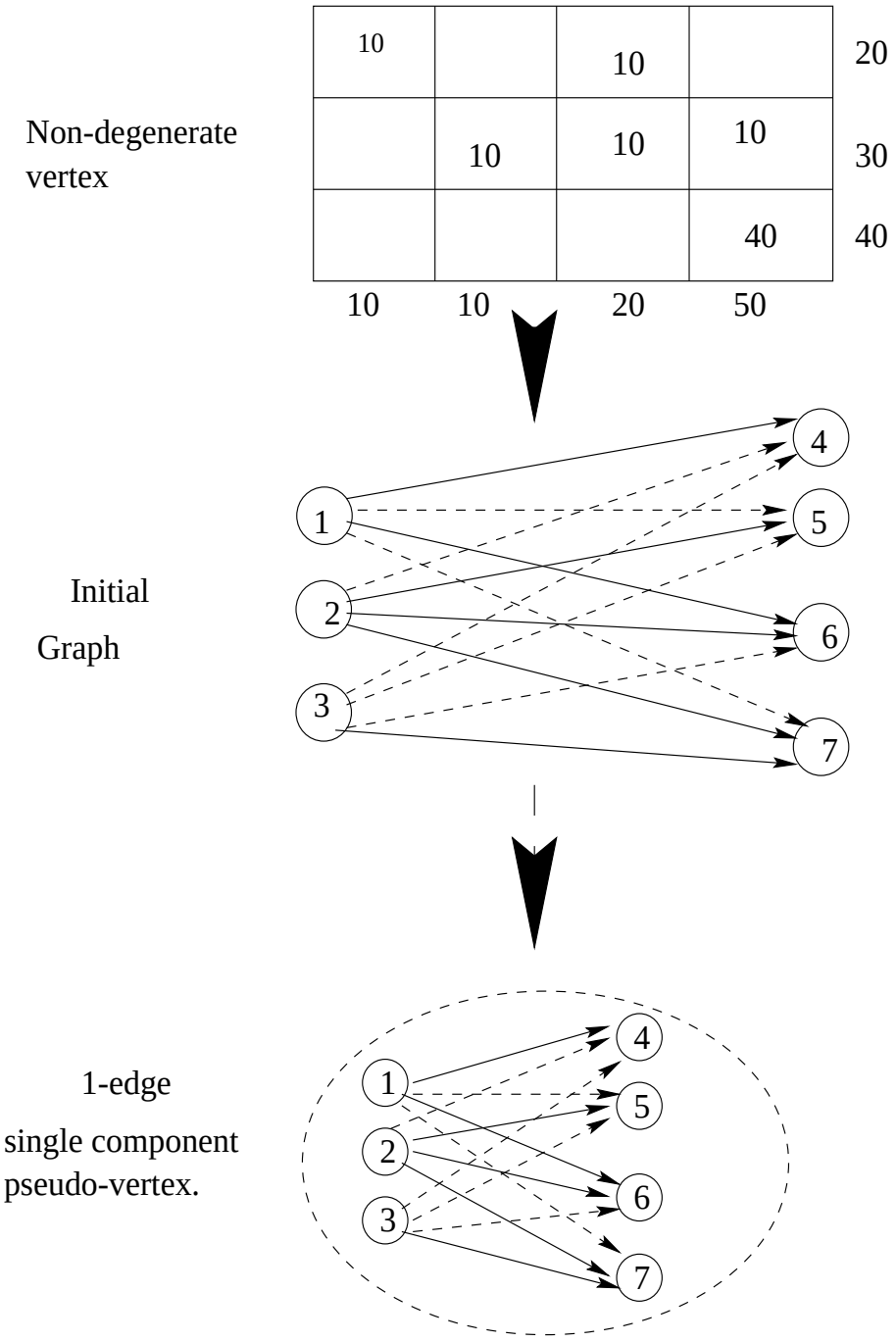


Figure 2.13: **Contraction of a non-degenerate vertex**

Chaining, Open Addressing etc. We use a hash function of modulo type:

$$h(r) = r \ mod p + 1$$

where r is an integer encoding of the vertex v, h is its hash value and p is a prime number. We choose $p = 15001$ and manage collisions by chaining method.

The hash table is implemented in the linear arrays, $occupy(1:p)$, $vlist(1:mvert)$ with the following interpretation:

$occupy(k) = 0 \Rightarrow$ there is no vertex which hashes to k

$occupy(k) = v > 0 \Rightarrow$ vertex v hashes to k and is the first encountered vertex which does so.

The linked list $vlist$ enables collisions to be traced as:

$v, vlist(v), vlist(vlist(v))$, ..., until $vlist(---) = 0$.

The vertices themselves are not stored in memory but are held on a random access file.

2.6.1 Encoding a vertex

A vertex of a network polytope is described by $flow(j), j = 1 : arcs$. Each arc can be in one of three states; we therefore encode the vertex by using 2 bits for each arc as follows:

$00 \Rightarrow flow(j) = low(j)$

$01 \Rightarrow low(j) < flow(j) < up(j)$

$11 \Rightarrow flow(j) = up(j)$

The encoded arc j is stored in bits $j-1$, j of the integer SET. If SET is an 8 byte integer, the largest available on the platform used, we can store 32 arcs in SET. For larger problems we can either encode only the first 32 arcs, which is likely to increase the number of collisions, or use multiple words for encoding.

2.7 Computational Results

Empirical evaluation of any algorithm depends on its computational results. Since test data sets are not readily available, some data sets were developed and used for benchmarking and testing the practical efficiency of Provan's algorithm [79]. These were selected from both transportation and assignment problems. It is noted that the latter are highly degenerate polyhedra, an $n \times n$ assignment problem having a degree of degeneracy *n-1*. Also note has been made that there is little or nothing in the literature with which to compare our results.

In order to ensure the accuracy of our program, most data sets have been run with several different initial vertices. This is to ensure conformity of the execution of the program and to confirm the exact number of vertices per data set. For example, the transportation problem of four sources and five sinks has been classified as; ex2a, ex2b, ex2c and ex2d. Each is a 4×5 transportation problem with the same total quantities of demand and supply; same lower and upper bounds, but different initial vertices. We denote these data sets as ex2 which represents the average of the four results.

Results are given for the execution time of the program in seconds obtained from an IBM Celeron 400MHZ machine with 128K memory running under Linux. The algorithm was implemented in Fortran 90 using the Portland Group pgf90 compiler. Table 2.1 gives the results for a set of transportation problems.

Problem	Sources	Sinks	Vertices	Degenerate Vertices	Time (secs)	Collisions	Unused Cells(%)
ex1	3	4	31	11	0.03	0	99
ex2	4	5	886	48	1.08	1	99
ex3	4	6	3734	825	6.57	37	97
ex5	5	6	23346	7308	60.41	1702	85

Table 2.1: **Computational results for transportation problems**

Solving the problems of degeneracy is of immense importance. Initially it was one of our research goals. However, this has proven to be a difficult task for the simple fact that for general polyhedra the number of vertices grow exponentially. As such concentration is given to specialized polytopes (like network polyhedra by Provan [79], simple polytope by Dyer [24] and polyhedra associated with *LI(2)* systems as we discussed in Chapter 5 of this thesis). In the implementation of Provan's work [79] we attempt to compare the type of initial vertex used and the number of degenerate vertices discovered plus the number of collisions found, also to know exactly the percentage of unused cells of the hash table used from the above transportation problems data sets. This is clearly shown in Table 2.1.

Another benchmark set of data that is used to test our program are assignment problems, for which all vertices are degenerate. The accuracy of the program is easily determined by the fact that for any $n \times n$ assignment problem there are always $n!$ vertices. Thus, in this case we considered 3×3 to 7×7 assignment problems. Computational results are given in Table 2.2.

Problem	n	Vertices	Time (secs)	Collisions	Unused Cells(%)
assn2	2	2	0.01	0	99
assn3	3	6	0.01	0	99
assn4	4	24	0.06	0	99
assn5	5	120	1.03	0	99
assn6	6	720	31.54	4	99
assn7	7	5040	1624.90	2869	98

Table 2.2: **Computational results for assignment problems**

Table 2.3 gives results for several assignment problems with p prohibited assignments.

The increased number of collisions in problems ex5 and assn7, for example, is due to them having more than 32 arcs and to the current implementation which uses

Problem	p	Vertices	Time(secs)	Collisions
assn6x1	1	600	19.84	3
assn6x2	2	504	13.59	3
assn7x3	3	3216	601.68	1619
assn7x5	5	2428	314.45	569
assn8x12	12	7870	3972.31	5645

Table 2.3: **Computational results for assignment problems with prohibited assignments**

only single-word hashing.

It is important to note that some larger data sets still give problems while trying to test them with our program. For example there is a 6×10 transportation problem which encountered problem with our code, i.e. our program is unable to execute. This is due to the fact that our computer cannot store and process the vast number of cycles encountered. Similar problems were encountered with higher dimensional assignment problems, e.g. 8×8.

2.8 Short-Comings of Provan's Algorithm

One of the short-comings observed with Provan's [79] algorithm is the procession of data sets that involved a large number of cycles. It is observed that in the 8×8 assignment problem whenever a new vertex is discovered, 16064 cycles correspond to edges adjacent to the vertex. As the 8×8 assignment problem has 40320 vertices, approximately 648 million cycles need to be processed to obtain all the vertices of this data set. The execution times in Table 2.2 are roughly in line with the theoretic quadratic performance of the algorithm. Based on this the execution time for the 8×8 assignment is likely to be of the order of $8^2 * 1624.90$ which is approximately 29 hours. This is clearly an unattractive proposition.

2.9 Conclusion

The vertex enumeration algorithm for polyhedra associated with network LP implemented in this chapter [79] is unusual for listing algorithms in the sense that it does not run in time linear in the number of objects (vertices) listed, therefore, cannot be referred to as *time-per-object* complexity measure as is done customarily. The running time of the algorithm is quadratic.

Provan's algorithm represents an important theoretical advance in dealing with degenerate polyhedra. Our experiments, however, show that even relatively low dimension degenerate polyhedra remain computationally challenging. It may be noted that for the explicitly perturbed 3×3 assignment problem, an implementation of Dyer's algorithm generated excess of $200,000$ basic feasible solutions. A version of Dyer's algorithm with an implicit degeneracy-handling mechanism [23] generated 18 basic feasible solution, 3 copies of each vertex.

Profiling information returned by pgf90 suggests that approximately 60% of the execution time is spent in checking potential new vertices. This compares with Dyer and Proll's [29] figure of 90% for the accounting procedure in their algorithm. In Provan's algorithm, each vertex is generated many times. There does not appear to be any way in which we can tell that a cycle will lead to an already discovered vertex without performing flow adjustment around the cycle and hashing. Generating the same vertex more than once will necessarily lead to collisions and hence to an expensive arc by arc check. In the Dyer-Proll algorithm [29] regeneration of basic feasible solution was avoided by the use of the $'gamma - set'$ which listed the variables which potentially could be pivoted into the basis to give a new basic feasible solution. Effectively, this relies on the fact that a basic feasible solution can be uniquely described by the labels of the basic variables. This is true for the general problem which Dyer and Proll [29] tackled but not for the problem tackled by Provan's algorithm because of the presence of non-trivial bounds on the variables.

Chapter 3

The *LI(2)* Problem

3.1 Introduction

In this chapter we survey some algorithms for Linear Programming problems with at most two variables per constraint. The problem is that of finding some or all solutions to the system of the form:

$$Ax \leq b \tag{3.1}$$

where A has at most two non-zeros per row and is denoted by *LI(2)*. The set of solutions to (3.1) forms a convex polyhedron. It is well known that the LP problem where one wants to maximize a linear function subject to linear inequality constraints is polynomial time (Turing) equivalent to *LI(2)* [37]. The complexity of the LP problem was one of the foremost open problems in theoretical computer science until Kachiyan [54] announced a polynomial time algorithm for the *LI* problem. Since then researchers have become interested in developing faster algorithms for this type of problem. *LI(2)* is of significant importance and has practical applications in, e.g. mechanical verification [74]. Let us review some of the important results.

3.2 Shostak's Algorithm for *LI(2)*

Shostak [90] has suggested that a linear program with at most two variables per inequality can be represented as an undirected graph G in which each vertex represents a variable. The inequalities involving two variables, say, x_i and x_j are represented as edges between vertices v_i and v_j; there may be multiple edges between two vertices. The graph has an additional vertex, say v_o, that represents inequalities involving one variable; these are edges connecting the corresponding node to v_o. The number of variables is normally denoted by n and the number of inequalities by m. Without loss of generality (w.l.o.g.) it is assumed that $m \geq n$. G therefore, consists of $n+1$ vertices and m edges. Shostak [90] proved that feasibility can be tested by following paths and cycles in this graph, and thus laid the foundation for all subsequently considered algorithms for this particular type of problem. The main feature of Shostak's algorithm is determining upper and lower bounds for each variable by following paths and cycles in the graph. Let us present some related definitions and then later summarise Shostak's result [90].

Definition 1: Let S be a set of *LI(2)* inequalities. A path through the associated graph $G(S)$ is called *admissible* if, for $1 \leq j \leq r-1$, the inequality $a_j v_j + b_{j+1} v_{j+1} \leq c_j$ representing edge e_j has a_j and b_{j+1} with opposite signs.

Definition 2: A path is a *loop* if its first and last vertices are identical.

Definition 3: The *closure* of $G(S)$ is a graph obtained by adding a new edge to $G(S)$ representing the residue inequality for each simple admissible loop of $G(S)$.

Definition 4: The *residue*, r_P of an admissible path P is define as the triple $< a_P, b_P, c_P >$ is given as:

$$< a_P, b_P, c_P > \;=\; < a_1, b_1, c_1 > * < a_2, b_2, c_2 > *.....* < a_n, b_n, c_n >$$

where, $< a, b, c > * < a', b', c' > \;=\; < kaa', -kbb', k(ca' - c'b) >$ and $k = \frac{a'}{|a'|}$. The *residue inequality* of P is define as $a_P x + b_P y \leq c_P$, where x and y are the first

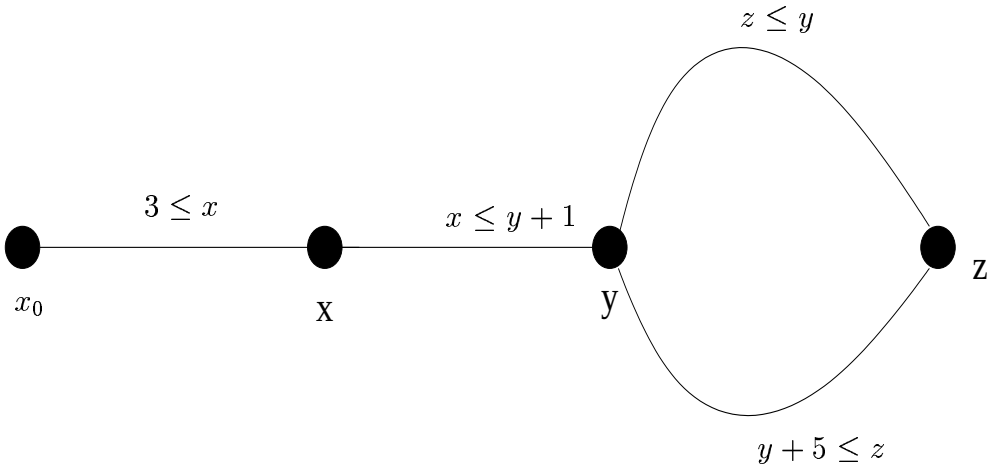


Figure 3.1: **A graph G with simple infeasible loop**

and last vertices of P respectively. If $a_P + b_P = 0$ and $c_P < 0$, the residue inequality of P is called *infeasible loop.*

Theorem 1 ($Shostak$): Let $G'(S)$ be a closure of $G(S)$. Then S is satisfiable if and only if $G'(S)$ has no simple infeasible loop.

Shostak's algorithm essentially has two steps:

- check for infeasible loops in $G(S)$ and, if none is found,
- check for infeasible loops in $G'(S)$

From Figure 3.1, $S = \{3 \leq x, x \leq y + 1, y + 5 \leq z, z \leq y\}$ and $G(S)$ has a simple loop for which the residue inequality, i.e. the inequality implied by the inequalities in the loop, is $0 \leq -5$. Hence, the loop is infeasible and S is unsatisfiable.

The algorithm of Shostak [90] is exponential in the number of variables (worst case), because the number of cycles in a graph on n vertices can grow exponentially in n. Aspvall and Shiloach [3] later used it as a basis with which they develop a refined polynomial time algorithm for the LP with at most two variables per inequality. We now present a survey on Aspvall and Shiloach's algorithm [3].

3.3 Aspvall and Shiloach's Algorithm

It is a well established fact that the solution space of a system of linear inequalities with at most two non-zero per row (or any LI) forms a convex polyhedron, see Jarvis et al. [63]. Aspvall and Shiloach [3] present an algorithm for *LI(2)* whose time complexity is $O(mn^3I)$ on a random access machine, where I is the size of the binary encoding of the input, m is the number of constraints and n is the number of variables. Let us present a summary of Aspvall and Shiloach's result which is also a refinement of Nelson's algorithm [73].

Theorem 2 (*Aspvall* and *Shiloach*): The time complexity of the *LI(2)* algorithm is $O(mn^3|I|)$ on a random access machine with uniform cost criterion.

Theorem 3 (*Aspvall* and *Shiloach*): The time complexity of the *LI(2)* algorithm is polynomial in the size of the input on a Turing machine.

The main idea of Aspvall and Shiloach is constructing a graph $G(S)$ for a set of *LI(2)* whose vertices $x \in [x\min^{(k)}, x\max^{(k)}]$ (where k is the maximum length of any path from x_0 to x) can be complemented to solutions, until all variables are marked and a feasible vector is returned. They construct in polynomial time a modified extension S^* from $G(S)$ without examining presumably all admissible simple loops, using a binary search technique. The algorithm will guess a value of x and push it through $G(S)$ in a breadth first manner. This can then give new, but not necessarily true, upper and lower bounds on the variables. By analysing the outcome of each guess, it is possible to chop the interval for either $x\min$ or $x\max$ by at least half. Let $(a_A + b_A)x \leq c_A$ be the extension and

$$x\min = \max\{\frac{c_A}{b_A} : A \text{ is an admissible path from } x_0 \text{ to } x \text{ in } G(S) \text{ and } b_A < 0\}$$

$$x\max = \min\{\frac{c_A}{b_A} : A \text{ is an admissible path from } x_0 \text{ to } x \text{ in } G(S) \text{ and } b_A > 0\}$$

The graph $G(S)$ is said to be closed for S if $[x\min, x\max] = \mathcal{X}(S)$, where $\mathcal{X}(S)$ is defined to be projection of the solution polyhedron on the x-axis for a given system

S. Aspvall and Shiloach's results [3], can be summarised in the following theorem.

Theorem 3(b) (*Aspvall* and *Shiloach*): Let $x\min^{(k)}$ and $x\max^{(k)}$ be defined with respect to $G(S^*)$. If S is satisfiable, then $[x\min^{(k)}, x\max^{(k)}] = \mathcal{X}(S)$ for each variable x; otherwise, there exists a variable x such that $[x\min^{(k)}, x\max^{(k)}] = \mathcal{X}(S) = \emptyset$.

Below is the algorithm of Aspvall and Shiloach for constructing a feasible solution to *LI(2)* systems.

Algorithm (Projector) (*Aspvall* and *Shiloach*):
Step 1. [*Send guess from* x_0] Let $i \leftarrow 1$. Transfer the value $g = 0$ over all edges incident to x_0. For each vertex $x \neq x_0$, record the most restrictive lower and upper bound on x.

Step 2. [*Termination*] If $i < n$, set $i \leftarrow i + 1$ and go to Step 3. Otherwise, the algorithm terminates and returns for each $x \neq x_0$ the current lower and upper bound on x as the result.

Step 3. [*Stage i.*] For each vertex $x \neq x_0$ do the following: (a) If the currently most restrictive lower bound on x was recorded during stage $i - 1$, send it over all its positive edges. (b) If the currently most restrictive upper bound on x was recorded during stage $i - 1$, send it over all its negative edges. (c) Record new, and more restrictive, bounds on x. Go to Step 2. □

We can understand more about Aspvall and Shiloach's algorithm with the aid of examples in Figures 3.2 and 3.3. Figure 3.2 is extracted from [3] and used for illustration of Aspvall and Shiloach's algorithm. Note that applying Projector yields a simple infeasible loop. For $0 \leq w \leq 1$, assign restricted lower bound $w = 0$ we have in step (3) $0 \leq x \leq -1$ which is contradictory. Thus, $G(S)$ in Figure 3.2 has an infeasible simple loop, so S is unsatisfiable.

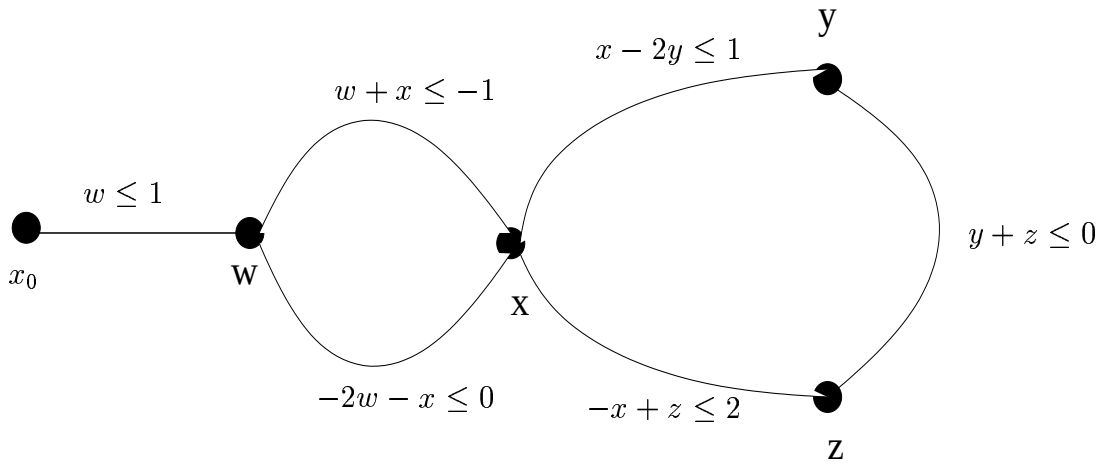


Figure 3.2: **The graph** $G(S)$

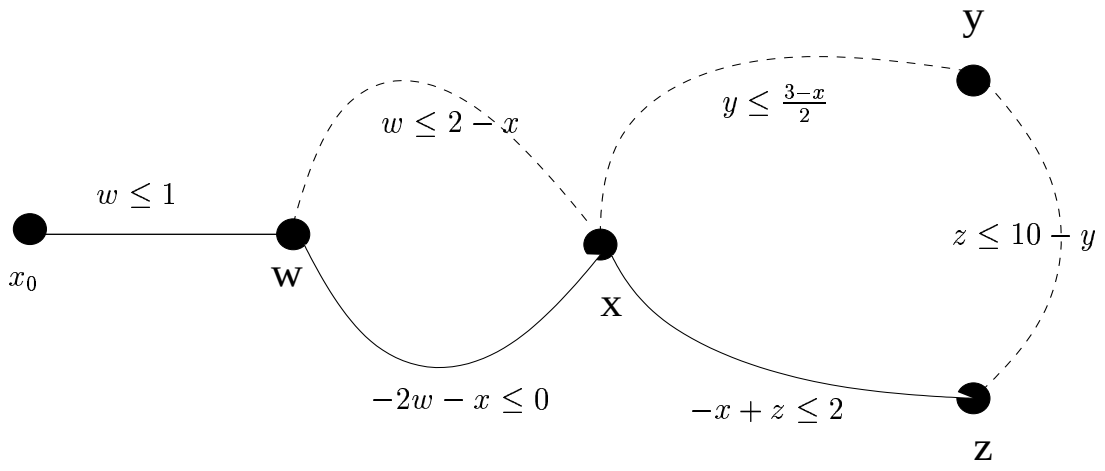


Figure 3.3: **The graph** $G(S^l)$

Let us examine another set of problem, $S^l = \{w \leq 1,\ w+x \leq 2,\ -2w-x \leq 0,\ x+2y \leq 3,\ -x+z \leq 2,\ y+z \leq 10\}$ (see figure 3.3), where $0 \leq x,\ y,\ z,\ w \leq \infty$ and by applying the algorithm we have, $w = 0$, $0 \leq x \leq 2$ or $x = 2$, $2+2y \leq 3$ or $y = \frac{1}{2}$ and $-2+z \leq 2$ or $z = 4$. So one of the feasible solutions is $(x, y, z, w) = (2, \frac{1}{2}, 4, 0)$ which is degenerate.

3.4 Hochbaum and Naor's Algorithm

Hochbaum and Naor [42] presented a strongly polynomial ($O(mn^2 logm)$) algorithm for the feasibility of *LI(2)* systems, based on the Fourier-Motzkin elimination method (*F-M*) [84]. It should be noted that the dual of the network LP structure presented in chapter 2 is an LP with at most two nonzeros per row. That is the dual of (2.1), (2.2) and (2.3) is given as follows:

$$Maximize \quad \sum_{u \in V} b_i y_i \tag{3.2}$$

subject to:

$$y_v - y_u = a_{uv} \tag{3.3}$$

$$(u, v) \in E, \lambda_{uv} = 0, \gamma_{uv} = 0 \tag{3.4}$$

Note: we assume that the associated bound l_{uv} or c_{uv} is not necessarily finite. Although, dual network linear programs are less familiar than primal ones they do come up in several contexts, most notably in description of PERT optimization problems and also dual objects such as min cuts, vertex covers, distances and vertex potential or shadow prices.

Let us algebraically examine *F-M* independently and see how it becomes the backbone of Hochbaum and Naor's [42] algorithm.

3.4.1 General overview of the *F-M* method

Let A be an $m \times n$ matrix and b be an n vector, then we can test whether the system:

$$Ax \leq b \tag{3.5}$$

has a solution, if it does we attempt to find at least one. In *F-M* we multiply each inequality by a positive scalar so that all entries in the first column of A are zero or ± 1 and solve (possibly after a reordering of the inequalities) the following:

$$\xi_1 + a_i x^{'} \leq \beta_i (i = 1, 2, ..., m^{'}) \tag{3.6}$$

$$-\xi_1 + a_i x^{'} \leq \beta_i (i = m^{'} + 1, ..., m^{''}) \tag{3.7}$$

$$a_i x^{'} \leq \beta_i (i = m^{''} + 1, ..., m) \tag{3.8}$$

Here, $x = (\xi_1, \xi_2, ..., \xi_n)^T$ and $x^{'} = (\xi_2, ..., \xi_n)^T$ and $a_1, a_2, ..., a_m$ are rows of A with the first entry deleted. Since (3.6) and (3.7) are equivalent to the following:

$$\max_{m^{'}+1 \leq j \leq m^{''}} (a_j x^{'} - \beta_j) \leq \xi_1 \leq \min_{1 \leq i \leq m^{'}} (\beta_i - a_i x^{'}) \tag{3.9}$$

the unknown ξ_1 in (3.6) and (3.7) can be eliminated giving:

$$a_j x^{'} - \beta_j \leq \beta_i - a_i x^{'} (i = 1, 2, ..., m^{'}; j = m^{'} + 1, ..., m^{''}) \tag{3.10}$$

$$a_i x^{'} \leq \beta_i (i = m^{''} + 1, ..., m) \tag{3.11}$$

The system (3.10) and (3.11) has $m^{'}(m^{''} - m^{'}) + m - m^{''}$ constraints and *n-1* unknowns. Any solution of (3.10) and (3.11), $x^{'}$, could be extended to a solution $(\xi_1, x^{'})$ of (3.6), (3.7) and (3.8) by choosing ξ_1 such that (3.9) is satisfied.

It is possible to eliminate the first *n-1* components of x by repeating the above procedure and end up with an equivalent problem in one unknown, which is trivial. A solution for the original system can be derived from any solution of the final system.

Geometrically, *F-M* methods consist of successive projections: e.g. (3.10) and (3.11) are projections of (3.6), (3.7) and (3.8) along the ξ_1-axis. Ziegler [101] was able to show how the *F-M* method is applicable to higher dimensional polytopes and has attempted to show their projections. Let us review some basic concepts here.

Basic objects for any discussion of geometry are points, lines, planes and so forth, which are *affine subspaces*, also called *flats*. Among them, the *vector subspaces* of $\mathbb{R}^n$ (which contain the origin $0 \in \mathbb{R}^n$) are referred to as *linear subspaces*. Thus,

non-empty affine subspaces are the translation of linear subspaces. The *dimension* of an affine subspace is the dimension of the corresponding linear vector space. Affine subspaces of dimension 0,1,2 and *n-1* in $\mathbb{R}^n$ are called *points*, *lines*, *planes* and *hyperplanes* respectively.

We can take for granted that affine subspaces can be described by affine of all affine combinations of a finite set of points:

$$F = \{x \in \mathbb{R}^n : x = \lambda_0 x_0 + ... + \lambda_n x_n, \lambda_i \in \mathbb{R}, \sum_{i=1}^{n} \lambda_i = 1\} \tag{3.12}$$

Thus, for any $K \subseteq \mathbb{R}^n$ the *smallest* convex set containing K is called the *convex hull* of K and can be constructed as the intersection of all convex sets that contain K:

$$conv(K) = \bigcap \{K' \subseteq \mathbb{R}^n : K \subseteq K', K' \ \ Convex\} \tag{3.13}$$

which is a convex set.

A V-polytope is the convex hull of a finite set of points in some $\mathbb{R}^n$. An H-polyhedron is an intersection of finitely many closed halfspaces in some $\mathbb{R}^n$. An H-polytope is an H-polyhedron that is bounded. So in general a *polytope* is a set of points $P \subseteq \mathbb{R}^n$ which can be presented either as a V-polytope or H-polytope. The *dimension* of a polytope is the dimension of its affine hull.

F-M is a *projection* down to one dimension at a time. This can be discussed only in the case of projections along coordinate axes and the general case can be reduced to this by an $affine\ coordinate\ transformation$.

We present a geometrical sketch for the case of affine polyhedra, the nice thing about this procedure is that its idea and most of its complications can be illustrated in dimension 2 as can be seen in Figure 3.4.

We can start the projection with an H-polyhedron $P = P(A, z) \subseteq \mathbb{R}^n$ and assume that we want to project $\{x \in \mathbb{R}^n : x_k = 0\} \equiv \mathbb{R}^{n-1}$ along the x_k-axis. The

projection of $P \subseteq \mathbb{R}^n$ can be defined in great generality; we will only use the cases of coordinate directions, where we can use the notation for the projection of P in the direction of e_k:

$$proj_k(P) = \{x \in \mathbb{R}^n : x_k = 0, \exists y \in \mathbb{R} : (x + ye_k) \in P\} \tag{3.14}$$

A closely related set is the elimination set that can be represented as follows:

$$elim_k(P) = \{x \in \mathbb{R}^n : \exists y \in \mathbb{R} : (x + ye_k) \in P\} \tag{3.15}$$

An alternative to (3.15) can be given below:

$$elim_k(P) = \{x - te_k : x \in P, t \in \mathbb{R}\} \tag{3.16}$$

Thus, it can be stated that $elim_k(P)$ is the set of all points in $\mathbb{R}^n$ which project to $proj_k(P)$. In particular, we can get an *isomorphism* $elim_k(P) \cong proj_k(P) \times \mathbb{R}$.

We can illustrate above concepts with the aid of an example below, the same example was used in [101] - see Figure 3.4 for the visualization of the aforementioned concept.

$$-x_1 - 4x_2 \leq -9 \tag{3.17}$$

$$-2x_1 - x_2 \leq -4 \tag{3.18}$$

$$x_1 - 2x_2 \leq 0 \tag{3.19}$$

$$x_1 \leq 4 \tag{3.20}$$

$$2x_1 + x_2 \leq 11 \tag{3.21}$$

$$-2x_1 + 6x_2 \leq 17 \tag{3.22}$$

$$-6x_1 - x_2 \leq -6 \tag{3.23}$$

By assumption we can eliminate x_1 and ask for the possible values of x_2. Then we see that (3.20) requires $x_1 \leq 4$. All other inequalities can be rewritten to give either an upper bound on x_2 (if the coefficient of x_2 is positive), or lower bound (if the coefficient of x_2 is negative). Furthermore, there is a solution for x_2 if and only if

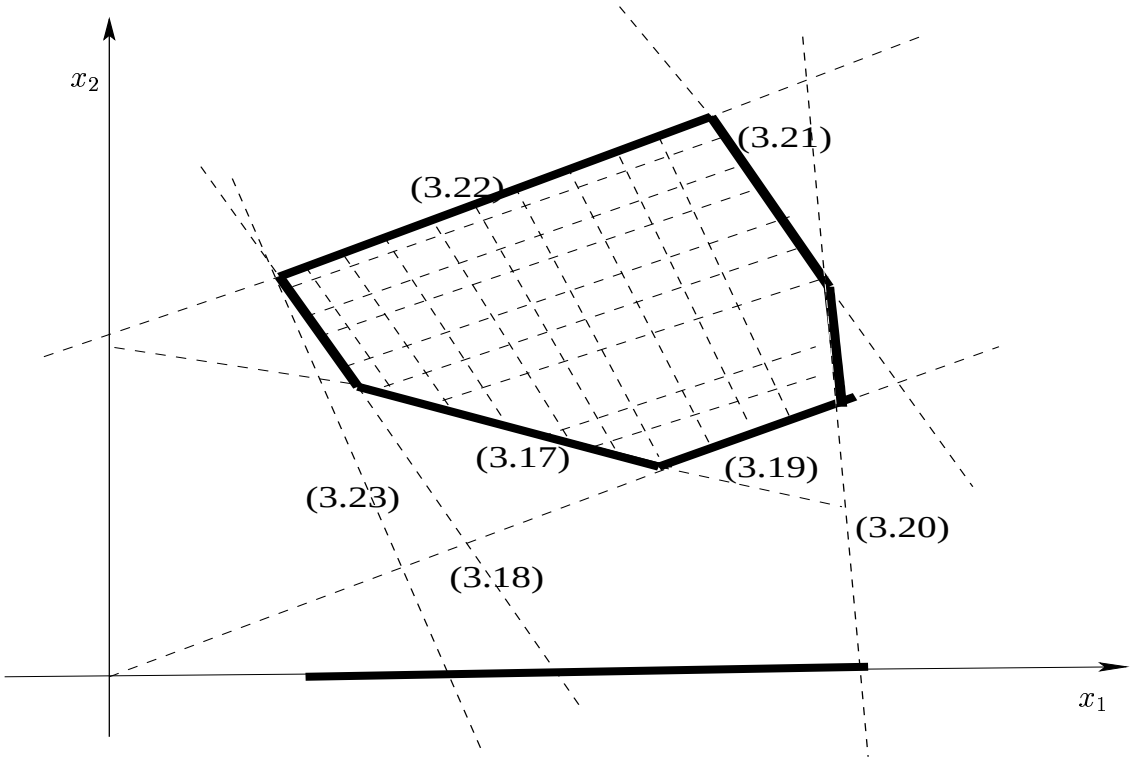


Figure 3.4: **A projection of a *2-D* polytope**

every upper bound for x_2 derived this way is larger than every lower bound.

The geometrical concept of *F-M* can be summarised as follows which is extracted from Ziegler[101]:

Theorem 4 ($Fourier - Motzkin$ Elimination-Geometrical View)
Let $P = P(A, z) \subseteq \mathbb{R}^n$ be a polyhedron, with $A \in \mathbb{R}^{m \times n}$ and $z \in \mathbb{R}^m$, and choose $k \leq n$. Construct the matrix $A^{/k} \in \mathbb{R}^{m' \times n}$ whose rows are:

- the rows a_i of A, for all i with $a_{ik} = 0$, and
- the sums $a_{ik}a_j + (-a_{jk})a_i \forall i, j$, with $a_{ik} > 0$ and $a_{jk} < 0$, and let $z^{/k} \in \mathbb{R}^{m'}$ be the corresponding column vector entries.
- z_i, $\forall i$ with $a_{ik} = 0$ and
- $a_{ik}z_j + (-a_{jk})z_i$, $\forall i, j$ with $a_{ik} > 0$ and $a_{jk} < 0$.

Then $elim_k(P) = P(A^{/k}, z^{/k})$ and $proj_k(P) = P(A^{/k}, z^{/k}) \bigcap \{x \in \mathbb{R}^k : x_k = 0\}$.

3.4.2 The Hochbaum and Naor algorithm

In general *F-M* does not run in polynomial time because of the simple fact that it may generate an exponential number of inequalities in the process of eliminating variables. However, Hochbaum and Naor show that, at each elimination step, the number of inequalities on every edge adjacent to the variable currently to be eliminated can be reduced to two. This serves as a control measure for the exponential growth of the number of inequalities. In addition, the monotone inequalities are maintained as upper and lower envelopes where the envelopes (that are piecewise linear functions) are characterized by their breakpoints. This allows one to dispose of redundant inequalities in each elimination step by quickly examining all breakpoints associated with the variable currently to be eliminated.

It may be interesting to note that the strongly polynomial feasibility algorithm for monotone inequalities does not extend to a strongly polynomial optimization algorithm over such inequalities. It is only strongly polynomial when the objective function consists of a fixed number of variables, say d.

The procedure in Aspvall and Shiloach's [3] algorithm plays an important role in Hochbaum and Naor's algorithm. Let us have an overview of this vital procedure. Suppose x_i^{min} and x_i^{max} denote the respective minimum and maximum feasible values of x_i so that any value of x_i in the range $[x_i^{min}, x_i^{max}]$ can be part of feasible solution.

Procedure 3.1 (*Aspvall* and *Shiloach*): Given a variable x_i and a constant λ, it can be decided in $O(mn)$ operations whether:

- $\lambda < x_i^{min}$ or
- $\lambda > x_i^{max}$ or
- $x_i^{min} \leq \lambda \leq x_i^{max}$.

If the procedure verifies that λ satisfies the third of these conditions, the equality $x_i = \lambda$ is propagated to the inequalities in a manner similar to the Bellman-Ford

algorithm for computing all shortest paths from a single source. It can be remarked that even if the linear program in hand is infeasible, the above procedure may still provide one of the above three answers. The high level view of the main idea behind the algorithm is as follows: the number of inequalities in which x_i (the variable to be eliminated) participates can be significantly reduced using above procedure.

The algorithm [42] is presented as follows: Let G_i denote the graph corresponding to the linear program E_i. At step i of *F-M* the following is performed:

1. Let the neighbors of x_i in the graph G_i be $x_{i1}, ..., x_{id}$. Let B_j, $1 \leq j \leq d$ denote the set of breakpoints of the edge (x_i, x_{ij}) projected on the x_i co-ordinate, sorted in ascending order.

2. Merge the d sequences B_i into a sorted sequence B (let the sorted sequence be $b_1, b_2, ..., b_k$).

3. Perform a binary search on the sequence B. The aim of the search is to either obtain:
   (a) a breakpoint $b_l \in B$ such that $x_i^{min} \leq b_l \leq x_i^{max}$, OR,
   (b) an interval $[b_l, b_{l+1}]$ $1 \leq l \leq k$ such that $b_l < x_i^{min}$ and $x_i^{max} < b_{l+1}$

4. In step 3(a) variable x_i is assigned the value b_l and contracted with vertex x_o in graph G_i. In step 3(b) the number of inequalities on each edge adjacent to x_i is reduced to at most two. Now the generic *F-M* step [99] is applied to the variable x_i.

3.4.3 An illustrative example

In this subsection we examine an example and attempt to solve it by using the algorithm of Hochbaum and Naor [42] as well as by directly applying *F-M*.

Determine a feasible solution of the following monotone inequalities:

$$y - x \leq 2 \tag{3.24}$$

$$x - y \leq 0 \tag{3.25}$$

$$2z - y \leq 3 \tag{3.26}$$

$$2y - z \leq 2 \tag{3.27}$$

$$x - 2z \leq 4 \tag{3.28}$$

$$2z - 5x \leq 1 \tag{3.29}$$

$$x,\ y,\ z \geq 0 \tag{3.30}$$

3.4.4 Solution using the *F-M* method

Let y be the variable we want to eliminate. As it does not occur in an equation in the above system then we can directly apply the second steps of the *F-M* method (i.e we eliminate y in between the inequalities) as follows.
(3.24) yields:

$$y \leq x + 2 \tag{3.31}$$

(3.25) yields:

$$x \leq y \tag{3.32}$$

(3.26) yields:

$$2z - 3 \leq y \tag{3.33}$$

And (3.27) yields:

$$y \leq 1/2(z + 2) \tag{3.34}$$

We can rearrange (3.31), (3.32), (3.33) and (3.34) as follows:

$$x, 2z - 3 \leq y \leq x + 2, 1/2(z + 2) \tag{3.35}$$

From (3.35) we can eliminate y as follows: $x \leq x + 2 \Rightarrow 0 \leq 2$ (obvious)
also we can have:

$$x \leq 1/2(z + 2) \tag{3.36}$$

$$2z - 3 \leq x + 2 \tag{3.37}$$

$$2z - 3 \leq 1/2(z + 2) \tag{3.38}$$

We can see that from (3.38) that:

$$4z - 6 \leq z + 2 \Leftrightarrow 3z \leq 8 \Leftrightarrow z \leq 8/3 \tag{3.39}$$

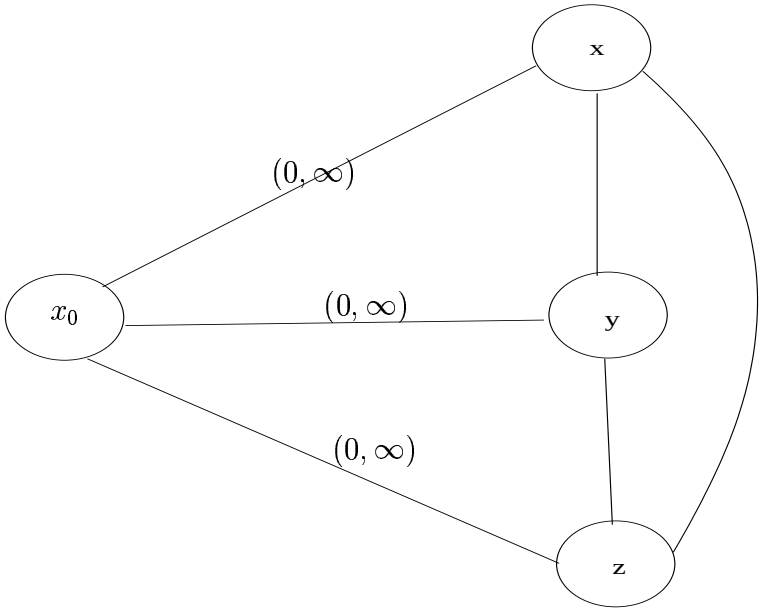


Figure 3.5: **A graph with initial lower and upper bounds**

So putting $z = 8/3$ yields, from (3.36) and (3.37), $x \leq 7/3$ and $x \geq 1/3$ respectively. Putting $x = 7/3$, for example, yields from (3.35):

$$7/3, 7/3 \leq y \leq 13/3, 7/3 \tag{3.40}$$

Setting $y = 7/3$ we finally have the following feasible solution:

$$(x, y, z) = (7/3, 7/3, 8/3).$$

3.4.5 Solution using Hochbaum and Naor's algorithm

Let z be the variable we are interested to eliminate from the given system. We have $x^{min} = y^{min} = z^{min} = 0$ from non-negativity and assume $x^{max} = y^{max} = z^{max} = \infty$. The associated graph G_0 is illustrated in Figure 3.5.

Let B_z be the set of breakpoints of constraints involving z. Then:

$$B_z = \{-2, \frac{1}{2}, \frac{3}{2}\}$$

Suppose $l = 2$, then $z = b_2 = \frac{1}{2}$ by step 3 (a). In step 4 we contract z with x_0 in the graph G_1 as illustrated in Figure 3.6. As a result we have $y \geq -2$ from (3.27), $y \leq \frac{5}{4}$ from (3.28), $x \leq 5$ from (3.29) and $x \geq 0$ from (3.31).

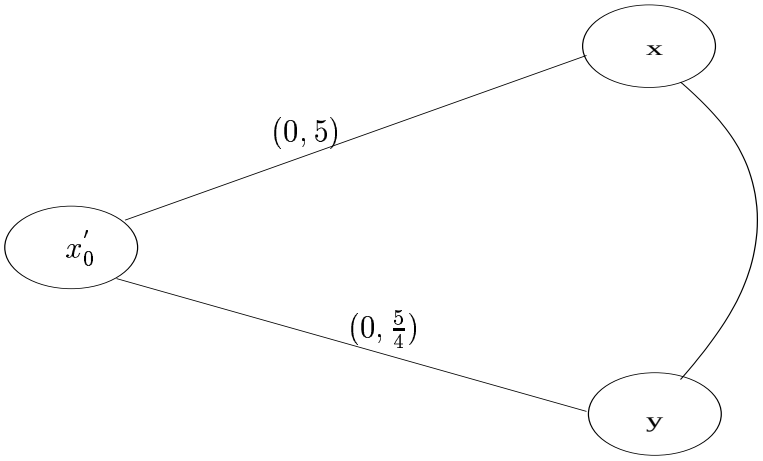


Figure 3.6: **A graph G_1 after contracting z**

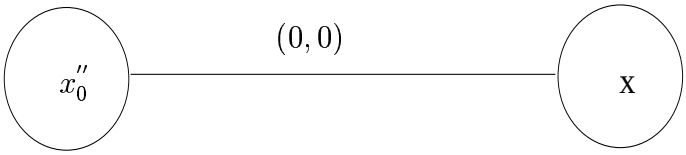


Figure 3.7: **A graph G_2 after contracting y**

We then eliminate y in between inequalities (3.24) and (3.25). Let B_y be the corresponding set of breakpoints, then:

$$B_y = \{0, 2\}$$

we suppose $l = 1 \Rightarrow y = b_1 = 0$. We apply step 4 and contract y with $x_0^{'}$ and get the resulting graph G_2 as shown in Figure 3.7. None of the original inequalities remain, so $x = 0$.

Therefore, one of the feasible solutions is $(0, 0, \frac{1}{2})$, which is also a *degenerate* basic feasible solution.

3.5 Cohen and Megiddo's Algorithm

A linear program with two variables per inequality is also called *monotone* if each inequality is of the form:

$$ax_i - bx_j \leq c \tag{3.41}$$

where both a and b are positive constants. Cohen and Megiddo [13] were able to give $O(mn^2)$ and $O(n^3 + mn)$ expected time algorithms for the feasibility problem using randomization. They have also shown that parallel implementation of these algorithms for solving systems of m linear inequalities with n variables (where each inequality involves at most two variables) requires $O(n)$ time and the deterministic algorithm uses $O(mn)$ processors and show that the randomized algorithm uses $O(n^2 + m)$ processors.

They present algorithms that deterministically solve the feasibility problem, that is, either find a point that satisfies all the inequalities or conclude that no such point exists. The analysis of Cohen and Megiddo's algorithm is quite lengthy, but the algorithm itself is simple. The underlying computation amounts to the basic Bellman-Ford and Floyd-Warshall shortest path algorithm where only simple data structures are used. The algorithms are strongly polynomial, that is, the number of operations does not depend on the size of input numbers. The size (number of bits) of the numbers generated by the algorithms is $O(nC)$ where, C is the size of the largest co-efficient in the input.

They give a time complexity of the randomized algorithm that involves many logarithmic factors ($log^5 n$). It was envisaged that, careful analysis may eliminate some of these factors. Although the algorithms given by Cohen and Megiddo [13] are theoretically tested, it is widely agreed that it could work well in practice. The *open question* is whether the $O(n^3 + mn)$ bound can be achieved deterministically. The possibility of improving the $O(n^3 + mn)$ bound can be discussed, the $O(n^3)$ factor results from the all-pairs shortest paths Floyd-Warshall computation and is inherent from some basic frameworks. A different approach, however, may yield an $O(mn)$ algorithm.

If one considers as an example, an *LI(2)* system such that the variables correspond to vertices of a three-dimensional grid and all inequalities are "local" i.e. each inequality is bounded by some constant. The associated graph has a separator de-

composition using $O(n^{\frac{2}{3}})$ size separators, and hence, these *LI(2)* systems can be solved in $O(mn)$ expected time. The feasibility problem of monotone systems includes, as a very special case, the problem of detecting existence of negative weight directed cycles in a graph with n nodes, m edges and real weights associated with the edges using a well- known reduction. The best known bound for detecting negative weight cycles is $O(mn)$ and hence it is believed that it is unlikely that an $O(mn)$ algorithm exists for solving *LI(2)* systems. The reduction can be sketched as follows: Consider a weighted graph $G = (V, E, w)$, where $w : E \rightarrow \mathbb{R}$. The corresponding monotone system is as follows: For each $v \in V$ assign a variable x_v. For each edge $e = (u, v) \in E$ assign the inequality $x_v - x_u \leq w(e)$. It follows from Cohen and Megiddo [13] that G contains a negative weight cycle if and only if the system is infeasible.

However, despite contributions of Cohen and Megiddo [13]: where some algorithms that solve the feasibility of *LI(2)* systems are adapted to find a solution that maximizes a specific variable and which can find the lexicographic maximum, some open questions remain in regard to finding an optimal solution relative to an arbitrary linear objective function over *LI(2)* systems. It is not known whether a strongly polynomial time algorithm exists.

3.6 Conclusion

In this chapter we have reviewed some algorithms for the feasibility problem in *LI(2)* systems. These algorithms either show that the system has no feasible solution or provide a single feasible solution. It is not obvious that these methods can be extended to determine all basic feasible solutions. Nevertheless, they provide some ideas which we build on in developing vertex enumeration algorithms for *LI(2)* and dual *LI(2)* systems in succeeding chapters.

Chapter 4

F-M Elimination and VE in *LI(2)*

4.1 Introduction

Primal Fourier-Motzkin (*F-M*) is of great importance in term of determining the optimum solution of systems that have 2 nonzeros per inequality. This method had been extensively described and reviewed in Chapter 3. The application of various primal *F-M* methods have also been discussed.

However, the dual of this method was not given much attention until recently when Williams [99] described briefly, how it can be used for vertex enumeration (VE) of polyhedra. Williams [98] was able to give a procedure for determining feasible solutions for LP consisting of an arbitrary number of variables (including the 2 variables per constraint problem) using *F-M* method for primal case and using a transformation for the dual case. The dual method can be used to generates all the extreme solutions (including the optimal solution) to an LP.

4.2 The Dual of Fourier's Method

The fact that every LP model has a dual model allows Williams [98] to convert Fourier's method into a dual method. For the dual method Williams [98] combines columns together two at a time, so as to eliminate constraints (rows) from the model. Ultimately, we will arrive at non-negative combinations of the columns which give the column of right-hand side coefficients of the model. The multipliers in these non-negative linear combinations will constitute feasible solutions to the dual model.

Suppose we are given the system (this is a similar problem used by [98]):

$$y_1 - 5y_2 \geq -6 \tag{4.1}$$

$$-3y_1 + 4y_2 \geq 3 \tag{4.2}$$

$$y_1 - 2y_2 \geq -2 \tag{4.3}$$

for $y_1,\ y_2 \geq 0$.

Introducing slack variables $y_3,\ y_4,\ y_5 \geq 0$ and y_0 we have:

$$6y_0 + y_1 - 5y_2 - y_3 = 0 \tag{4.4}$$

$$-3y_0 - 3y_1 + 4y_2 - y_4 = 0 \tag{4.5}$$

$$2y_0 + y_1 - 2y_2 - y_5 = 0 \tag{4.6}$$

$$y_0 = 1 \tag{4.7}$$

In order to eliminate constraint (4.4) we introduce a transformation of variables $y_0, y_1, ...,$ to variables u_1,u_2,....,. The transformation can be represented by a 'transportation' graph in which the variables with positive coefficients are the source nodes and those with negative coefficients are destination nodes. Figure 4.1 illustrates this for constraint (4.4).

$$y_0 = u_1 + u_2$$

$$y_1 = u_3 + u_4$$

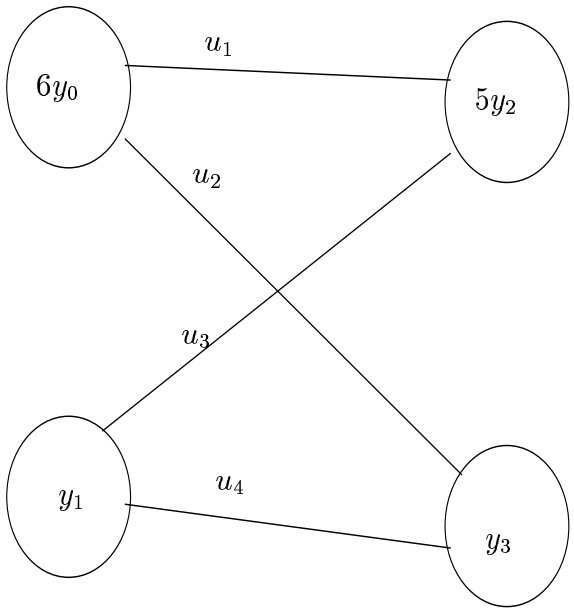


Figure 4.1: **Transformation of variables** $y \rightarrow u$

$$y_2 = 6/5u_1 + 1/5u_3$$
$$y_3 = 6u_2 + u_4$$

Then

$$y = A^1 u$$

where

$$y^T = [y_0,\ y_1,\ y_2,\ y_3,\ y_4,\ y_5] \tag{4.8}$$

$$A^1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 6/5 & 0 & 1/5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

and

$$u^T = [u_1,\ u_2,\ u_3,\ u_4,\ y_4,\ y_5] \tag{4.10}$$

In the new variables (4.5) becomes:

$$(-3,-3,4,0,-1,0)\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 6/5 & 0 & 1/5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = (9/5,-3,-11/5,-3,-1,0) \tag{4.11}$$

Similarly, (4.6) becomes:

$$(-2/5, 2, 3/5, 1, 0, -1) \tag{4.12}$$

and (4.7) becomes:

$$(1, 1, 0, 0, 0, 0) \tag{4.13}$$

We now attempt to eliminate constraint (4.11). The corresponding transportation model in Figure 4.2 leads to:

$$\begin{aligned} u_1 &= v_1 + v_2 + v_3 + v_4 \\ u_2 &= 9/15 v_1 \\ u_3 &= 9/11 v_2 \\ u_4 &= 9/15 v_3 \\ y_4 &= 9/5 v_4 \end{aligned}$$

We again describe the transformation in matrix form as $u = A^2 v$ with:

$$u^T = [u_1,\ u_2,\ u_3,\ u_4,\ y_4,\ y_5] \tag{4.14}$$

$$A^2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 9/15 & 0 & 0 & 0 & 0 \\ 0 & 9/11 & 0 & 0 & 0 \\ 0 & 0 & 9/15 & 0 & 0 \\ 0 & 0 & 0 & 9/5 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

and

$$v^T = [v_1,\ v_2,\ v_3,\ v_4,\ y_5] \tag{4.16}$$

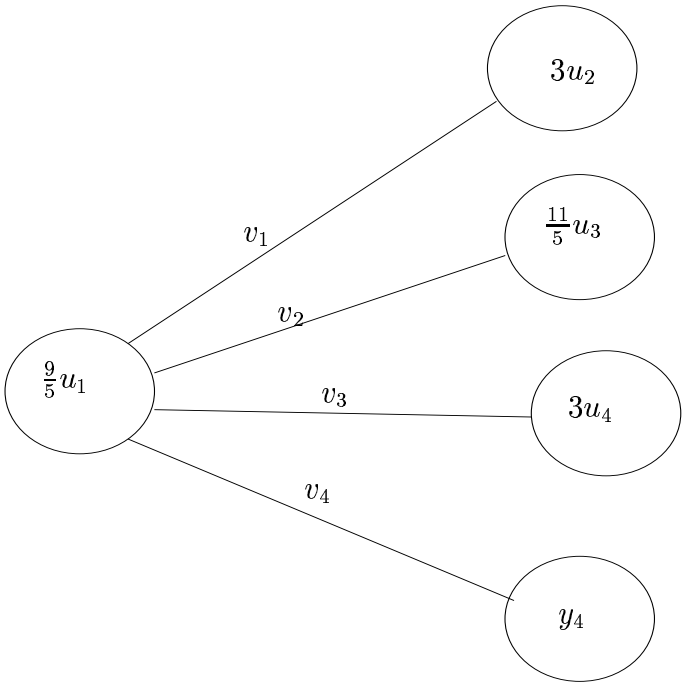


Figure 4.2: **Transformation of variables** $u \to v$

Now (4.12) becomes:

$$(4/5, 1/11, 1/5, -2/5, -1) \tag{4.17}$$

and (4.13) becomes:

$$(24/15, 1, 1, 1, 0) \tag{4.18}$$

we have $y = A^1 A^2 v$ where,

$$A^1 A^2 = \begin{bmatrix} 24/15 & 1 & 1 & 1 & 0 \\ 0 & 9/11 & 9/15 & 0 & 0 \\ 6/5 & 15/11 & 6/5 & 6/5 & 0 \\ 18/5 & 0 & 9/15 & 0 & 0 \\ 0 & 0 & 0 & 9/5 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

We now eliminate (4.17). The transportation model, Figure 4.3, yields:

$$v_1 = w_1 + w_2$$
$$v_2 = w_3 + w_4$$
$$v_3 = w_5 + w_6$$

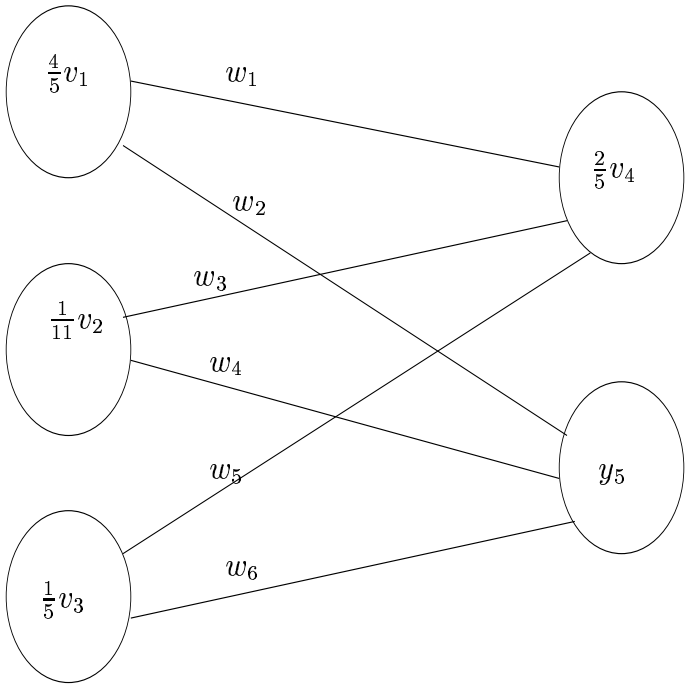


Figure 4.3: **Transformation of variables** $v \to w$

$$\frac{2}{5}v_4 = \frac{4}{5}w_1 + \frac{1}{11}w_3 + \frac{1}{5}w_5$$
$$y_5 = \frac{4}{5}w_2 + \frac{1}{11}w_4 + \frac{1}{5}w_6$$

From above we have $v = A^3 w$ with v as in (4.16) and

$$A^3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 2 & 0 & 5/22 & 0 & 1/2 & 0 \\ 0 & 4/5 & 0 & 1/11 & 0 & 1/5 \end{bmatrix} \tag{4.20}$$

$$w^T = [w_1,\ w_2,\ w_3,\ w_4,\ w_5,\ w_6] \tag{4.21}$$

(4.18) becomes:

$$(54/15, 24/15, 27/22, 1, 3/2, 1) \tag{4.22}$$

y can be written in term of w as follows:

$$y = A^1 A^2 A^3 w$$

where y is as in (4.8), w as in (4.21) and

$$A^1A^2A^3 = \begin{bmatrix} 54/15 & 24/15 & 27/22 & 1 & 3/2 & 1 \\ 0 & 0 & 9/11 & 9/11 & 9/15 & 9/15 \\ 18/5 & 6/5 & 18/11 & 15/11 & 9/5 & 6/5 \\ 18/5 & 18/5 & 0 & 0 & 9/15 & 9/15 \\ 18/5 & 0 & 9/22 & 0 & 9/10 & 0 \\ 0 & 4/5 & 0 & 1/11 & 0 & 1/5 \end{bmatrix} \tag{4.23}$$

All constraints have now been eliminated except $y_0 = 1$, so the procedure terminates. In order to obtain vertices, we scale each column of (4.23) so that the coefficient in the first row is 1. This gives:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 2/3 & 9/11 & 2/5 & 3/5 \\ 1 & 3/4 & 4/3 & 15/11 & 6/5 & 6/5 \\ 1 & 9/4 & 0 & 0 & 2/5 & 3/5 \\ 1 & 0 & 1/3 & 0 & 3/5 & 0 \\ 0 & 1/2 & 0 & 1/11 & 0 & 1/5 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{bmatrix} \tag{4.24}$$

The extreme solutions to $y_0 = 1$ are then $Z = (1,0,0,0,0,0)$, $Z = (0,1,0,0,0,0)$, $Z = (0,0,1,0,0,0)$, $Z = (0,0,0,1,0,0)$, $Z = (0,0,0,0,1,0)$, $Z = (0,0,0,0,0,1)$. However, it is clear that $Z = (0,0,0,0,1,0)$, $Z = (0,0,0,0,0,1)$ do not correspond to vertices of systems (4.1) to (4.3) as they have more than 4 positive components of y, and hence do not correspond to a basic feasible solution to (4.4) to (4.7).

Williams [100] proves two properties of the (primal) *F-M* method for which the dual analogies are:

Property 4.1: If, after eliminating m constraints by dual *F-M* elimination from a linear inequality system, a variable results from a linear combination of more than $m+1$ of the original variables, it is redundant.

Property 4.2: Any non-redundant variables, after the non-trivial elimination of m constraints, depend on *at most* $m+1$ of the original variables.

The implication of property 4.1 is that z_5, z_6 can be deleted from (4.24) and hence that the vertices of (4.4) to (4.7) are:

$$v_1 = (0, 1, 1, 1, 0)$$
$$v_2 = (0, 3/4, 9/4, 0, 1/2)$$
$$v_3 = (2/3, 4/3, 0, 1/3, 0)$$
$$v_4 = (9/11, 15/11, 0, 0, 1/11)$$

It is easily verified graphically that this is the complete and correct set of vertices. In fact, as our experiments showed, failure to eliminate redundant variables in the intermediate stages of the method can lead to duplication of vertices.

F-M is a potentially useful method for vertex enumeration because it is unaffected by primal degeneracy. However a known problem with this method is that, in the general case, the number of variables introduced through the transformations may grow exponentially. We investigate whether this is the case for *LI(2)* problem or its dual. The theoretical analysis will be given later. It should be noted that Williams [98] did not give an explicit algorithmic description of the dual *F-M* method for enumerating vertices of polyhedra, and none is available in any literature. Such a description is given below.

4.3 An Algorithm for Vertex Enumeration via Dual *F-M*

Dual *F-M* uses the idea of eliminating constraints in between variables and can be expressed in terms of matrix transformations, as illustrated in Figure 4.4.

Suppose $a(:,:)$ holds the constraint matrix, $T(:,:)$ holds the cumulative transformation matrix, and $T_{-1}(:,:)$ holds the current matrix. A row of $a(:,:)$ is used to get the current transformation matrix $T_{-1}(:,:)$. The cumulative matrix $T(:,:)$ is the same as the current matrix $T_{-1}(:,:)$, when the first constraint of $a(:,:)$ is eliminated. Subsequently we set $T(:,:)_{new} \leftarrow T(:,:)_{old} * T_{-1}(:,:)$. The constraint matrix

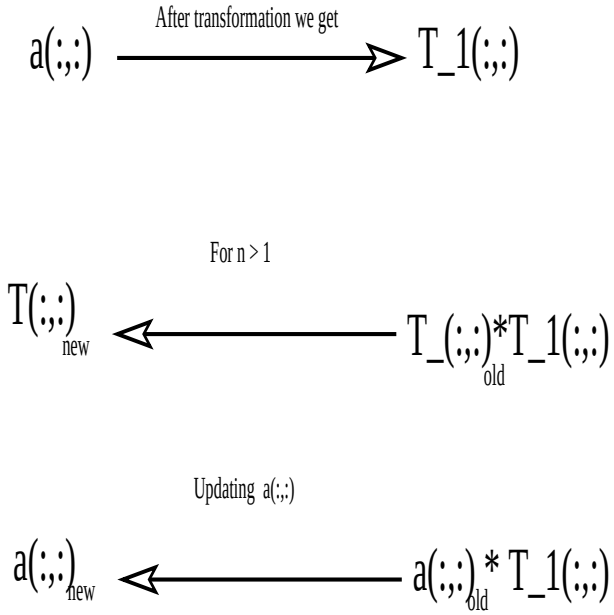


Figure 4.4: **Matrix transformations in Dual *F-M***

$a(:,:)$ is therefore updated as: $a(:,:)_{new} \leftarrow a(:,:)_{old} * T_{-1}(:,:)$. At each iteration the columns of $T(:,:)$ that have more than $n+1$ variables are redundant and are deleted. The corresponding columns of $a(:,:)$ are also deleted. It is also clear that if, at any stage, all non-zero coefficients in a constraint have the same sign, the corresponding columns can be deleted. This is the case because the corresponding variables must have value 0 as the constraints are homogeneous equations in non-negative variables.

A comprehensive algorithmic description of the procedure to find the set of vertices $\{v^1, v^2, ..., v^q\}$ of:

$$Ax = b, \qquad x \geq 0 \tag{4.25}$$

is given below. We assume w.l.o.g $b \leq 0$

Preliminaries:

Let

$$\Psi(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x \neq 0 \end{cases}$$

e_j^s be a s-dimensional unit vector with a 1 in the j^{th} element (i.e. it is a column vector), dim(T) be the number of columns in T and P_i, N_i, Z_i be the number of

positive, negative and zero coefficients in row i, respectively.

Algorithm

- Initialisation

$$y \leftarrow \begin{bmatrix} y_o \\ x \end{bmatrix} \qquad a^o \leftarrow \begin{bmatrix} -b & A \\ 1 & 0 \end{bmatrix}$$

$$T^o \leftarrow I_{n+1} \qquad t \leftarrow n+1$$

FOR $i = 1, 2, ..., m$ **DO**

- Elimination

IF $P_i = 0 \wedge N_i = 0$, **CYCLE** (constraint i is redundant)

IF $P_i = 0 \vee N_i = 0$ **THEN**

$C \leftarrow \{c_j : j = 1, 2, ...k\}$ where $c_1 < c_2 < .... < c_k$, and $a_{ij}^{i-1} = 0$

$T^i \leftarrow T^{i-1}$

GO TO *Reduction*

$T' \leftarrow 0; \quad S \leftarrow 0; \quad K_p \leftarrow 0; \quad K_n \leftarrow 0; \quad K_0 \leftarrow 0$

FOR $r \leftarrow 1$ **TO** t **DO**

IF $a_{i,r}^{i-1} > 0$ **THEN**

FOR $\delta \leftarrow 1$ **TO** N_i **DO** $T'_{r,S+\delta} \leftarrow \frac{1}{a_{i,r}}$

$S \leftarrow S + N_i; \; K_p \leftarrow K_p + 1$

$w(K_p) \leftarrow (K_p - 1) * N_i + K_0$

IF $a_{i,r}^{i-1} = 0$ **THEN**

$S \leftarrow S + 1; \quad K_0 \leftarrow K_0 + 1; \quad T'_{r,S} \leftarrow 1$

END DO

FOR $r \leftarrow 1$ **TO** t **DO**

IF $a_{i,r}^{i-1} < 0$ **THEN**

$K_n \leftarrow K_n + 1$

FOR $\delta \leftarrow 1$ **TO** P_i **DO** $T'_{r,w(\delta)+K_n} \leftarrow \frac{-1}{a_{i,r}}$

END DO

$a^i \leftarrow a^{i-1}T' \qquad T^i \leftarrow T^{i-1}T'$

$C \leftarrow \{c_j : j = 1, 2, ...k\}$ where $c_1 < c_2 < .... < c_k$
and $\sum_{p=1}^{n} \Psi(T^i_{pc_j}) < i + 1$

- Reduction

  $T^i \leftarrow T^i[e^t_{c_1}, e^t_{c_2}, ..., e^t_{c_k}]$

  $t \leftarrow dim(T^i)$

  END DO

- Extraction

  $v^j \leftarrow (\frac{T^i_{2c_j}}{T^i_{1j}}, \frac{T^i_{3c_j}}{T^i_{1j}}, ..., \frac{T^i_{(n+1)c_j}}{T^i_{1_j}}), \quad (j = 1, 2, ..., q)$
  where $c_1 < c_2 < .... < c_q$ and $T^i_{1c_j} \neq 0, \quad \square$

4.4 Computational Results

The algorithm described in previous section has been implemented in Fortran 90 using the Portland Group pgf90 compiler running under Linux. Table 4.1 records execution times and the maximum number of intermediate variables for a small set of test problems taken either from this thesis or [23]. Several variants of each problem, derived by arbitrary reorderings of the constraints, were run and the best and worst cases are recorded in Table 4.1. Results were obtained on a 700MHz Pentium III processor.

The results clearly show that the performance of the dual *F-M* method is sensitive to the ordering of the constraints. However it is not clear how to identify a static ordering which would control the growth in the number of intermediate variables. Suppose that constraint i is being eliminated and that:

$$p = |\{a_{ij} : a_{ij} > 0\}|, \; n = |\{a_{ij} : a_{ij} < 0\}|, \; z = |\{a_{ij} : a_{ij} = 0\}|$$

then the number of variables in the transformed constraints is $v = p \times n + z$. Hence a simple greedy heuristic for dynamically ordering the constraints is, at each iteration,

Problem	Variables	Constraints	Vertices	Variant	Max No of Variables	Time (secs)
1	3	5	3	a	6	0.01
				b	7	0.01
2	4	10	36	a	96	0.02
				b	246	0.10
3	12	7	31	a	310	0.47
				b	1276	0.67
4	5	10	38	a	199	0.22
				b	383	0.43
5	6	10	64	a	496	1.27
				b	1351	3.12
6	7	10	62	a	4145	49.62
				b	> 6500	-

Table 4.1: **Computational results for the *F-M* code**

to choose to eliminate that for which v is smallest. Table 4.2 records the results using this ordering of the constraints. These results suggest that greedy ordering does have a good effect. However the performance of the dual *F-M* method does not seem to compare with that of other methods. Problem 3, for example, is the problem referred to as 'ex1' in Table 2.1. The Provan code solved this problem significantly faster on a considerably slower computer. Problems 4-6 are solved much faster by the hash variant of Dyer's code, e.g. problem 6 takes only 0.03 secs.

4.5 Complexity Analysis for Dual *F-M*

Suppose we wish to solve the following system:

$$Ax = 0 \tag{4.26}$$

and,

$$b^T x = 1, \quad x \geq 0$$

Problem	Max Variables	Time(secs)
1	6	0.01
2	96	0.02
3	125	0.47
4	380	0.40
5	1052	2.10
6	1220	13.07

Table 4.2: **Computational results for greedy ordering**

Example: flows x through a network with given cost, say c, where A has at most two nonzeros per column, which is associated with a digraph, say G of m nodes and n edges, and $b \geq 0$, is the available supply of an item. And $b < 0$ if items are demanded (note: $b = 0$ implies none of the item is available or demanded, and the associated node in this case is called *transshipment* or *intermediate* node). Then, of interest, is the dual of (4.26) which is:

$$A^T y \geq b \tag{4.27}$$

and where A^T is a two per row matrix. We can use *F-M* on the dual (4.27) to (4.26), as follows:

Suppose we are trying to "eliminate" row i from system (4.26). We can do so by eliminating x_i in the dual system (4.27). Hochbaum and Naor [42] discuss solving LPs in this setting. They do not consider generating all vertices of polyhedra formed by *LI(2)*. Hochbaum and Naor's idea [42] is to solve *LI(2)* using *F-M* by constraining variables between breakpoints. As vertex enumeration is concerned with all the vertices of the polyhedron, this idea is no longer valid. We must develop new algorithms, using some inspiration from [42]. The new idea is based on considering the 3-dimensional relationship between variables, say x_j and x_k, and another variable, say x_i, which is to be eliminated.

4.6 Analysis of the Algorithm

Suppose we are to solve (4.27) using *F-M*. Specifically, we are interested in *F-M* on systems with 2 variables per constraint. Suppose m is the total number of constraints, let x_1, x_2,.....,x_n be the variables and m_{ij} be the number of constraints involving x_i and x_j, where $m_{ij} = m_{ji}$. Assume that $0 \leq m_{ii} \leq 2$ since for m_{ii}, we can have only the following:

$$c_1 x_i + 0x_1 \geq b \tag{4.28}$$

which gives either a lower or upper bound on x_i. Suppose the associated graph G (e.g a bipartite graph with m sources and n sinks) is connected, that is, there exists a *chain* or *path* between every node in G. This implies that the total number of new constraints is bounded by:

$$\sum_{j,k}(m_{ij} + m_{ik}) \leq 2(n-1)m_i \tag{4.29}$$

where

$$m_i = \sum_{j \neq i} m_{ij} \tag{4.30}$$

We know that:

$$\sum_{i=1}^{n} m_i = 2m \tag{4.31}$$

So, $\exists\ i$ such that:

$$m_i \leq \tfrac{2m}{n}$$

where m_i is the number of two-variable constraints containing x_i and m is the total number of such constraints.

We can choose x_1 (by reordering variables if necessary) as the variable to be eliminated, so that from (4.28) we can have:

$$m_1 = \min_i\{m_i\} = \min_i \sum_{j \neq i} m_{ij} \tag{4.32}$$

Since:

$$\sum_{i=1}^{n} m_i = \sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij} = 2m$$

we have:

$$m_1 \leq \frac{2m}{n} \tag{4.33}$$

Let us consider elimination of variable x_1 from constraints involving x_i and x_j. The resulting graph is illustrated in Figure 4.5.

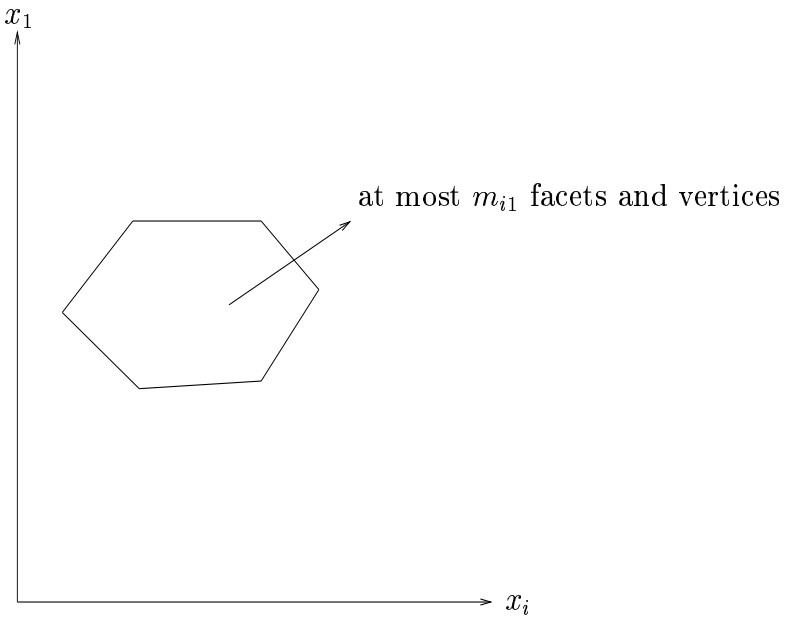


Figure 4.5: **Projection of x_1 along y-axis and x_i along x-axis**

The elimination of x_1 is easily visualizable in 3-dimensional space as can be seen clearly in Figure 4.6. The assumption is that all variables x_1, x_2,.....,x_n are bounded. As can be observed from Figure 4.6, x_1 is plotted along the z-axis, x_i along the x-axis and x_j along the y-axis. Perpendicular to the x_1, x_i plane lies a cylinder with at most m_{i1} facets. Similarly, perpendicular to the x_i, x_j plane lies another cylinder with at most m_{j1} facets. The intersection of these two cylinders gives a 3-dimensional polyhedron with at most $(m_{i1}+m_{j1})$ facets, and less than 3-dimensional polyhedron with at most $(m_{i1} + m_{j1})$ facets, and less than $2(m_{i1} + m_{j1})$ vertices and $3(m_{ij} + m_{ji})$ edges. (See below).

The projection of the 3-dimensional polyhedron along x_i, x_j plane is obtained after elimination of the variable x_1 from the system. The result is a 2-dimensional

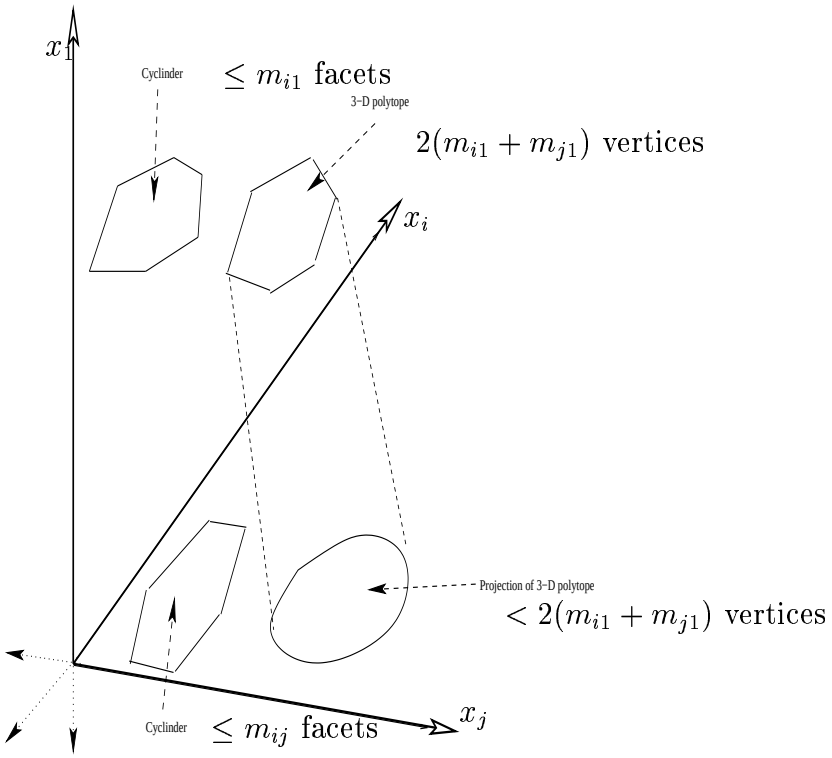


Figure 4.6: **A *3-D* polyhedron and its *2-D* projection on x_i and x_j plane**

polyhedron, each vertex of which results from projection of a vertex from the 3-dimensional polyhedron. The number of constraints i, after elimination of x_1, is less than or equal to the number of vertices of the 3-dimensional polyhedron, which we now show is less than $2(m_{i1} + m_{j1})$.

Claim 4.1: The number of vertices of a 3-dimensional polyhedron with at most $(m_{i1} + m_{j1})$ facets is at most $2(m_{i1} + m_{j1})$.

Proof:
Let f be the number of facets, e be the number of edges and v be the number of vertices of the polyhedron.

For any 3-dimensional polyhedron, the following are true:

$$3v \leq 2e \tag{4.34}$$

since each vertex is adjacent to at least 3 edges, and by Euler's formula:

$$v - e + f = 2 \tag{4.35}$$

From (4.34) we have:

$$v + f - 2 = e \geq \frac{3v}{2} \tag{4.36}$$

From that we have:

$$f - 2 \geq \frac{v}{2} \tag{4.37}$$

And then we have:

$$v \leq 2(f - 2) < 2f \tag{4.38}$$

Thus,

$$v < 2f \tag{4.39}$$

So the 3-dimensional polyhedron is less than $2(m_{i1} + m_{j1})$ vertices. QED

After eliminating x_1, let the number of constraints corresponding to m_i be $m_i^{'}$, and the number of constraints corresponding to m_{ij} be $m_{ij}^{'}$.

Our goal is to find the relationship between the total number of constraints in the original system, m, and the total number of constraints, $m^{'}$, after eliminating the variable x_1 which occurs in the fewest constraints.

After eliminating variable x_1, the following inequality holds:

$$m_{ij}^{'} < m_{ij} + 2(m_{i1} + m_{j1}) \tag{4.40}$$

since m_{ij} is the old number of constraints involving i and j, and $2(m_{i1} + m_{j1})$ is at most the number of vertices after eliminating variable x_1, where $i \neq j \neq 1$.

If we sum (4.40) over i and j, we have:

$$\sum_{i=2}^{n}\sum_{j=2}^{n} m_{ij}^{'} < \sum_{i=2}^{n}\sum_{j=2}^{n} m_{ij} + 2\sum_{i=2}^{n}\sum_{j=2}^{n}(m_{i1} + m_{j1}) \tag{4.41}$$

This gives:

$$2m^{'} < 2(m - m_1) + 4(n - 1)m_1 \tag{4.42}$$

and then using (4.33) we have:

$$m^{'} < m + (2n-3)m_1 \leq m + (2n-3)\frac{2m}{n} < 5m \tag{4.43}$$

Thus,

$$m^{'} < 5m \tag{4.44}$$

Suppose there are M_k constraints when k variables remain during the elimination. Then we have:

$$M_k < 5M_{k+1} \tag{4.45}$$

So we have:

$$M_2 < 5^{n-2}M_n = 5^{n-2}m \tag{4.46}$$

We can improve the argument above, as follows: Any edge in the 3-dimensional polyhedron arises from some edges in one of the two intersected polyhedra. There are only $(m_{i1} + m_{j1})$ such edges. Also, any edge in the 2-dimensional projection contains the projection of at least one edge in the 3-dimensional polyhedron. Hence the projection can, in fact, have at most $(m_{i1} + m_{j1})$ edges and vertices. This removes a factor of 2, and repeating the above analysis gives us a bound of $3^{n-2}m$, rather than $5^{n-2}m$. Thus, it is possible to generate at most $3^{n-2}m$ non-redundant inequalities at any level of the elimination.
i.e.

$$M_2 < 3^{n-2}m \tag{4.47}$$

Property 4.3: For the system (4.26) consisting of at most two non-zeros per column, this property remains true in the process of eliminating constraints.

Proof :
Suppose we have a two non-zeros per column system of the form $A^{'}x = b$ where:

$$A^{'} = \begin{bmatrix} b_1 & b_2 & . & . & b_n \\ . & a_{2,2} & . & 0 & 0 \\ . & . & . & a_{3,j} & . \\ a_{4,1} & 0 & . & . & . \\ . & . & . & . & . \\ 0 & 0 & . & a_{m,j} & 0 \end{bmatrix} \tag{4.48}$$

and $x_1,\ x_2, .., \ x_n \geq 0$.

In order to find feasible solutions, we start eliminating constraints by non-negative combination of columns. In each case we end up with a two non-zeros per column matrix. Since all $b_i \geq 0$, we do not need to eliminate the first row. Then, for example, if we want to eliminate constraint 2 from (4.48), we combine a multiple of column 1 and column 2 and end up with the following:

$$A_1^{'} = \begin{bmatrix} b_1 & b_2 & b_3 & . & . \\ c_{3,1} & . & c_{3,j} & . & . \\ 0 & . & . & . & . \\ . & . & . & c_{j,j} & . \\ 0 & . & 0 & . & c_{m-1,n} \end{bmatrix} \tag{4.49}$$

It can be noted that the entries in column 1 and column 2 for row 1 disappeared and still we have two non-zero per column matrix as in equation (4.49). Whichever row we eliminate we still end up with a two non-zero per column matrix. QED

4.7 Conclusion

In this chapter we have studied dual *F-M* and find that we can use it to enumerate vertices of polyhedra by eliminating constraints one at a time, as suggested by Williams [99]. One problem with this method is when a system has many redundant constraints. A lot of energy may be used to eliminate constraints that may not be important after all. We have presented an original algorithmic description of vertex enumeration via dual *F-M*. Some computational results are presented which show little promise for the method, even with a dynamic constraint ordering heuristic. The relatively poor performance is particularly important as the method is a dual method and thus does not provide any vertices until the algorithm terminates. We have also presented theoretical results about dual *F-M* for certain dual *LI(2)* systems and shown that for systems where variables intersect (are present in between constraints), there is an upper bound which is *exponential* only in the number of columns, but *linear* in the number of rows.

Chapter 5

A BOP Algorithm for VE in *LI(2)*

5.1 Introduction

In Chapter 4 we have discussed how the Fourier-Motzkin (*F-M*) elimination method and its dual method are used for vertex enumeration (VE) of polyhedra associated with LP that have at most two variables per constraint. Various methods were initially reviewed for the systems that are sometimes referred to as *monotone inequalities* (inequalities of the form $ax - by \leq c$, where both a and b are positive). The methods included those of Hochbaum and Naor [42], Aspvall and Shiloach [3], Shostak [90] and Megiddo [13]. Hochbaum and Naor's method [42] is $O(mn^2 logm)$ and is one of the most efficient in term of theoretical efficiencies. The backbone of their algorithm is *F-M*. Williams [98] also exploited and adapted *F-M* to solve linear programmes and to generate all the vertices of polyhedra. We showed that, except in a special case, the number of variables does not grow exponentially in the process of *F-M* for *LI(2)* systems. However the number of variables does grow rapidly enough for this to be a significant practical problem.

We now look at a different algorithm, which may be classified as a basis oriented

pivoting method [23]. Two important propositions on the structure of the basis for *LI(2)* and dual *LI(2)* systems form the backbone of our new algorithm.

5.2 Basis Structure for Dual *LI(2)*

The question that can be raised at this junction is, what does a basis of LP consisting of no more than two variables per constraint (column) look like?

The following result is for two nonzeros per column, we will later give that of its dual (system with no more than two per row). The backbone of our investigation is the following proposition.

Proposition 5.1: Let $B = \{x \in \mathbb{R}^n : Ax = b, b \in \mathbb{R}^m, \ \ x \geq 0\}$ be a linear program with at most two non-zeros per column in A, where A is an $m \times n$ matrix; x an $n \times 1$ matrix and b an $m \times 1$ matrix, with $n > m$. Let G be a graph on m vertices corresponding to the rows of A, with an edge corresponding to each column of A (i.e. an edge between i and j if these are the non-zero positions in the column). Then the set of edges in G corresponding to any basis of A has vertex-disjoint components which are either:

1. A tree OR
2. Contain exactly one cycle.

Proof :
Suppose A is the matrix with 2 nonzeros per column. Each column of A can be

represented as follows:

$$c_{i,j} = \begin{bmatrix} 0 \\ 0 \\ . \\ a_{i,j} \\ . \\ d_{i,j} \\ 0 \\ 0 \end{bmatrix} \tag{5.1}$$

We can start building up components of the basis by rearranging rows and columns (using only the edges that correspond to the basis). At each stage we add a column which has one non-zero in a new-row, but may also have a non-zero in a row already used. Obviously, the set of edges that form the basis in this case cannot have more than 2 edges (variables) that are connected to 2 adjacent nodes, otherwise we have 3 vectors spanning $\mathbb{R}^2$.

Rearranging the rows and columns of B we obtain a matrix of the form:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & . & 0 & 0 & . & 0 \\ d_{2,1} & 0 & . & 0 & 0 & . & 0 \\ 0 & d_{3,2} & . & a_{3,9} & 0 & . & a_{3,n} \\ 0 & 0 & . & 0 & 0 & . & 0 \\ 0 & 0 & . & d_{5,9} & a_{5,10} & . & d_{5,n} \\ 0 & 0 & . & 0 & d_{6,10} & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & . & 0 & 0 & . & 0 \end{bmatrix} \tag{5.2}$$

As we can see from above matrix, we reach an *upper triangular matrix* with one additional row as the component, when a stage is reached where one can add no more columns.

If we have r rows, then we have *r-1* linearly independent vectors, and the structures of the arranged matrix yield a tree with *r-1* edges plus (possibly) one edge giving r

spanning vectors. This is a component in G with one cycle (if we terminate with a tree then these rows have rank deficiency 1). We can remove this component and start again. We obtain a collection of vertex disjoint components, each having at most one cycle.

Indeed, the basis corresponds to a collection of vertex disjoint components of G each of which may be a tree or contains at most one cycle. QED

Corollary 5.1: If i is the number of component trees for any basis, then the rank of the matrix A is given by:

$$Rank(A) = m - i \tag{5.3}$$

(i.e. i components are missing rank 1).

Proof :
Follows from the proof of above Proposition.

Corollary 5.2: For a connected network LP system, every basis corresponds to a spanning tree.

Proof :
Since the non-zeros are all $+1$ and -1 in every column, there can be no *non-singular cycles*. If the graph is connected it has at least one spanning tree. This is a basis and hence all other bases must correspond to spanning trees.

5.2.1 Example

Consider the following system:

$$x_1 + 2x_3 + 2x_4 = 7 \tag{5.4}$$

$$-x_1 + 2x_2 - x_8 = 4 \tag{5.5}$$

$$-3x_2 + x_3 + 4x_5 = -1 \tag{5.6}$$

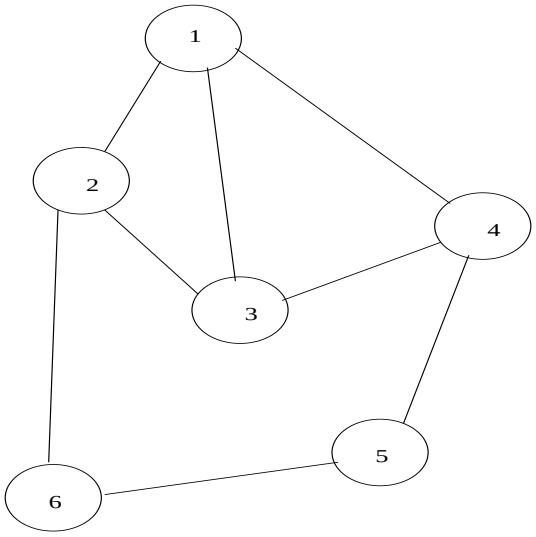


Figure 5.1: **The graph of the matrix A**

$$-2x_4 + 3x_5 + x_6 = 2 \tag{5.7}$$

$$3x_6 + 2x_7 = 3 \tag{5.8}$$

$$4x_7 + 5x_8 = 5 \tag{5.9}$$

The above system can be re-written in matrix form as: $Ax = b$, where:

$$A = \begin{bmatrix} 1 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & -3 & 1 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 5 \end{bmatrix} \tag{5.10}$$

$$x^T = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8] \tag{5.11}$$

and :

$$b^T = [7, 4, -1, 2, 3, 5] \tag{5.12}$$

The graph of A is shown in Figure 5.1.

We can start building up components of the basis by rearranging rows and columns as follows. Consider an $m \times m$ submatrix B of A, with columns; 1, 2, 4, 6, 7 and

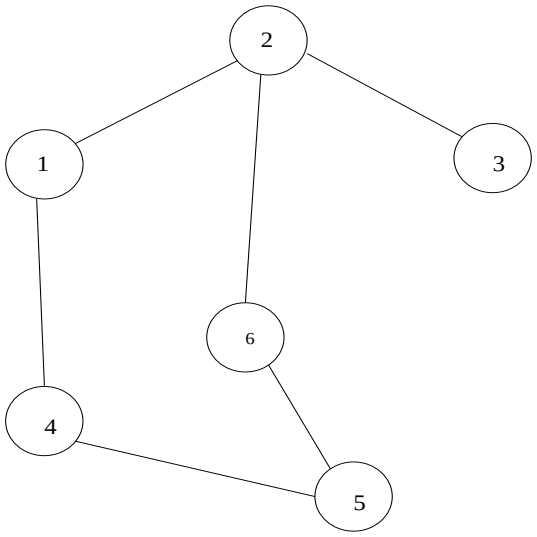


Figure 5.2: **A graph G of the basis component B with one cycle**

8 selected from A and arranged in that order i.e. column 1 of B is column 1 of A, column 2 of B is column 2 of A, column 3 of B is column 4 of A, column 4 of B is column 6 of A, column 5 of B is column 7 of A, column 6 of B is column 8 of A, we get:

$$B = \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 & -1 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 & 5 \end{bmatrix} \tag{5.13}$$

B may be considered as one basis of the above LP with the rank of A given as $Rank(A) = m - i$, where i is the number of trees. The graph of B is shown in Figure 5.2, in this case there is no tree, so $i = 0$, i.e. $Rank(A) = 6 - 0 = 6$.

The determinant of B can be found using row 3 of the matrix and is given as: $Det(B) = |B| = 3(20 + 24) = 132$. So the determinant of B is non-zero. Thus, B is a non-singular matrix that has at most one cycle. In general, we can remove this component and start again. Note that there is only one component here.

If A is a matrix with two nonzeros per column, then its dual is a matrix, say A^T with two nonzeros per row. Based on this relationship, what will be the graphical structures for basis of LP with at most two nonzeros per row? We try to investigate an answer to this question below.

5.3 Basis Structure for *LI(2)*

Proposition 5.2: Let $\beta = \{y \in \mathbb{R}^m : A^T y \leq c, \quad c \in \mathbb{R}^n\}$ be a linear program with at most two non-zeros per row in A^T, where A^T is an $n \times m$ matrix; y an $m \times 1$ matrix and c an $n \times 1$ matrix, with $n > m$. Let G' be a graph on m vertices corresponding to the columns of A^T, with an edge corresponding to each row of A^T (i.e. an edge between r and k if these are the non-zero positions in the row). Then the set of edges in G' corresponding to any basis of A^T has vertex-disjoint components which are either:

1. A tree OR
2. Contain exactly one cycle.

Proof :
A basis is determined by m rows of A^T, since there are no sign-restrictions. Proof follows from proposition 5.1 above, adding rows of A^T instead.

5.3.1 Example

Consider the following system:

$$3y_1 + 4y_2 \leq c_1 \tag{5.14}$$

$$-2y_1 + 3y_3 \leq c_2 \tag{5.15}$$

$$5y_2 - y_4 \leq c_3 \tag{5.16}$$

$$6y_2 + 8y_5 \leq c_4 \tag{5.17}$$

$$3y_3 - 3y_4 \leq c_5 \tag{5.18}$$

$$2y_3 - 2y_5 \leq c_6 \tag{5.19}$$

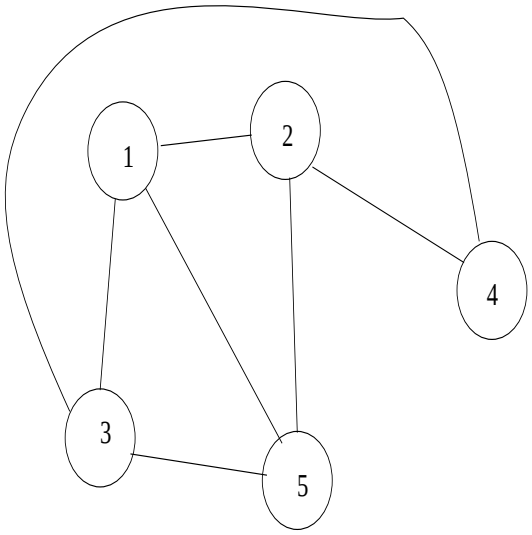


Figure 5.3: **A graph for the matrix** A^T

$$2y_1 - 4y_5 \leq c_7 \tag{5.20}$$

The above system can be re-written as:

$$\begin{bmatrix} 3 & 4 & 0 & 0 & 0 \\ -2 & 0 & 3 & 0 & 0 \\ 0 & 5 & 0 & -1 & 0 \\ 0 & 6 & 0 & 0 & 8 \\ 0 & 0 & 3 & -3 & 0 \\ 0 & 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \leq \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} \tag{5.21}$$

which is a system of the form $A^T y \leq c$, where:

$$A^T = \begin{bmatrix} 3 & 4 & 0 & 0 & 0 \\ -2 & 0 & 3 & 0 & 0 \\ 0 & 5 & 0 & -1 & 0 \\ 0 & 6 & 0 & 0 & 8 \\ 0 & 0 & 3 & -3 & 0 \\ 0 & 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & 0 & -4 \end{bmatrix} \tag{5.22}$$

The graph of A^T is given in Figure 5.3.

Let us start building one of the components of the basis by rearranging and selecting rows and columns of A^T as follows. Consider an $n \times n$ submatrix B^T with $n = 5$ then we choose B^T as 5×5 matrix with rows; 1, 2, 5, 6 and 7 of A^T selected and arranged in that order i.e. row 1 from row 1 of A^T, row 2 from row 2 of A^T, row 3 is row 5 of A^T, row 4 is row 6 of A^T and row 5 is row 7 of A^T. We get B^T as follows:

$$B^T = \begin{bmatrix} 3 & 4 & 0 & 0 & 0 \\ -2 & 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & -3 & 0 \\ 0 & 0 & 2 & 0 & -2 \\ 2 & 0 & 0 & 0 & -4 \end{bmatrix} \tag{5.23}$$

B^T may be considered as one of the bases for the above linear inequalities with the rank of matrix A^T given as $Rank(A^T) = n-i$, where i is the number of trees. Figure 5.4 shows the graph of B^T which contains no tree. So $Rank(A^T) = n-i = 5-0 = 5$.

B^T is non-singular, since its determinant is $Det(B^T) = 12(16 - 12) = 48$. The graph of B^T from Figure 5.4 consists of component graph with only one cycle, as predicted by Proposition 5.2. We can remove this component and start again. Note that we have only one component here from our example, it is possible to select another set of rows from matrix A^T and in each case, we may either end up with graph whose components are trees or consists of at most one cycle each.

Theorem 5.1: Let $B_n^T = \{x \in \mathbb{R}^n : Tx = b,\ b \in \mathbb{R}^n \text{ and } x \geq 0\}$ be a linear program with at most two non-zeros per column with T as $n \times n$ triangular matrix and x an $n \times 1$ matrix with b an $n \times 1$ matrix. Let G_T be a graph on n vertices with an edge corresponding to each column of T. Then the set of edges in G_T corresponds to any basis of vertex-disjoint components of T if and only if it is a tree.

Proof :
By induction:

Suppose T is an $n \times n$ triangular matrix with at most two non-zeros per column.

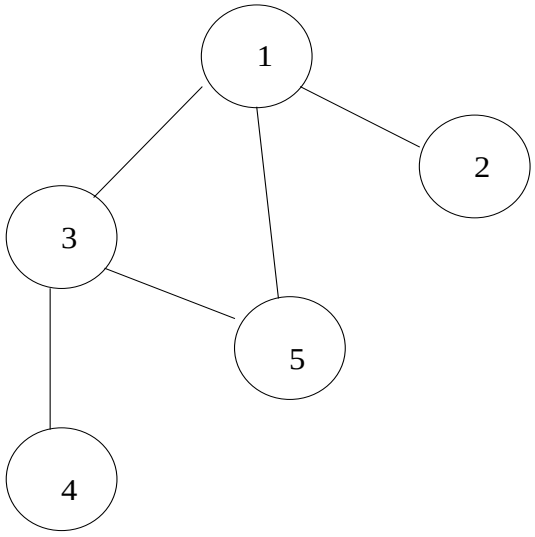


Figure 5.4: **The graph for the basis matrix B^T**

Choose $n = 2$ then we can have a triangular matrix B_T^2 as a basis of the form:

$$B_T^2 = \begin{bmatrix} a_{1,1} & a_{1,2} \\ 0 & a_{2,2} \end{bmatrix} \tag{5.24}$$

The graph corresponding to the basis 2×2 triangular matrix B_T^2 is given in Figure 5.5.

Suppose it is true for k, where k is as large as possible. Our *goal* is to show that it is true for n. The graph for the basis $k \times k$ triangular matrix B_T^k given in Figure 5.6.

The basis triangular matrix for $n = k$ is obtained as follows: We start with a column that has only one non-zero as its first row, then we add second column in such a way that its second row entry is non-zero, and all entries beneath this row in the same column are zeros, we continue in a similar fashion, until we can add no more columns. Note that all the entries in the main diagonal of the matrix have to be non-zeros and each column must have at most two non-zeros, with exception of the

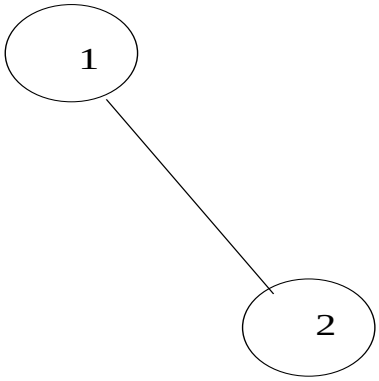


Figure 5.5: **A tree graph for matrix** B_T^2

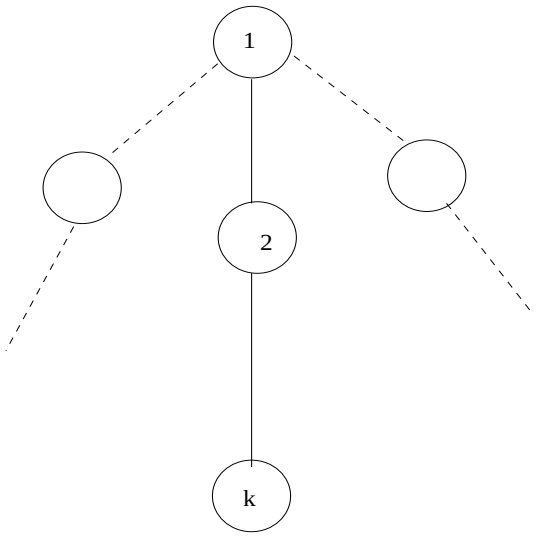


Figure 5.6: **A tree graph for matrix** B_T^k

first column. The basis B_T^k matrix graph in this case is given as follows.

$$B_T^k = \begin{bmatrix} a_{1,1} & a_{1,2} & . & 0 \\ . & a_{2,2} & . & a_{2,k} \\ . & . & . & . \\ 0 & 0 & . & a_{k,k} \end{bmatrix} \tag{5.25}$$

Thus, we can continue to add columns in a similar way until all the n columns of T are exhausted. In all there will be only spanning tree with no cycle. We get B_T^n

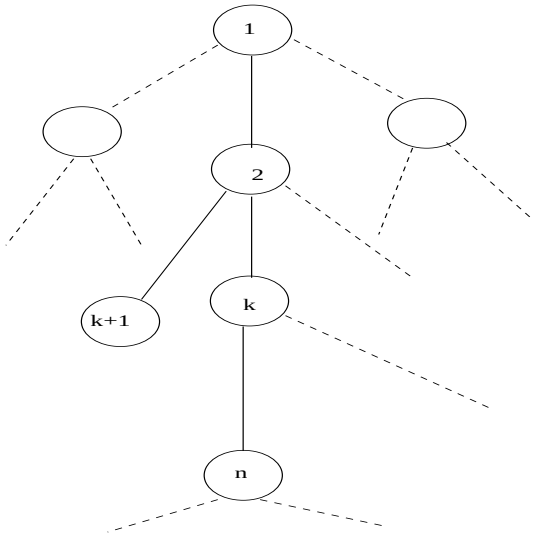


Figure 5.7: **A tree graph for matrix** B_T^n

as the basis of the triangular matrix as follows:

$$B_T^n = \begin{bmatrix} a_{1,1} & a_{1,2} & . & 0 & 0 & 0 \\ . & a_{2,2} & . & a_{2,k} & a_{2,k+1} & 0 \\ . & . & . & . & . & . \\ . & . & . & a_{k,k} & . & a_{k,n} \\ . & . & . & 0 & a_{k+1,k+1} & . \\ 0 & 0 & . & 0 & 0 & a_{n,n} \end{bmatrix} \tag{5.26}$$

The graph of the basis triangular matrix B_T^n is given in Figure 5.7. Note: B_T^n is non-singular since its diagonal entries are all non-zeros, hence the determinant is non-zero.

5.4 A New VE Algorithm for Dual *LI(2)*

Suppose we have a system of the form:

$$Ax = b, \quad x \geq 0 \tag{5.27}$$

where A is $m \times n$ matrix with no more than 2 nonzeros per column. The graph associated with (5.27) has:

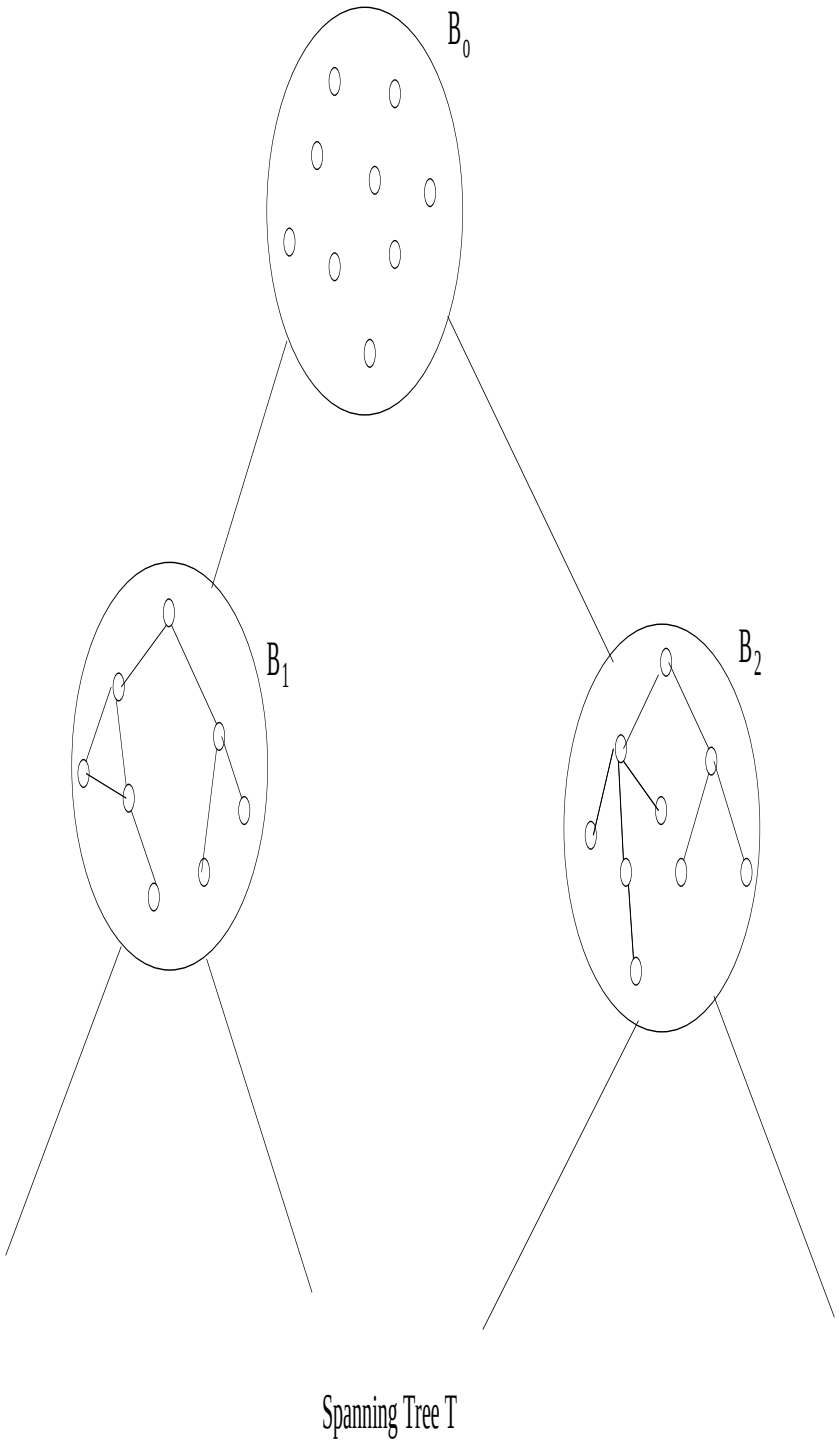


Figure 5.8: **Spanning Tree whose nodes are either a (i) tree, or (ii) subgraph with one cycle**

- m nodes, labelled R_1, R_2,...., R_m, each associated with a row of A;
- n (undirected) edges, labelled E_1, E_2,....,E_n, each associated with a column of A;
- Each edge E_j connecting nodes R_i, R_k if $a_{ij} \neq 0$ and $a_{kj} \neq 0$ or forming a loop at node R_i if $a_{ij} \neq 0$ and $a_{kj} = 0$, $\forall k$.

The algorithm enumerates vertices of the polyhedron associated with (5.27) in a similar fashion to that of Dyer [24] for more general polyhedra. The principal differences are:

1. vertices are generated via operations on the basis graph, rather than by simplex transformations;
2. the adjacency test is performed via a hash table rather than via an AVL tree;
3. the spanning tree of the feasible-basis graph of the polyhedron is constructed breadth-first rather than depth-first.

The algorithm as described here is valid for simple bounded polyhedra. Modification of the algorithm to handle unbounded and degenerate polyhedra are discussed later in the chapter. 'gamma sets' which record the set of edges that may lead from a vertex to previously unknown vertices, are again used to control the enumeration. The algorithm **EnumerateVertices** can now be outlined below. The description uses a_j to represent the jth column of A, B to represent a basis and β to represent its index set, $G(B)$ to represent the graph corresponding to B and V to represent a vertex of (5.27). **EnumerateVertices** uses a subsidiary routine **SOLVE**, whose purpose is to solve sets of linear equations via the components of the basis graph. **SOLVE** is described separately.

EnumerateVertices

1. Compute a basic feasible solution to (5.27), by LP if necessary. Let,

$$\beta_1 \leftarrow \{\text{indices of basic edges}\}, \quad \gamma_1 \leftarrow \{\text{indices of non} - \text{basic edges}\}$$

$$L \leftarrow \{(\beta_1, \gamma_1)\}, \quad p \leftarrow 1, \quad r \leftarrow 1$$

2. Construct $G(B_r)$ from β_r and determine its components

3. Determine V_r using **SOLVE**$(G(B_r), b, x)$ and output it

4. $\forall j \in \gamma_r$,

   determine a'_j using **SOLVE**$(G(B_r), a_j, a'_j)$

   Perform the simplex ratio test, so that if $k = argmin\{\frac{V_{ri}}{a'_{ji}} : a'_{ji} > 0\}$,

   the basic edge x_k leaves basis B_r

   $\beta \leftarrow \beta_r \cup \{j\} - \{k\}$

   $\gamma \leftarrow \{1, 2, ..., n\} - (\beta \cup \{k\})$

   $\gamma_r \leftarrow \gamma_r - \{j\}$

   If $\exists t \in \{1, 2, ..., p\}$ such that $\beta \equiv \beta_t$, $\gamma_t \leftarrow \gamma_t - \{k\}$

   else $p \leftarrow p + 1, \quad L \leftarrow L \cup \{(\beta, \gamma)\}$

5. $r \leftarrow r + 1$

   If $r \leq p$, goto 2

   Stop. $\square$

Hashing is used within Step 4 to test equivalence of bases and can be implemented as for Provan's algorithm, using an integer encoding of the basis index set. The advantage of using breadth-first search is that, for each basis, the components of its associated graph need be determined once only.

SOLVE(G, w, x)

Let G consist of nodes $i = 1, 2, ...v$ labelled with w_i and edges (i, j, k), $i \leq j$, associated with x_k and labelled with (a_{ik}, a_{jk}).
For each component of G:

(a) if the component is a simple loop comprising edge (i, i, k), $x_k \leftarrow w_i / a_{ik}$

(b) if the component is a tree

repeat until all tree edges deleted

if i is a leaf node with incident edge (i, j, k) or (j, i, k)

$x_k \leftarrow w_i/a_{ik}, \quad w_j \leftarrow w_j - a_{jk}x_k$

delete incident edge

(c) if the component contains a cycle,

1. remove nodes of degree 1 as in (b)
2. if a simple loop remains, apply (a)
3. if a cycle $(i_1, i_2, k_1), (i_2, i_3, k_2), ..., (i_{t-1}, i_t, k_{t-1})$, where $i_t = i_1$, remains, solve parametrically for $x_{k_1}, x_{k_2}, ..., x_{k_{t-1}}$ as follows:

   $x_{k_1} \leftarrow \lambda$

   for s = 2,t

   $w_{i_s} \leftarrow w_{i_s} - a_{i_s k_{s-1}} x_{k_{s-1}}$

   $x_{k_{s-1}} \leftarrow w_{i_s}/a_{i_s k_{s-1}}$

   Solve $x_{k_t} = \lambda$ for λ, and hence determine $x_{k_2}, ..., x_{k_{t-1}}$. □

5.4.1 An illustrative example

In this subsection we present an example to visualize how our new algorithm works. For simplicity, simple lookup, rather than hashing, is used to test basis equivalence and, at each vertex, details of how **SOLVE** operates are displayed only for finding the vertex coordinates.

Problem: Find all the vertices of the following system of 4×6 inequalities using **EnumerateVertices**.

$$2x_1 + 0x_2 - 2x_3 + 0x_4 + x_5 + 0x_6 \leq 5 \tag{5.28}$$

$$0x_1 - x_2 + 2x_3 + 0x_4 + 0x_5 + 3x_6 \leq 6 \tag{5.29}$$

$$-x_1 + 2x_2 + 0x_3 - 3x_4 + 0x_5 + 0x_6 \leq 8 \tag{5.30}$$

$$0x_1 + 0x_2 + 0x_3 + x_4 + 2x_5 + x_6 \leq 4 \tag{5.31}$$

where $x_i \geq 0$ for $i = 1, 2, ..., 6$. If we add slack variables to (5.28) to (5.31) we have:

$$2x_1 + 0x_2 - 2x_3 + 0x_4 + x_5 + 0x_6 + x_7 + 0x_8 + 0x_9 + 0x_{10} = 5 \tag{5.32}$$

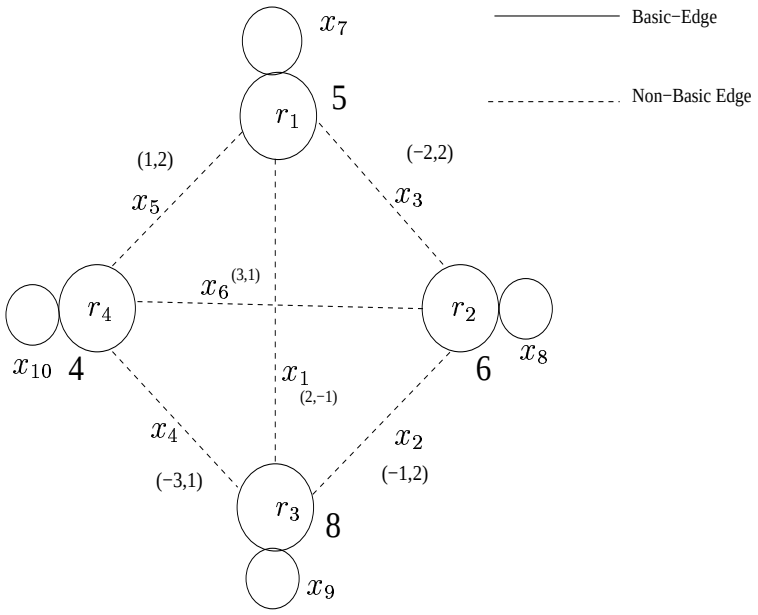


Figure 5.9: **The graph of the initial vertex**

$$0x_1 - x_2 + 2x_3 + 0x_4 + 0x_5 + 3x_6 + 0x_7 + x_8 + 0x_9 + 0x_{10} = 6 \tag{5.33}$$

$$-x_1 + 2x_2 + 0x_3 - 3x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + x_9 + 0x_{10} = 8 \tag{5.34}$$

$$0x_1 + 0x_2 + 0x_3 + x_4 + 2x_5 + x_6 + 0x_7 + 0x_8 + 0x_9 + x_{10} = 4 \tag{5.35}$$

In matrix form, (5.32) to (5.35) can be written $Ax = b$ where:

$$A = \begin{bmatrix} 2 & 0 & -2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & 3 & 0 & 1 & 0 & 0 \\ -1 & 2 & 0 & -3 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.36}$$

$$x^T = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}] \tag{5.37}$$

$$b^T = [5, 6, 8, 4] \tag{5.38}$$

For this example, the 4×4 identity matrix provides an initial basis B_1. Hence

$$\beta_1 = \{7, 8, 9, 10\}, \quad \gamma_1 \leftarrow \{1, 2, 3, 4, 5, 6\}, p \leftarrow 1, \quad r \leftarrow 1$$

We illustrate B_1 with aid of Figure 5.9; $G(B_1)$ consists of four components that are simple loops. Hence SOLVE step (a) gives $x_7 = 5$, $x_8 = 6$, $x_9 = 8$, $x_{10} = 4$.

The first vertex V_1 is:

$$V_1 : \begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 8 \\ 4 \end{bmatrix} \tag{5.39}$$

Step 4 gives, via SOLVE,

$$a_1' = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} a_2' = \begin{bmatrix} 0 \\ -1 \\ 2 \\ 0 \end{bmatrix} a_3' = \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix} a_4' = \begin{bmatrix} 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} a_5' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix} a_6' = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

Performing simplex pivots from V_1 on each of these in turn gives:

$\beta_2 \leftarrow \beta_1 \cup \{1\} - \{7\} = \{1, 8, 9, 10\}$

$\gamma_2 \leftarrow \{2, 3, 4, 5, 6\} \quad \gamma_1 \leftarrow \gamma_1 - \{1\} = \{2, 3, 4, 5, 6\}$

$\beta_3 \leftarrow \beta_1 \cup \{2\} - \{9\} = \{7, 8, 2, 10\}$

$\gamma_3 \leftarrow \{1, 3, 4, 5, 6\} \quad \gamma_1 \leftarrow \gamma_1 - \{2\} = \{3, 4, 5, 6\}$

$\beta_4 \leftarrow \beta_1 \cup \{3\} - \{8\} = \{7, 3, 9, 10\}$

$\gamma_4 \leftarrow \{1, 2, 4, 5, 6\} \quad \gamma_1 \leftarrow \gamma_1 - \{3\} = \{4, 5, 6\}$

$\beta_5 \leftarrow \beta_1 \cup \{4\} - \{10\} = \{7, 8, 9, 4\}$

$\gamma_5 \leftarrow \{1, 2, 3, 5, 6\} \quad \gamma_1 \leftarrow \gamma_1 - \{4\} = \{5, 6\}$

$\beta_6 \leftarrow \beta_1 \cup \{5\} - \{10\} = \{7, 8, 9, 5\}$

$\gamma_6 \leftarrow \{1, 2, 3, 4, 6\} \quad \gamma_1 \leftarrow \gamma_1 - \{5\} = \{6\}$

$\beta_7 \leftarrow \beta_1 \cup \{6\} - \{8\} = \{7, 6, 9, 10\}$

$\gamma_7 \leftarrow \{1, 2, 3, 4, 5\} \quad \gamma_1 \leftarrow \gamma_1 - \{6\} = \emptyset$

All β correspond to new BFS so we have: $p = 7, L = \{(\beta_1, \emptyset), (\beta_2, \gamma_2), .., (\beta_7, \gamma_7)\}$

Now $r = 2$, the graph of B_2 is illustrated in Figure 5.10 and consists of 3 components; two simple loops and one loop joined to an edge. SOLVE step (a) gives $x_8 = 6$ and $x_{10} = 4$ from the simple loops. SOLVE (c) gives $2x_1 = 5 \Rightarrow x_1 = \frac{5}{2}$ from node r_1, changes the label of r_3 to $8 + x_1 = \frac{21}{2}$ and the remaining simple loop at r_3 gives $x_9 = \frac{21}{2}$.

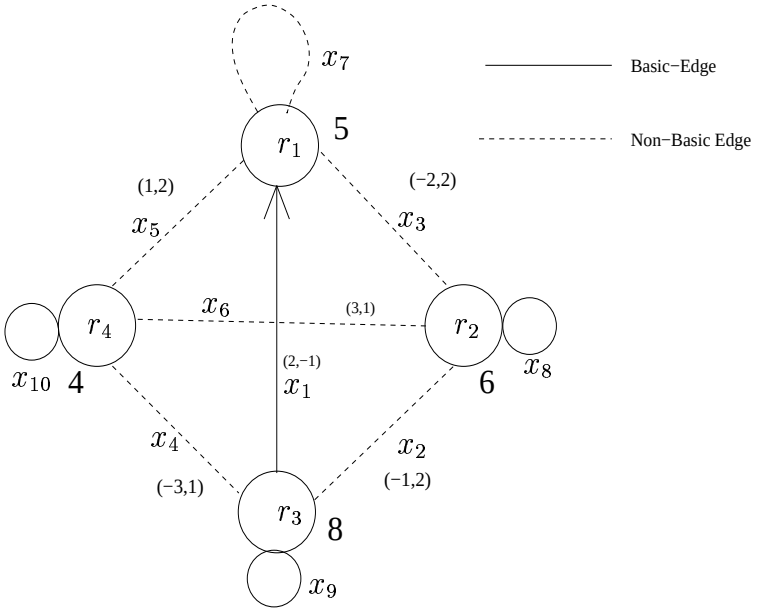


Figure 5.10: **The graph of vertex** 2

The second vertex is:

$$V_2 : \begin{bmatrix} x_1 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ 6 \\ \frac{21}{2} \\ 4 \end{bmatrix} \tag{5.40}$$

Step 4 gives, via SOLVE,

$$a_2^{'} = \begin{bmatrix} 0 \\ -1 \\ 2 \\ 0 \end{bmatrix} a_3^{'} = \begin{bmatrix} -1 \\ 2 \\ -1 \\ 0 \end{bmatrix} a_4^{'} = \begin{bmatrix} 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} a_5^{'} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 2 \end{bmatrix} a_6^{'} = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

Performing simplex pivots from V_2 on each of these in turn gives:

$\beta_8 \leftarrow \beta_2 \cup \{2\} - \{9\} = \{1, 8, 2, 10\}$

$\gamma_8 \leftarrow \{3, 4, 5, 6, 7\} \quad \gamma_2 \leftarrow \gamma_2 - \{2\} = \{3, 4, 5, 6\}$

$\beta_9 \leftarrow \beta_2 \cup \{3\} - \{8\} = \{1, 3, 9, 10\}$

$\gamma_9 \leftarrow \{2, 4, 5, 6, 7\} \quad \gamma_2 \leftarrow \gamma_2 - \{3\} = \{4, 5, 6\}$

$\beta_{10} \leftarrow \beta_2 \cup \{4\} - \{10\} = \{1, 8, 9, 4\}$

$\gamma_{10} \leftarrow \{2, 3, 5, 6, 7\} \quad \gamma_2 \leftarrow \gamma_2 - \{4\} = \{5, 6\}$

$\beta_{11} \leftarrow \beta_2 \cup \{5\} - \{10\} = \{1, 8, 9, 5\}$

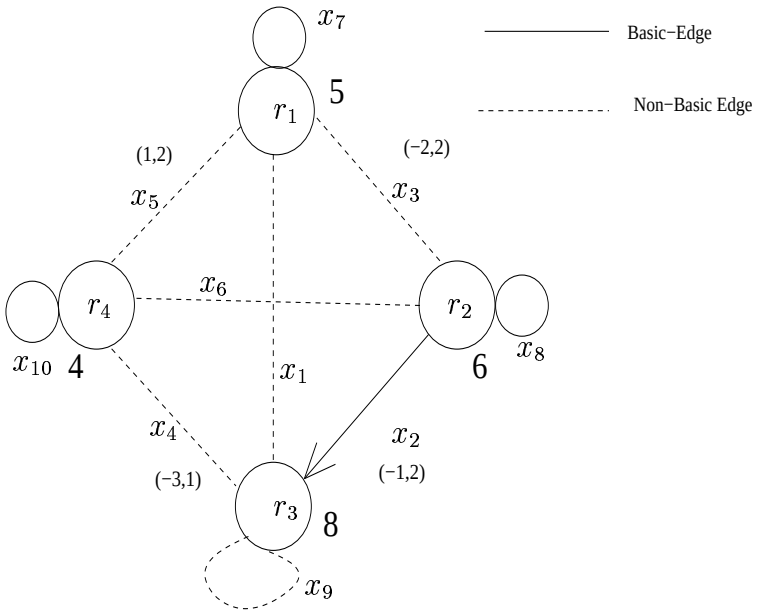


Figure 5.11: **The graph of vertex** 3

$\gamma_{11} \leftarrow \{2,3,4,6,7\} \quad \gamma_2 \leftarrow \gamma_2 - \{5\} = \{6\}$

$\beta_{12} \leftarrow \beta_2 \cup \{6\} - \{8\} = \{1,6,9,10\}$

$\gamma_{12} \leftarrow \{2,3,4,5,7\} \quad \gamma_2 \leftarrow \gamma_2 - \{6\} = \emptyset$

All β correspond to new BFS so we have: $p = 12, L = \{(\beta_1, \emptyset), (\beta_2, \emptyset), ..., (\beta_{12}, \gamma_{12})\}$

Now $r = 3$, the graph of B_3 is given in Figure 5.11 and consists of three components; two simple loops and one edge attached to another simple loop. SOLVE gives $x_7 = 5$ and $x_{10} = 4$ from step (a), step (c) gives $2x_2 = 8$ or $x_2 = 4$ from node r_3 and then $x_8 = x_2 + 6 = 10$ from node r_2.

The third vertex is:

$$V_3 : \begin{bmatrix} x_7 \\ x_8 \\ x_2 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 5 \\ 10 \\ 4 \\ 4 \end{bmatrix} \tag{5.41}$$

As $\gamma_3 = \{1,3,4,5,6\}$ we update a_1 using SOLVE to get:

$$a_1' = \begin{bmatrix} 2 \\ \frac{1}{2} \\ -\frac{1}{2} \\ 0 \end{bmatrix}$$

Simplex pivoting gives x_7 as the exiting variable to give $\beta = \{1,8,2,10\}$ which is identical to β_8 so no new vertex is found and

$$\gamma_3 \leftarrow \gamma_3 - \{1\} = \{3,4,5,6\}, \quad \gamma_8 \leftarrow \gamma_8 - \{7\} = \{3,4,5,6\}$$

Processing the rest of γ_3 through SOLVE gives:

$$a_3' = \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix} \quad a_4' = \begin{bmatrix} 0 \\ \frac{3}{2} \\ \frac{3}{2} \\ 1 \end{bmatrix} \quad a_5' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix} \quad a_6' = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

Performing simplex pivots from V_3 on each of these gives:

$\beta_{13} \leftarrow \beta_3 \cup \{3\} - \{8\} = \{7,3,2,10\}$

$\gamma_{13} \leftarrow \{1,4,5,6,8\} \quad \gamma_3 \leftarrow \gamma_3 - \{3\} = \{4,5,6\}$

$\beta_{14} \leftarrow \beta_3 \cup \{4\} - \{10\} = \{7,8,2,4\}$

$\gamma_{14} \leftarrow \{1,3,5,6,9\} \quad \gamma_3 \leftarrow \gamma_3 - \{4\} = \{5,6\}$

$\beta_{15} \leftarrow \beta_3 \cup \{5\} - \{10\} = \{7,8,2,5\}$

$\gamma_{15} \leftarrow \{1,3,4,6,9\} \quad \gamma_3 \leftarrow \gamma_3 - \{5\} = \{6\}$

$\beta_{16} \leftarrow \beta_3 \cup \{6\} - \{8\} = \{7,6,2,10\}$

$\gamma_{16} \leftarrow \{1,3,4,5,9\} \quad \gamma_3 \leftarrow \gamma_3 - \{6\} = \emptyset$

All these β correspond to new BFS, so we have: $p = 16$, $L = \{(\beta_1, \emptyset), .., (\beta_3, \emptyset), .., (\beta_{16}, \gamma_{16})\}$

Now $r = 4$, the graph of B_4 is given in Figure 5.12 which consists of three components; two simple loops and one loop joined to an edge. SOLVE step (a) gives: $x_9 = 8$ and $x_{10} = 4$. SOLVE (c) gives $2x_3 = 6$ or $x_3 = 3$ from node r_2, and $x_7 = 5 + 2x_3 = 11$.

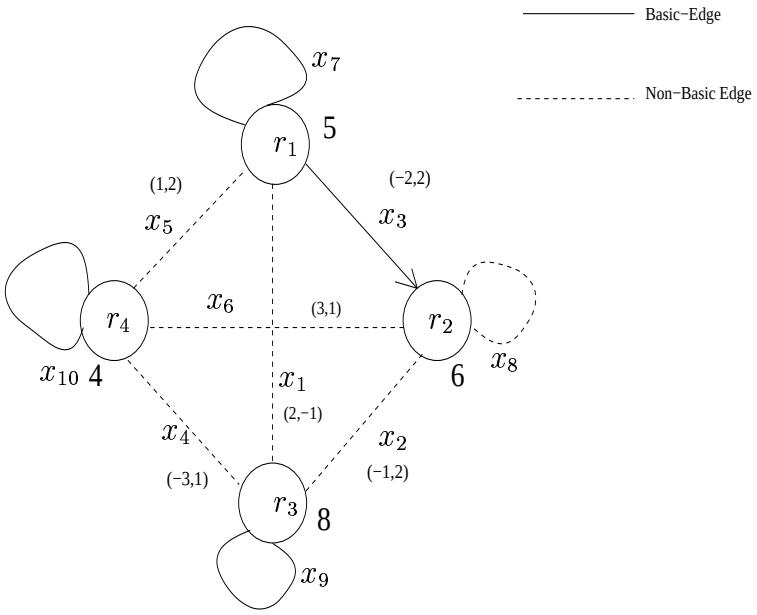


Figure 5.12: **The graph of vertex** 4

The 4^{th} vertex is:

$$V_4 : \begin{bmatrix} x_7 \\ x_3 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 11 \\ 3 \\ 8 \\ 4 \end{bmatrix} \tag{5.42}$$

As $\gamma_4 = \{1, 2, 4, 5, 6\}$ updating a_1, a_2 using SOLVE gives:

$$a_1^{'} = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} a_2^{'} = \begin{bmatrix} -1 \\ \frac{1}{2} \\ 2 \\ 0 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 3, 9, 10\}$ which is identical to β_9 so

$$\gamma_4 \leftarrow \gamma_4 - \{1\} = \{2, 4, 5, 6\}, \quad \gamma_9 \leftarrow \gamma_9 - \{7\} = \{2, 4, 5, 6\}$$

and x_2 for x_9 to give $\beta = \{7, 3, 2, 10\}$ which is identical to β_{13} so

$$\gamma_4 \leftarrow \gamma_4 - \{2\} = \{4, 5, 6\}, \quad \gamma_{13} \leftarrow \gamma_{13} - \{9\} = \{1, 4, 5, 6\}$$

SOLVE then gives:

$$a_4' = \begin{bmatrix} 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} \quad a_5' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

Performing simplex pivots from V_4 on each of these gives:

$\beta_{17} \leftarrow \beta_4 \cup \{4\} - \{10\} = \{7, 3, 9, 4\}$

$\gamma_{17} \leftarrow \{1, 2, 5, 6, 8\} \quad \gamma_4 \leftarrow \gamma_4 - \{4\} = \{5, 6\}$

$\beta_{18} \leftarrow \beta_4 \cup \{5\} - \{10\} = \{7, 3, 9, 5\}$

$\gamma_{18} \leftarrow \{1, 2, 4, 6, 8\} \quad \gamma_4 \leftarrow \gamma_4 - \{5\} = \{6\}$

As $\gamma_4 = \{6\}$ we update a_6 using SOLVE to get:

$$a_6' = \begin{bmatrix} 3 \\ \frac{3}{2} \\ 0 \\ 1 \end{bmatrix}$$

Simplex pivoting gives x_3 as the exiting variable to give $\beta = \{7, 6, 9, 10\}$ which is identical to β_7 so no new vertex is found and

$$\gamma_4 \leftarrow \gamma_4 - \{6\} = \emptyset, \quad \gamma_7 \leftarrow \gamma_7 - \{3\} = \{1, 2, 4, 5\}$$

We now have: $p = 18$, $L = \{(\beta_1, \emptyset), .., (\beta_7, \emptyset), (\beta_{18}, \gamma_{18})\}$

Now $r = 5$, the graph of B_5 is illustrated in Figure 5.13 which consists of three components; two simple loops and an edge joined to a loop. SOLVE step (a) gives $x_7 = 5$ and $x_8 = 6$ from the simple loops. SOLVE (c) gives $x_4 = 4$ from node r_4, and then $x_9 = 3x_4 + 8 = 20$.

The 5^{th} vertex is:

$$V_5 : \begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 20 \\ 4 \end{bmatrix} \tag{5.43}$$

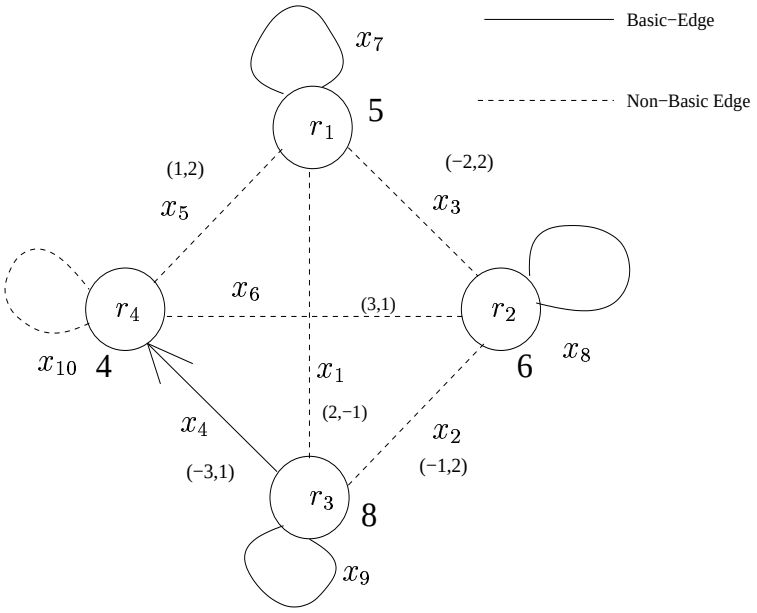


Figure 5.13: **The graph of vertex** 5

As $\gamma_5 = \{1, 2, 3, 5, 6\}$ we update a_1, a_2, a_3, a_5 using SOLVE to get:

$$a_1^{'} = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} \; a_2^{'} = \begin{bmatrix} 0 \\ -1 \\ 2 \\ 0 \end{bmatrix} \; a_3^{'} = \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix} \; a_5^{'} = \begin{bmatrix} 1 \\ 0 \\ 6 \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 8, 9, 4\}$ which is identical to β_{10} so no new vertex is found and

$$\gamma_5 \leftarrow \gamma_5 - \{1\} = \{2, 4, 5, 6\}, \qquad \gamma_{10} \leftarrow \gamma_{10} - \{7\} = \{2, 3, 5, 6\}$$

and x_2 for x_9 to give $\beta = \{7, 8, 2, 4\}$ which is identical to β_{14} so no new vertex is found and

$$\gamma_5 \leftarrow \gamma_5 - \{2\} = \{3, 5, 6\}, \qquad \gamma_{14} \leftarrow \gamma_{14} - \{9\} = \{1, 3, , 5, 6\}$$

and x_3 for x_8 to give $\beta = \{7, 3, 9, 4\}$ which is identical to β_{17} so no new vertex is found and

$$\gamma_5 \leftarrow \gamma_5 - \{3\} = \{5, 6\}, \qquad \gamma_{17} \leftarrow \gamma_{17} - \{8\} = \{1, 2, 5, 6\}$$

and x_5 for x_4 to give $\beta = \{7, 8, 9, 5\}$ which is identical to β_6 so no new vertex is found and

$$\gamma_5 \leftarrow \gamma_5 - \{5\} = \{6\}, \quad \gamma_6 \leftarrow \gamma_6 - \{4\} = \{1, 2, 3, 6\}$$

Step 4 gives through SOLVE,

$$a_6^{'} = \begin{bmatrix} 0 \\ 3 \\ 3 \\ 1 \end{bmatrix}$$

Performing a simplex pivot from V_5 on this gives:

$\beta_{19} \leftarrow \beta_5 \cup \{6\} - \{8\} = \{7, 6, 9, 4\}$

$\gamma_{19} \leftarrow \{1, 2, 3, 5, 10\} \quad \gamma_5 \leftarrow \gamma_5 - \{6\} = \emptyset$

We have: $p = 19$, $L = \{(\beta_1, \emptyset), .., (\beta_5, \emptyset), ..., (\beta_{19}, \gamma_{19})\}$

Now $r = 6$, the graph of B_6 is illustrated in Figure 5.14 which consists of three components; two simple loops and an edge joined to a loop. SOLVE step (a) gives $x_8 = 6$ and $x_9 = 8$ from the simple loops. SOLVE (c) gives $2x_5 = 4$ or $x_5 = 2$ from node r_4, and then $x_7 = 5 - x_5 = 3$.

The 6^{th} vertex is:

$$V_6 : \begin{bmatrix} x_7 \\ x_8 \\ x_9 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 8 \\ 2 \end{bmatrix} \tag{5.44}$$

As $\gamma_6 = \{1, 2, 3, 6\}$ we update a_1, a_2, a_3 using SOLVE to get:

$$a_1^{'} = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} \quad a_2^{'} = \begin{bmatrix} 0 \\ -1 \\ 2 \\ 0 \end{bmatrix} \quad a_3^{'} = \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 8, 9, 5\}$ which is identical to

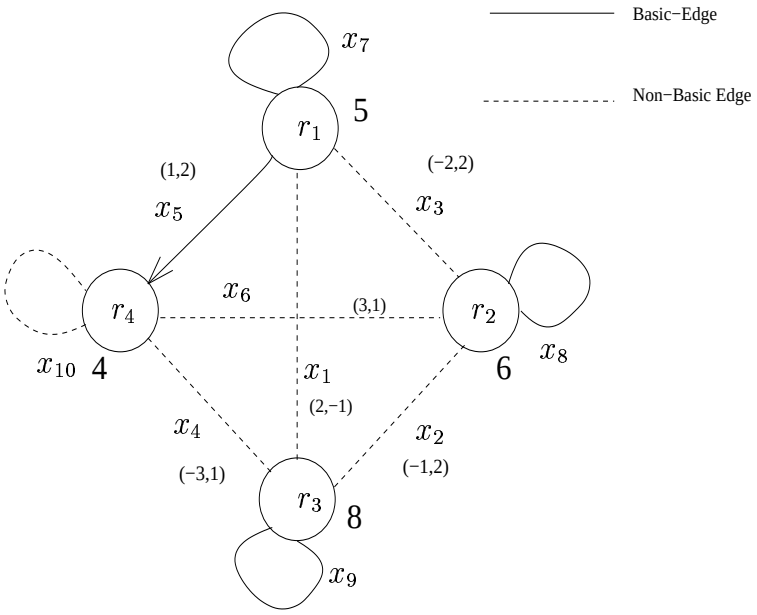


Figure 5.14: **The graph of vertex** 6

β_{11} so no new vertex is found and

$$\gamma_6 \leftarrow \gamma_6 - \{1\} = \{2,3,6\}, \qquad \gamma_{11} \leftarrow \gamma_{11} - \{7\} = \{2,3,4,6\}$$

and x_2 for x_9 to give $\beta = \{7,8,2,5\}$ which is identical to β_{15} so no new vertex is found and

$$\gamma_6 \leftarrow \gamma_6 - \{2\} = \{3,6\}, \qquad \gamma_{15} \leftarrow \gamma_{15} - \{9\} = \{1,3,4,6\}$$

and x_3 for x_8 to give $\beta = \{7,3,9,5\}$ which is identical to β_{18} so no new vertex is found and

$$\gamma_6 \leftarrow \gamma_6 - \{3\} = \{6\}, \qquad \gamma_{18} \leftarrow \gamma_{18} - \{8\} = \{1,2,4,6\}$$

Step 4 gives through SOLVE,

$$a_6^{'} = \begin{bmatrix} -\frac{1}{2} \\ 3 \\ 0 \\ \frac{1}{2} \end{bmatrix}$$

Performing a simplex pivot from V_6 on this gives:

$\beta_{20} \leftarrow \beta_6 \cup \{6\} - \{8\} = \{7,6,9,5\}$

$\gamma_{20} \leftarrow \{1,2,3,4,10\} \qquad \gamma_6 \leftarrow \gamma_6 - \{6\} = \emptyset$

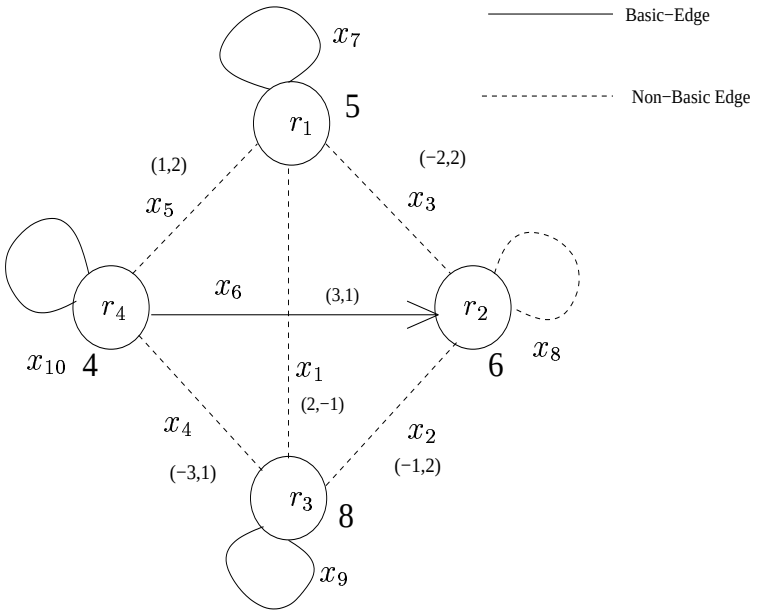


Figure 5.15: **The graph of vertex** 7

We have: $p = 20$, $L = \{(\beta_1, \emptyset), .., (\beta_6, \emptyset), ..., (\beta_{20}, \gamma_{20})\}$.

Now $r = 7$, the graph of B_7 is illustrated in Figure 5.15 which consists of three components; two simple loops and an edge attached to a loop. SOLVE step (a) gives $x_7 = 5$ and $x_9 = 8$ from the simple loops. SOLVE (c) gives $3x_6 = 6$ or $x_6 = 2$ from node r_2, and then $x_{10} = 4 - x_6 = 2$.

The 7^{th} vertex is:

$$V_7 : \begin{bmatrix} x_7 \\ x_6 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 8 \\ 2 \end{bmatrix} \tag{5.45}$$

As $\gamma_7 = \{1, 2, 4, 5\}$ we update a_1, a_2, a_4, a_5 using SOLVE to get:

$$a_1^{'} = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} \; a_2^{'} = \begin{bmatrix} 0 \\ \frac{-1}{3} \\ 2 \\ \frac{1}{3} \end{bmatrix} \; a_4^{'} = \begin{bmatrix} 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} \; a_5^{'} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 6, 9, 10\}$ which is identical to β_{12} so no new vertex is found and

$$\gamma_7 \leftarrow \gamma_7 - \{1\} = \{2, 4, 5\}, \quad \gamma_{12} \leftarrow \gamma_{12} - \{7\} = \{2, 3, 4, 5\}$$

and x_2 for x_9 to give $\beta = \{7, 6, 2, 10\}$ which is identical to β_{16} so no new vertex is found and

$$\gamma_7 \leftarrow \gamma_7 - \{2\} = \{4, 5\}, \quad \gamma_{16} \leftarrow \gamma_{16} - \{9\} = \{1, 3, 4, 5\}$$

and x_4 for x_{10} to give $\beta = \{7, 6, 9, 4\}$ which is identical to β_{19} so no new vertex is found and

$$\gamma_7 \leftarrow \gamma_7 - \{4\} = \{5\}, \quad \gamma_{19} \leftarrow \gamma_{19} - \{10\} = \{1, 2, 3, 5\}$$

and x_5 for x_{10} to give $\beta = \{7, 6, 9, 5\}$ which is identical to β_{20} so no new vertex is found and

$$\gamma_7 \leftarrow \gamma_7 - \{5\} = \emptyset, \quad \gamma_{20} \leftarrow \gamma_{20} - \{10\} = \{1, 2, 3, 4\}$$

There is no new vertex adjacent to V_7; p remains 20.

Now $r = 8$, the graph of B_8 is illustrated in Figure 5.16 which consists of two components; one simple loop and loop joined to two adjacent edges. SOLVE step (a) gives $x_{10} = 4$ from the simple loop. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1, then $x_2 = \frac{8+x_1}{2} = \frac{21}{4}$ and $x_8 = x_2 + 6 = \frac{45}{4}$.

So vertex 8 is:

$$V_8 : \begin{bmatrix} x_1 \\ x_8 \\ x_2 \\ x_{10} \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{45}{4} \\ \frac{21}{4} \\ 4 \end{bmatrix} \tag{5.46}$$

Step 4 gives via SOLVE,

$$a_3' = \begin{bmatrix} -1 \\ \frac{3}{2} \\ -\frac{1}{2} \\ 0 \end{bmatrix} \; a_4' = \begin{bmatrix} 0 \\ -\frac{3}{2} \\ -\frac{3}{2} \\ 1 \end{bmatrix} \; a_5' = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{4} \\ \frac{1}{4} \\ 2 \end{bmatrix} \; a_6' = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

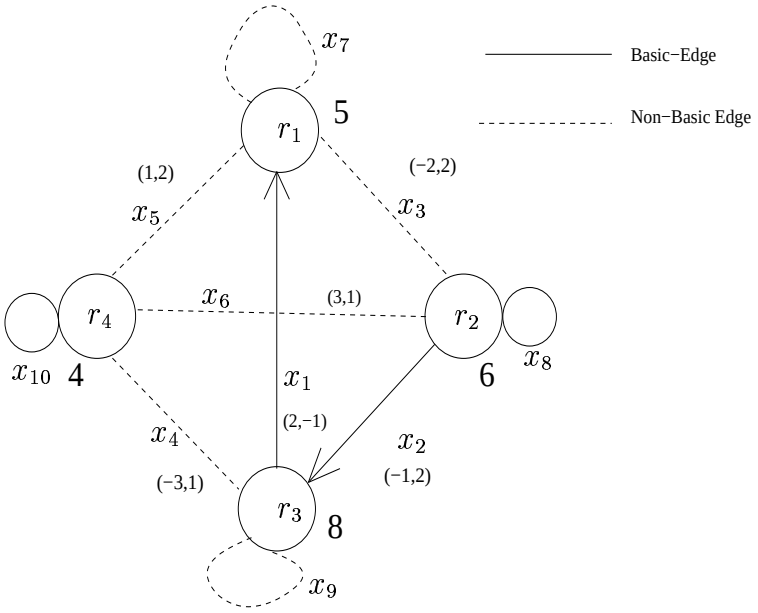


Figure 5.16: **The graph of vertex** 8

Performing simplex pivots from V_8 on each of these gives:

$\beta_{21} \leftarrow \beta_8 \cup \{3\} - \{8\} = \{1, 3, 2, 10\}$

$\gamma_{21} \leftarrow \{4, 5, 6, 7, 9\} \quad \gamma_8 \leftarrow \gamma_8 - \{3\} = \{4, 5, 6\}$

$\beta_{22} \leftarrow \beta_8 \cup \{4\} - \{10\} = \{1, 8, 2, 4\}$

$\gamma_{22} \leftarrow \{3, 5, 6, 7, 9\} \quad \gamma_8 \leftarrow \gamma_8 - \{4\} = \{5, 6\}$

$\beta_{23} \leftarrow \beta_8 \cup \{5\} - \{10\} = \{1, 8, 2, 5\}$

$\gamma_{23} \leftarrow \{3, 4, 6, 7, 9\} \quad \gamma_8 \leftarrow \gamma_8 - \{5\} = \{6\}$

$\beta_{24} \leftarrow \beta_8 \cup \{6\} - \{8\} = \{1, 6, 2, 10\}$

$\gamma_{24} \leftarrow \{3, 4, 5, 7, 9\} \quad \gamma_8 \leftarrow \gamma_8 - \{6\} = \emptyset$

All β correspond to new BFS, so we have: $p = 24, L = \{(\beta_1, \emptyset), .., (\beta_8, \emptyset), ..(\beta_{24}, \gamma_{24})\}$

Now $r = 9$, the graph of B_9 is illustrated in Figure 5.17 which consists of two components; one simple loop and loop joined to two adjacent edges. SOLVE step (a) gives $x_{10} = 4$ from the simple loop. SOLVE (c) gives $2x_3 = 6$ or $x_3 = 3$ from node r_2, then $x_1 = \frac{5+2x_3}{2} = \frac{11}{2}$ and $x_9 = x_1 + 8 = \frac{27}{2}$.

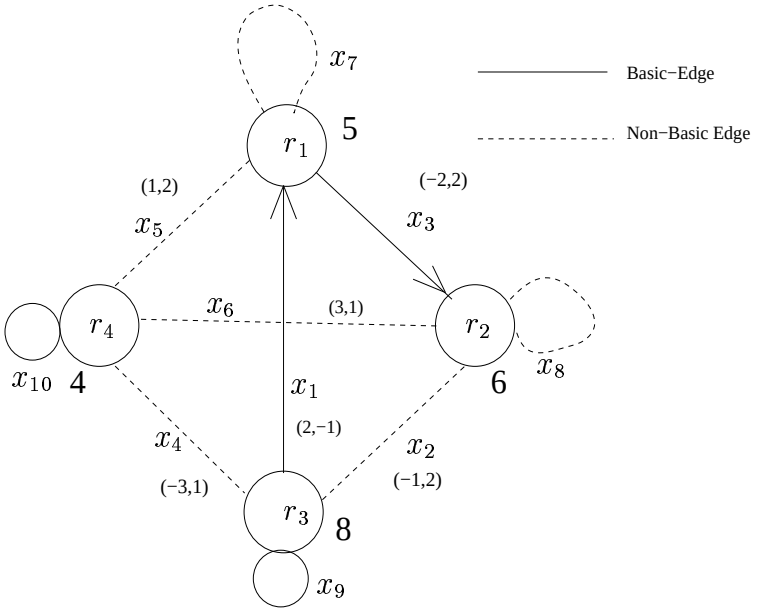


Figure 5.17: **The graph of vertex** 9

The 9^{th} vertex is:

$$V_9 : \begin{bmatrix} x_1 \\ x_3 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} \frac{11}{2} \\ 3 \\ \frac{27}{2} \\ 4 \end{bmatrix} \tag{5.47}$$

As $\gamma_9 = \{2, 4, 5, 6\}$ we update a_2 using SOLVE to get:

$$a_2^{'} = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{3}{2} \\ 0 \end{bmatrix}$$

Simplex pivoting gives x_9 as the exiting variable to give $\beta = \{1, 3, 2, 10\}$ which is identical to β_{21} so no new vertex is found and

$$\gamma_9 \leftarrow \gamma_9 - \{2\} = \{4, 5, 6\}, \qquad \gamma_{21} \leftarrow \gamma_{21} - \{9\} = \{4, 5, 6, 7\}$$

Step 4 gives via SOLVE,

$$a_4^{'} = \begin{bmatrix} 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} \quad a_5^{'} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 2 \end{bmatrix}$$

Performing simplex pivots from V_9 on each of these gives:

$\beta_{25} \leftarrow \beta_9 \cup \{4\} - \{10\} = \{1, 3, 9, 4\}$

$\gamma_{25} \leftarrow \{2, 5, 6, 7, 8\} \quad \gamma_9 \leftarrow \gamma_9 - \{4\} = \{5, 6\}$

$\beta_{26} \leftarrow \beta_9 \cup \{5\} - \{10\} = \{1, 3, 9, 5\}$

$\gamma_{26} \leftarrow \{2, 4, 6, 7, 8\} \quad \gamma_9 \leftarrow \gamma_9 - \{5\} = \{6\}$

We update a_6 using SOLVE to give:

$$a_6^{'} = \begin{bmatrix} \frac{3}{2} \\ \frac{3}{2} \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

Simplex pivoting gives x_3 as the exiting variable to give $\beta = \{1, 6, 9, 10\}$ which is identical to β_{12} so no new vertex is found and

$$\gamma_9 \leftarrow \gamma_9 - \{6\} = \emptyset, \quad \gamma_{12} \leftarrow \gamma_{12} - \{3\} = \{2, 4, 5\}$$

We have: $p = 26$, $L = \{(\beta_1, \emptyset), .., (\beta_9, \emptyset), ..(\beta_{26}, \gamma_{26})\}$

Now $r = 10$, the graph of B_{10} is illustrated in Figure 5.18 which consists of two components; one simple loop and a loop joined to two adjacent edges. SOLVE step (a) gives $x_8 = 6$ from the simple loop. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1, then $x_4 = 4$ from node r_4 and $x_9 = x_1 + 3x_4 + 8 = \frac{45}{2}$ from node r_3.

Vertex 10 is:

$$V_{10} : \begin{bmatrix} x_1 \\ x_8 \\ x_9 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ 6 \\ \frac{45}{2} \\ 4 \end{bmatrix} \tag{5.48}$$

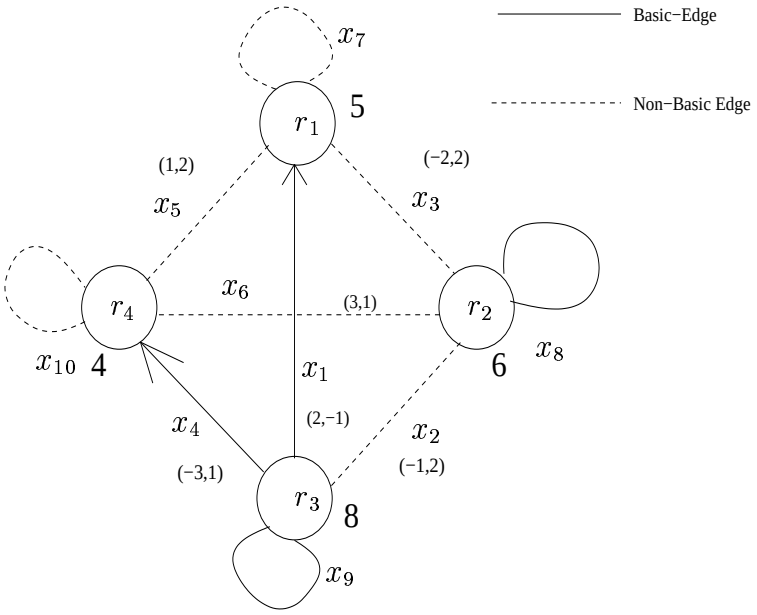


Figure 5.18: **The graph of vertex** 10

As $\gamma_{10} = \{2, 3, 5, 6\}$ we update $a_2, a_(3), a_(5)$ using SOLVE to get:

$$a_2^{'} = \begin{bmatrix} 0 \\ -1 \\ 2 \\ 0 \end{bmatrix} \; a_3^{'} = \begin{bmatrix} -1 \\ 2 \\ -1 \\ 0 \end{bmatrix} \; a_5^{'} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{13}{2} \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_2 for x_9 to give $\beta = \{1, 8, 2, 4\}$ which is identical to β_{22} so no new vertex is found and

$$\gamma_{10} \leftarrow \gamma_{10} - \{2\} = \{3, 5, 6\}, \quad \gamma_{22} \leftarrow \gamma_{22} - \{9\} = \{3, 5, 6, 7\}$$

and x_3 for x_8 to give $\beta = \{1, 3, 9, 4\}$ which is identical to β_{25} so no new vertex is found and

$$\gamma_{10} \leftarrow \gamma_{10} - \{3\} = \{5, 6\}, \quad \gamma_{25} \leftarrow \gamma_{25} - \{8\} = \{2, 5, 6, 7\}$$

and x_5 for x_4 to give $\beta = \{1, 8, 9, 5\}$ which is identical to β_{11} so no new vertex is found and

$$\gamma_{10} \leftarrow \gamma_{10} - \{5\} = \{6\}, \quad \gamma_{11} \leftarrow \gamma_{11} - \{4\} = \{2, 3, 6\}$$

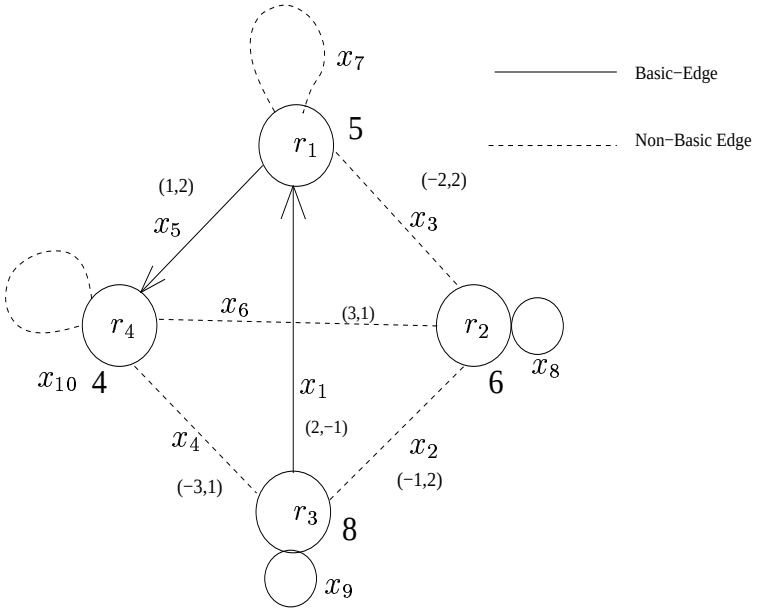


Figure 5.19: **The graph of vertex** 11

Step 4 gives via SOLVE,

$$a_6^{'} = \begin{bmatrix} 0 \\ 3 \\ 3 \\ 1 \end{bmatrix}$$

Performing a simplex pivot from V_{10} gives:

$\beta_{27} \leftarrow \beta_{10} \cup \{6\} - \{8\} = \{1, 6, 9, 4\}$

$\gamma_{27} \leftarrow \{2, 3, 5, 7, 10\}$ $\quad \gamma_{10} \leftarrow \gamma_{10} - \{6\} = \emptyset$

We have: $p = 27$, $L = \{(\beta_1, \emptyset), .., (\beta_{10}, \emptyset), ..(\beta_{27}, \gamma_{27})\}$

Now $r = 11$, the graph of B_{11} is illustrated in Figure 5.19 which consists of two components; one simple loop and loop joined to two adjacent edges. SOLVE step (a) gives $x_8 = 6$ from the simple loop. SOLVE (c) gives $2x_5 = 4$ or $x_5 = 2$ from node r_4, then $2x_1 + x_5 = 5$ or $x_1 = \frac{5-x_5}{2} = \frac{3}{2}$ from node r_1 and $x_9 = x_1 + 8 = \frac{19}{2}$.

Vertex 11 is:

$$V_{11}: \begin{bmatrix} x_1 \\ x_8 \\ x_9 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{3}{2} \\ 6 \\ \frac{19}{2} \\ 2 \end{bmatrix} \tag{5.49}$$

As $\gamma_{11} = \{2, 3, 6\}$ we update a_2, a_3 using SOLVE to get:

$$a_2' = \begin{bmatrix} 0 \\ -1 \\ 2 \\ 0 \end{bmatrix} \quad a_3' = \begin{bmatrix} -1 \\ 2 \\ -1 \\ 0 \end{bmatrix}$$

Simplex pivoting exchanges x_2 for x_9 to give $\beta = \{1, 8, 2, 5\}$ which is identical to β_{23} so no new vertex is found and

$$\gamma_{11} \leftarrow \gamma_{11} - \{2\} = \{3, 6\}, \qquad \gamma_{23} \leftarrow \gamma_{23} - \{9\} = \{3, 4, 6, 7\}$$

and x_3 for x_8 to give $\beta = \{1, 3, 9, 5\}$ which is identical to β_{26} so no new vertex is found and

$$\gamma_{11} - \{3\} = \{6\}, \qquad \gamma_{26} \leftarrow \gamma_{26} - \{8\} = \{2, 4, 6, 7\}$$

Step 4 gives via SOLVE,

$$a_6' = \begin{bmatrix} -\frac{1}{4} \\ 3 \\ -\frac{1}{4} \\ \frac{1}{2} \end{bmatrix}$$

Performing a simplex pivot from V_{11} gives:

$\beta_{28} \leftarrow \beta_{11} \cup \{6\} - \{8\} = \{1, 6, 9, 5\}$

$\gamma_{28} \leftarrow \{2, 3, 4, 7, 10\} \quad \gamma_{11} \leftarrow \gamma_{11} - \{6\} = \emptyset$

We have: $p = 28$, $L = \{(\beta_1, \emptyset), .., (\beta_{11}, \emptyset), ..(\beta_{28}, \gamma_{28})\}$

Now $r = 12$, the graph of B_{12} is illustrated in Figure 5.20 which consists of two components which are simple loops each joined to an edge. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1, $x_9 = 8 + x_1 = \frac{21}{2}$ from node r_3 then $3x_6 = 6$ or $x_6 = 2$ from

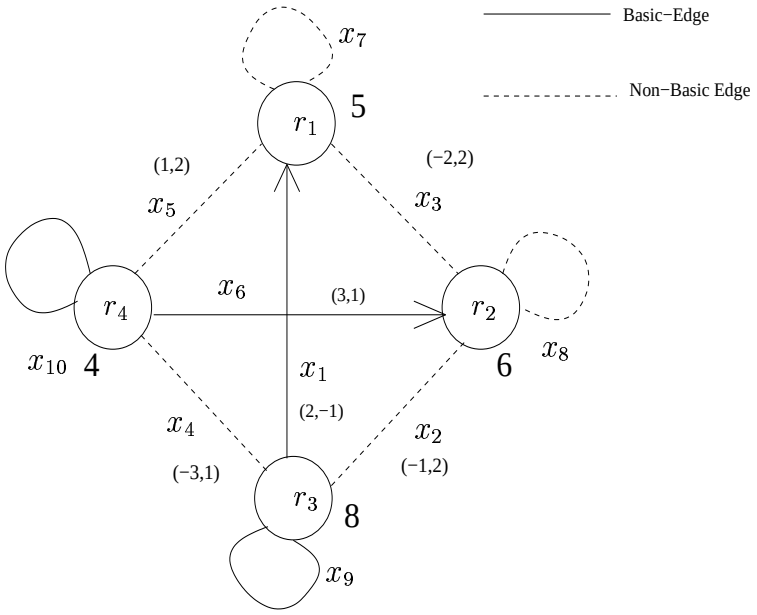


Figure 5.20: **The graph of vertex** 12

node r_2, $x_{10} = 4 - x_6 = 2$ from node r_4.

So vertex 12 is:

$$V_{12} : \begin{bmatrix} x_1 \\ x_6 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ 2 \\ \frac{21}{2} \\ 2 \end{bmatrix} \tag{5.50}$$

As $\gamma_{12} = \{2, 4, 5\}$ we update a_2, a_4, a_5 using SOLVE to get:

$$a_2^{'} = \begin{bmatrix} 0 \\ -\frac{1}{3} \\ 2 \\ \frac{1}{3} \end{bmatrix} \quad a_4^{'} = \begin{bmatrix} 0 \\ 0 \\ -3 \\ 1 \end{bmatrix} \quad a_5^{'} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_2 for x_9 to give $\beta = \{1, 6, 2, 10\}$ which is identical to β_{24} so no new vertex is found and

$$\gamma_{12} \leftarrow \gamma_{12} - \{2\} = \{4, 5\}, \qquad \gamma_{24} \leftarrow \gamma_{24} - \{9\} = \{3, 4, 5, 7\}$$

and x_3 for x_{10} to give $\beta = \{1, 6, 9, 4\}$ which is identical to β_{27} so no new vertex is

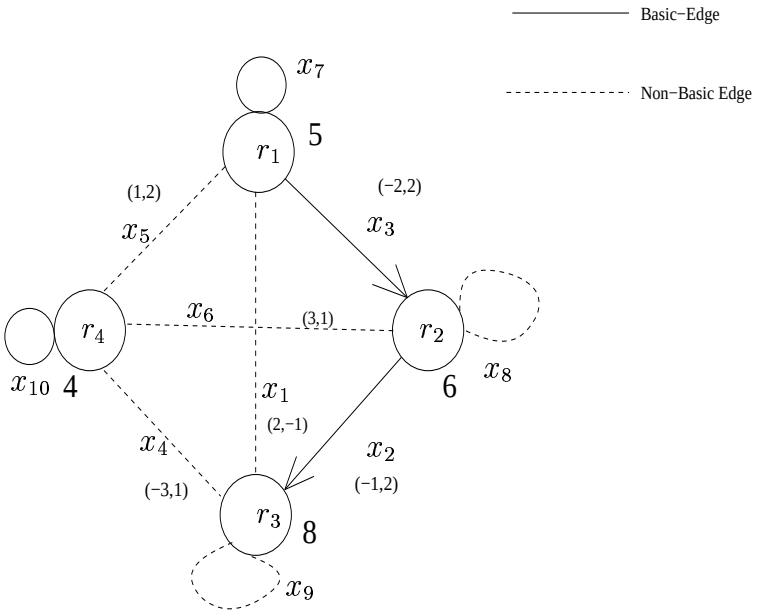


Figure 5.21: **The graph of vertex** 13

found and

$$\gamma_{12} \leftarrow \gamma_{12} - \{4\} = \{5\}, \qquad \gamma_{27} \leftarrow \gamma_{27} - \{10\} = \{2, 3, 5, 7\}$$

and x_5 for x_{10} to give $\beta = \{1, 6, 9, 5\}$ which is identical to β_{28} so no new vertex is found and

$$\gamma_{12} \leftarrow \gamma_{12} - \{5\} = \emptyset, \qquad \gamma_{28} \leftarrow \gamma_{28} - \{10\} = \{2, 3, 4, 7\}$$

There is no *new* vertex adjacent to V_{12}; p remains 28.

Now $r = 13$, the graph of B_{13} is illustrated in Figure 5.21 which consists of two components; one simple loop and a loop joined to two adjacent edges. SOLVE (a) gives $x_{10} = 4$ from node r_4. SOLVE (c) gives $2x_2 = 8$ or $x_2 = 4$ from node r_3 and $2x_3 = x_2 + 6$ or $x_3 = 5$ from node r_2, then $x_7 = 5 + 2x_2 = 15$ from node r_1.

Vertex 13 is:

$$V_{13} : \begin{bmatrix} x_7 \\ x_3 \\ x_2 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 15 \\ 5 \\ 4 \\ 4 \end{bmatrix} \tag{5.51}$$

As $\gamma_{13} = \{1, 4, 5, 6\}$ we update a_1 using SOLVE to get:

$$a_1^{'} = \begin{bmatrix} \frac{3}{2} \\ -\frac{1}{4} \\ -\frac{1}{2} \\ 0 \end{bmatrix}$$

Simplex pivoting gives x_7 as the exiting variable to give $\beta = \{1, 3, 2, 10\}$ which is identical to β_{21} so no new vertex is found and

$$\gamma_{13} \leftarrow \gamma_{13} - \{1\} = \{4, 5, 6\}, \quad \gamma_{21} \leftarrow \gamma_{21} - \{7\} = \{4, 5, 6\}$$

Step 4 gives via SOLVE,

$$a_4^{'} = \begin{bmatrix} -\frac{3}{2} \\ -\frac{3}{4} \\ -\frac{3}{2} \\ 1 \end{bmatrix} \quad a_5^{'} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

Performing simplex pivots from V_{13} on each of these gives:

$\beta_{29} \leftarrow \beta_{13} \cup \{4\} - \{10\} = \{7, 3, 2, 4\}$

$\gamma_{29} \leftarrow \{1, 5, 6, 8, 9\} \quad \gamma_{13} \leftarrow \gamma_{13} - \{4\} = \{5, 6\}$

$\beta_{30} \leftarrow \beta_{13} \cup \{5\} - \{10\} = \{7, 3, 2, 5\}$

$\gamma_{30} \leftarrow \{1, 4, 6, 8, 9\} \quad \gamma_{13} \leftarrow \gamma_{13} - \{5\} = \{6\}$

As $\gamma_{13} = \{6\}$ we update a_6 using SOLVE to get:

$$a_6^{'} = \begin{bmatrix} 3 \\ \frac{3}{2} \\ 0 \\ 1 \end{bmatrix}$$

Simplex pivoting gives x_3 as the exiting variable to give $\beta = \{7, 6, 2, 10\}$ which is identical to β_{16} so no new vertex is found and

$$\gamma_{13} \leftarrow \gamma_{13} - \{6\} = \emptyset, \quad \gamma_{16} \leftarrow \gamma_{16} - \{3\} = \{1, 4, 5\}$$

We have: $p = 30$, $L = \{(\beta_1, \emptyset), .., (\beta_{13}, \emptyset), ..(\beta_{30}, \gamma_{30})\}$

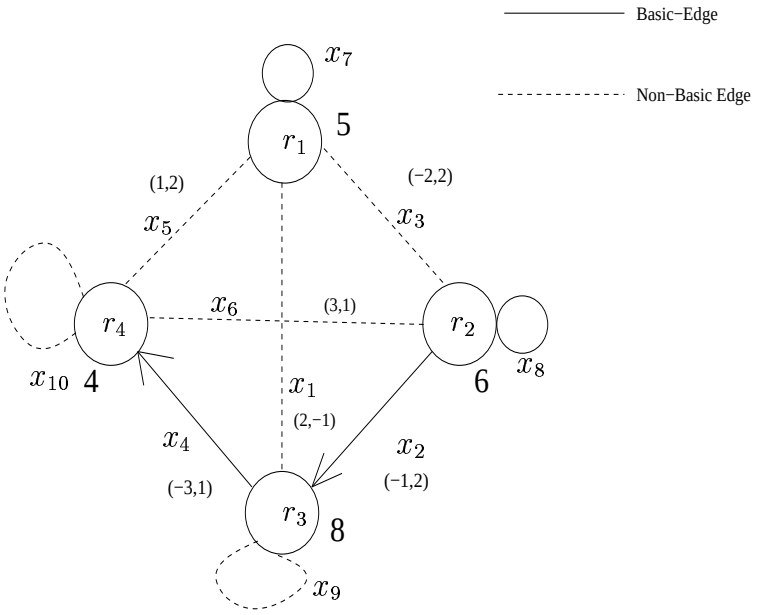


Figure 5.22: **The graph of vertex** 14

Now $r = 14$, the graph of B_{14} is illustrated in Figure 5.22 which consists of two components; one simple loop and loop joined to two adjacent edges. SOLVE step (a) gives $x_7 = 5$ from the simple loop. SOLVE (c) gives $x_4 = 4$ from node r_4, then $2x_8 - 3x_4 = 8$ or $x_2 = \frac{8+3x_4}{2} = 10$ from node r_3 and $x_8 = x_2 + 8 = 16$ from node r_2.

Vertex 14 is:

$$V_{14} : \begin{bmatrix} x_7 \\ x_8 \\ x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 16 \\ 10 \\ 4 \end{bmatrix} \tag{5.52}$$

As $\gamma_{14} = \{1, 3, 5, 6\}$ we update a_1, a_3, a_5 using SOLVE to get:

$$a_1^{'} = \begin{bmatrix} 2 \\ -\frac{1}{2} \\ -\frac{1}{2} \\ 0 \end{bmatrix} \quad a_3^{'} = \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix} \quad a_5^{'} = \begin{bmatrix} 1 \\ 3 \\ 3 \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 8, 2, 4\}$ which is identical to

β_{22} so no new vertex is found and

$$\gamma_{14} \leftarrow \gamma_{14} - \{1\} = \{3,5,6\}, \quad \gamma_{22} \leftarrow \gamma_{22} - \{7\} = \{3,5,6\}$$

and x_3 for x_8 to give $\beta = \{7,3,2,4\}$ which is identical to β_{29} so no new vertex is found and

$$\gamma_{14} \leftarrow \gamma_{14} - \{3\} = \{5,6\}, \quad \gamma_{29} \leftarrow \gamma_{29} - \{8\} = \{1,5,6,9\}$$

and x_5 for x_4 to give $\beta = \{7,8,2,5\}$ which is identical to β_{15} so no new vertex is found and

$$\gamma_{14} \leftarrow \gamma_{14} - \{5\} = \{6\}, \quad \gamma_{15} \leftarrow \gamma_{15} - \{4\} = \{1,3,6\}$$

Step 4 gives via SOLVE,

$$a_6' = \begin{bmatrix} 0 \\ \frac{9}{2} \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

Performing a simplex pivot from V_{14} on this gives:

$\beta_{31} \leftarrow \beta_{14} \cup \{6\} - \{8\} = \{7,6,2,4\}$

$\gamma_{31} \leftarrow \{1,3,5,9,10\} \quad \gamma_{14} \leftarrow \gamma_{14} - \{6\} = \emptyset$

We have: $p = 31$, $L = \{(\beta_1, \emptyset), .., (\beta_{14}, \emptyset), ..(\beta_{31}, \gamma_{31})\}$

Now $r = 15$, the graph of B_{15} is illustrated in Figure 5.23 which consists of two components; each a simple loop joined to an edge. SOLVE (c) gives $2x_5 = 4$ or $x_5 = 2$ from node r_4 and $x_7 = 5 - x_5 = 3$ from node r_1 then $2x_2 = 8$ or $x_2 = 4$ from node r_3 and $x_8 = x_2 + 6 = 10$ from node r_2.

Vertex 15 is:

$$V_{15} : \begin{bmatrix} x_7 \\ x_8 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 4 \\ 2 \end{bmatrix} \tag{5.53}$$

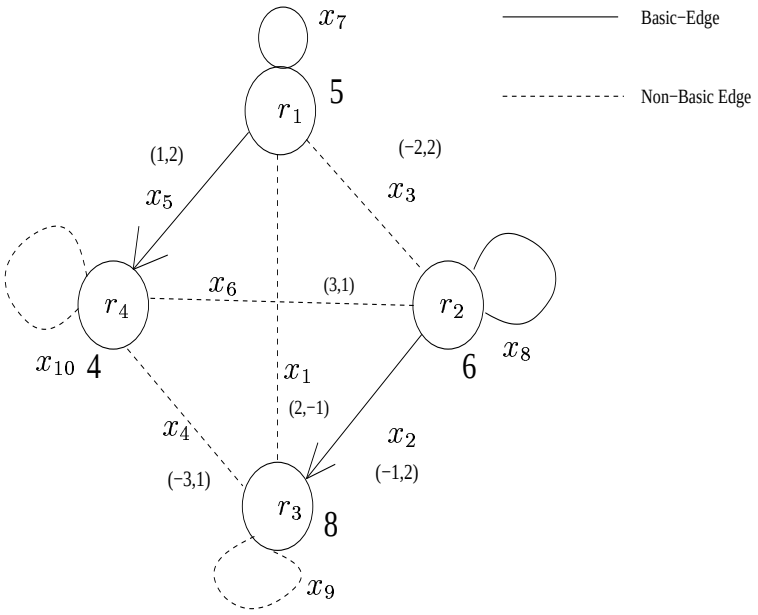


Figure 5.23: **The graph of vertex** 15

As $\gamma_{15} = \{1, 3, 6\}$ we update a_1, a_3 using SOLVE to get:

$$a_1' = \begin{bmatrix} 2 \\ -\frac{1}{2} \\ -\frac{1}{2} \\ 0 \end{bmatrix} \quad a_3' = \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 8, 2, 5\}$ which is identical to β_{23} so no new vertex is found and

$$\gamma_{15} \leftarrow \gamma_{15} - \{1\} = \{3, 6\}, \qquad \gamma_{23} \leftarrow \gamma_{23} - \{7\} = \{3, 4, 6\}$$

and x_3 for x_8 to give $\beta = \{7, 3, 2, 5\}$ which is identical to β_{30} so no new vertex is found and

$$\gamma_{15} \leftarrow \gamma_{15} - \{3\} = \{6\}, \qquad \gamma_{30} \leftarrow \gamma_{30} - \{8\} = \{1, 4, 6, 9\}$$

Step 4 gives via SOLVE,

$$a_6' = \begin{bmatrix} -\frac{1}{2} \\ 3 \\ 0 \\ \frac{1}{2} \end{bmatrix}$$

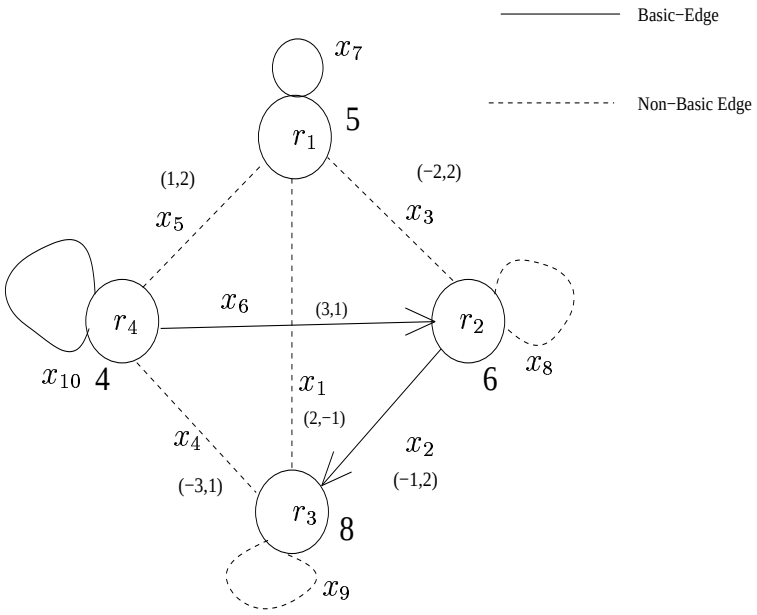


Figure 5.24: **The graph of vertex** 16

Performing a simplex pivot from V_{15} on this gives:

$\beta_{32} \leftarrow \beta_{15} \cup \{6\} - \{8\} = \{7, 6, 2, 5\}$

$\gamma_{32} \leftarrow \{1, 3, 4, 9, 10\} \quad \gamma_{15} \leftarrow \gamma_{15} - \{6\} = \emptyset$

We have: $p = 32$, $L = \{(\beta_1, \emptyset), .., (\beta_{15}, \emptyset), ..(\beta_{32}, \gamma_{32})\}$

Now $r = 16$, the graph of B_{16} is illustrated in Figure 5.24 which consists of two components; one simple loop and a loop joined to adjacent edges. SOLVE (a) gives $x_7 = 5$ from node r_1. SOLVE (c) gives $2x_2 = 8$ or $x_2 = 4$ from node r_3, then $3x_3 = 6 + x_2$ or $x_3 = \frac{10}{3}$ from node r_2, and $x_{10} = 4 - x_6 = \frac{2}{3}$ from node r_4.

Vertex 16 is given as:

$$V_{16} : \begin{bmatrix} x_7 \\ x_6 \\ x_2 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 5 \\ \frac{10}{3} \\ 4 \\ \frac{2}{3} \end{bmatrix} \tag{5.54}$$

As $\gamma_{16} = \{1, 4, 5\}$ we update a_1, a_4, a_5 using SOLVE to get:

$$a_1' = \begin{bmatrix} 2 \\ -\frac{1}{6} \\ -\frac{1}{2} \\ \frac{1}{6} \end{bmatrix} \quad a_4' = \begin{bmatrix} 0 \\ -\frac{1}{2} \\ -\frac{3}{2} \\ \frac{3}{2} \end{bmatrix} \quad a_5' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 6, 2, 10\}$ which is identical to β_{24} so no new vertex is found and

$$\gamma_{16} \leftarrow \gamma_{16} - \{1\} = \{4, 5\}, \qquad \gamma_{24} \leftarrow \gamma_{24} - \{7\} = \{3, 4, 5\}$$

and x_4 for x_{10} to give $\beta = \{7, 6, 2, 4\}$ which is identical to β_{31} so no new vertex is found and

$$\gamma_{16} \leftarrow \gamma_{16} - \{4\} = \{5\}, \qquad \gamma_{31} \leftarrow \gamma_{31} - \{10\} = \{1, 3, 5, 9\}$$

and x_5 for x_{10} to give $\beta = \{7, 6, 2, 5\}$ which is identical to β_{32} so no new vertex is found and

$$\gamma_{16} \leftarrow \gamma_{16} - \{5\} = \emptyset, \qquad \gamma_{32} \leftarrow \gamma_{32} - \{10\} = \{1, 3, 4, 9\}$$

No new vertex is found adjacent to V_{16}; p remains 32.

Now $r = 17$, the graph of B_{17} is illustrated in Figure 5.25 which consists of two components; each a simple loop joined to an edge. SOLVE (c) gives $x_4 = 4$ from node r_4, then $x_9 = 8 + 3x_4 = 20$ from node r_3, and $2x_3 = 6$ or $x_3 = 3$ from node r_2 then $x_7 = 2x_3 + 5 = 11$ from node r_1.

Vertex 17 is:

$$V_{17} : \begin{bmatrix} x_7 \\ x_3 \\ x_9 \\ x_4 \end{bmatrix} = \begin{bmatrix} 11 \\ 3 \\ 20 \\ 4 \end{bmatrix} \tag{5.55}$$

As $\gamma_{17} = \{1, 2, 5, 6\}$ we update a_1 using SOLVE to get:

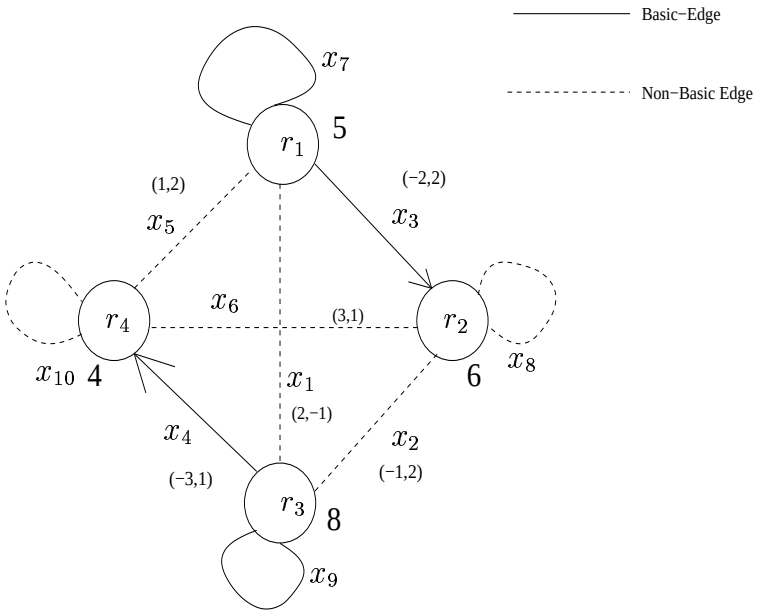


Figure 5.25: **The graph of vertex** 17

$$a_1' = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} \quad a_2' = \begin{bmatrix} -1 \\ -\frac{1}{2} \\ 2 \\ 0 \end{bmatrix} \quad a_5' = \begin{bmatrix} 1 \\ 0 \\ 6 \\ 2 \end{bmatrix} \quad a_6' = \begin{bmatrix} 3 \\ \frac{3}{2} \\ 3 \\ 1 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 3, 9, 4\}$ which is identical to β_{25} so no new vertex is found and

$$\gamma_{17} \leftarrow \gamma_{17} - \{1\} = \{2, 5, 6\}, \quad \gamma_{25} \leftarrow \gamma_{25} - \{7\} = \{2, 5, 6\}$$

and x_2 for x_9 to give $\beta = \{7, 3, 2, 4\}$ which is identical to β_{29} so no new vertex is found and

$$\gamma_{17} \leftarrow \gamma_{17} - \{2\} = \{5, 6\}, \quad \gamma_{29} \leftarrow \gamma_{29} - \{9\} = \{1, 5, 6\}$$

and x_5 for x_4 to give $\beta = \{7, 3, 9, 5\}$ which is identical to β_{18} so no new vertex is found and

$$\gamma_{17} \leftarrow \gamma_{17} - \{5\} = \{6\}, \quad \gamma_{18} \leftarrow \gamma_{18} - \{4\} = \{1, 2, 6\}$$

and x_6 for x_3 as the exiting variable to give $\beta = \{7, 6, 9, 4\}$ which is identical to β_{19} so no new vertex is found and

$$\gamma_{17} \leftarrow \gamma_{17} - \{6\} = \emptyset, \quad \gamma_{19} \leftarrow \gamma_{19} - \{3\} = \{1, 2, 5\}$$

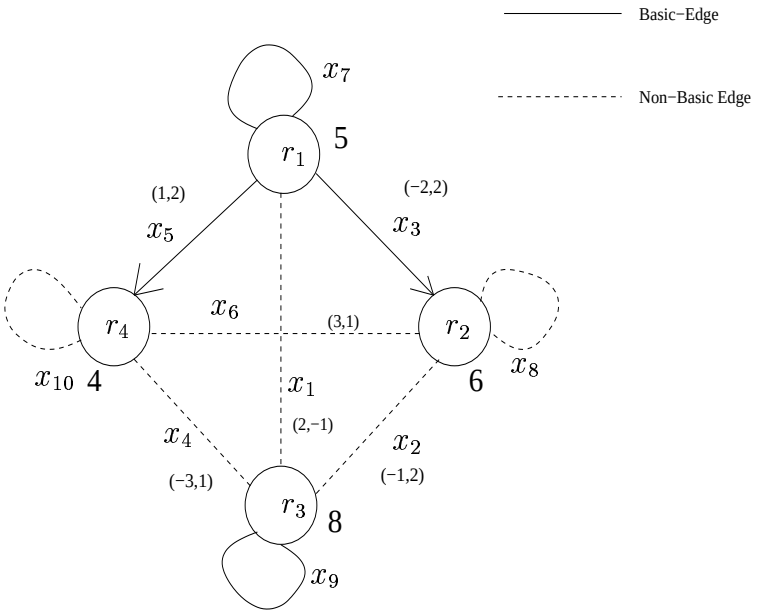


Figure 5.26: **The graph of vertex** 18

There is no new vertex found adjacent to V_{17}; p remains 32.

Now $r = 18$, the graph of B_{18} is illustrated in Figure 5.26 which consists of two components; one simple loop and a loop joined to two adjacent edges. SOLVE (a) gives $x_9 = 8$ from node r_3. SOLVE (c) gives $2x_3 = 6$ or $x_3 = 3$ from node r_2, then $2x_5 = 4$ or $x_5 = 2$ from node r_4 and $x_7 = 5 + 2x_3 - x_5 = 9$ from node r_1.

The 18^{th} vertex is:

$$V_{18} : \begin{bmatrix} x_7 \\ x_3 \\ x_9 \\ x_5 \end{bmatrix} = \begin{bmatrix} 9 \\ 3 \\ 8 \\ 2 \end{bmatrix} \tag{5.56}$$

As $\gamma_{18} = \{1, 2, 6\}$ we update a_1 using SOLVE to get:

$$a_1^{'} = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} \; a_2^{'} = \begin{bmatrix} -1 \\ -\frac{1}{2} \\ 2 \\ 0 \end{bmatrix} \; a_6^{'} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 0 \\ \frac{1}{2} \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 3, 9, 5\}$ which is identical to

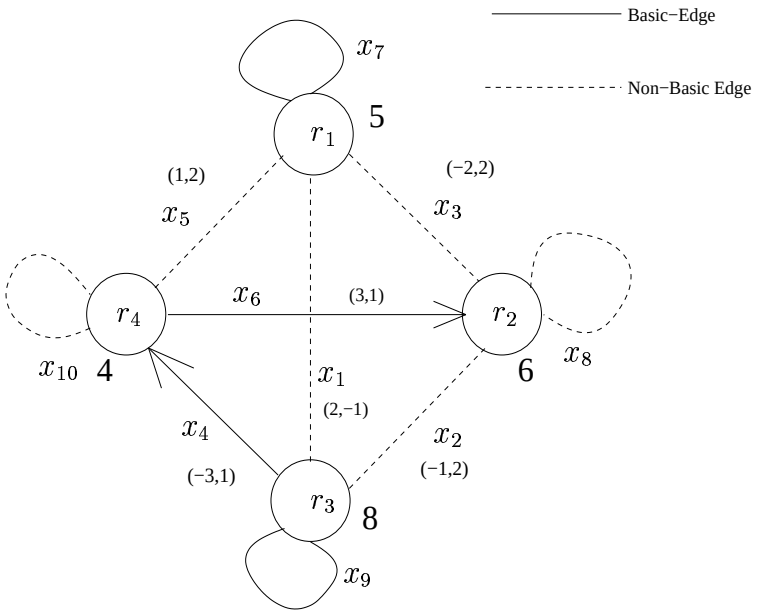


Figure 5.27: **The graph of vertex** 19

β_{26} so no new vertex is found and

$$\gamma_{18} \leftarrow \gamma_{18} - \{1\} = \{2, 6\}, \quad \gamma_{26} \leftarrow \gamma_{26} - \{7\} = \{2, 4, 6\}$$

and x_2 for x_9 to give $\beta = \{7, 3, 2, 5\}$ which is identical to β_{30} so no new vertex is found and

$$\gamma_{18} \leftarrow \gamma_{18} - \{2\} = \{6\}, \quad \gamma_{30} \leftarrow \gamma_{30} - \{9\} = \{1, 4, 6\}$$

and x_6 for x_3 to give $\beta = \{7, 6, 9, 5\}$ which is identical to β_{20} so no new vertex is found and

$$\gamma_{18} \leftarrow \gamma_{18} - \{6\} = \emptyset, \quad \gamma_{20} \leftarrow \gamma_{20} - \{3\} = \{1, 2, 4\}$$

There is no new vertex found adjacent to V_{18}; p remains 32.

Now $r = 19$, the graph of B_{19} is illustrated in Figure 5.27 which consists of two components; one simple loop and a loop joined to two adjacent edges. SOLVE (a) gives $x_7 = 5$ from node r_1. SOLVE (c) gives $3x_6 = 6$ or $x_6 = 2$ from node r_2, then $x_4 = 4 - x_6 = 2$ from node r_4 and $x_9 = 8 + 3x_4 = 14$ from node r_3.

Vertex 19 is:

$$V_{19}: \begin{bmatrix} x_7 \\ x_6 \\ x_9 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 14 \\ 2 \end{bmatrix} \tag{5.57}$$

As $\gamma_{19} = \{1, 2, 5\}$ we update a_1 using SOLVE to get:

$$a_1' = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 4 \end{bmatrix} \quad a_2' = \begin{bmatrix} 0 \\ -\frac{1}{3} \\ 3 \\ \frac{1}{3} \end{bmatrix} \quad a_5' = \begin{bmatrix} 1 \\ 0 \\ 6 \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 6, 9, 4\}$ which is identical to β_{27} so no new vertex is found and

$$\gamma_{19} \leftarrow \gamma_{19} - \{1\} = \{2, 5\}, \qquad \gamma_{27} \leftarrow \gamma_{27} - \{7\} = \{2, 3, 5\}$$

and x_2 for x_9 to give $\beta = \{7, 6, 2, 4\}$ which is identical to β_{31} so no new vertex is found and

$$\gamma_{19} \leftarrow \gamma_{19} - \{2\} = \{5\}, \qquad \gamma_{31} \leftarrow \gamma_{31} - \{9\} = \{1, 3, 5\}$$

and x_5 for x_4 to give $\beta = \{7, 6, 9, 5\}$ which is identical to β_{20} so no new vertex is found and

$$\gamma_{19} \leftarrow \gamma_{19} - \{5\} = \emptyset, \qquad \gamma_{20} \leftarrow \gamma_{20} - \{4\} = \{1, 2\}$$

There is no new vertex found adjacent to V_{19}; p remains 32.

Now $r = 20$, the graph of B_{20} is illustrated in Figure 5.28 which consists of two components; one simple loop and a loop joined to two adjacent edges. SOLVE (a) gives $x_9 = 8$ from node r_3. SOLVE (c) gives $3x_6 = 6$ or $x_6 = 2$ from node r_2, then $x_5 = \frac{4 - x_6}{2} = 1$ from node r_4 and $x_7 = 5 - x_5 = 4$ from node r_1.

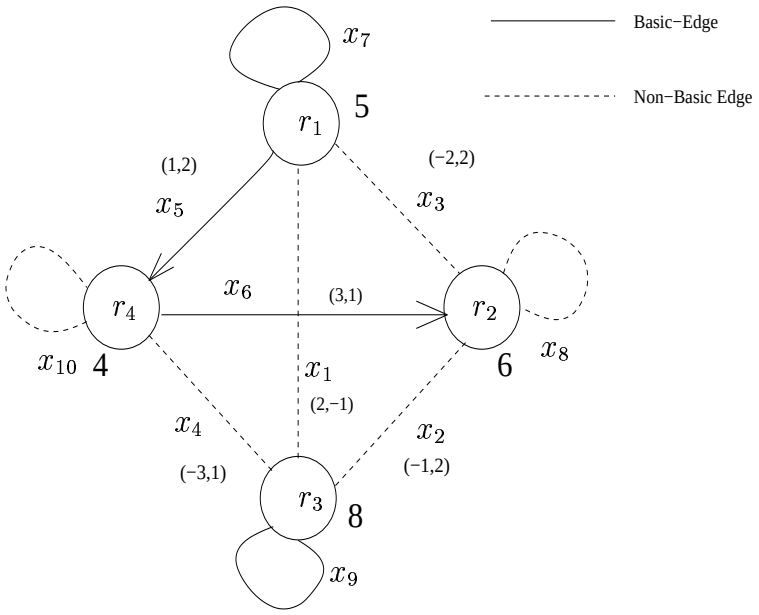


Figure 5.28: **The graph of vertex** 20

Vertex 20 is:

$$V_{20}: \begin{bmatrix} x_7 \\ x_6 \\ x_9 \\ x_5 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 8 \\ 1 \end{bmatrix} \tag{5.58}$$

As $\gamma_{20} = \{1, 2\}$ we update a_1 using SOLVE to get:

$$a_1' = \begin{bmatrix} 2 \\ 0 \\ -1 \\ 0 \end{bmatrix} \; a_2' = \begin{bmatrix} -\frac{1}{6} \\ -\frac{1}{3} \\ 2 \\ \frac{1}{6} \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 6, 9, 5\}$ which is identical to β_{28} so no new vertex is found and

$$\gamma_{20} \leftarrow \gamma_{20} - \{1\} = \{2\}, \quad \gamma_{28} \leftarrow \gamma_{28} - \{7\} = \{2, 3, 4\}$$

and x_2 for x_9 to give $\beta = \{7, 6, 2, 5\}$ which is identical to β_{32} so no new vertex is found and

$$\gamma_{20} \leftarrow \gamma_{20} - \{2\} = \emptyset, \quad \gamma_{32} \leftarrow \gamma_{32} - \{9\} = \{1, 3, 4\}$$

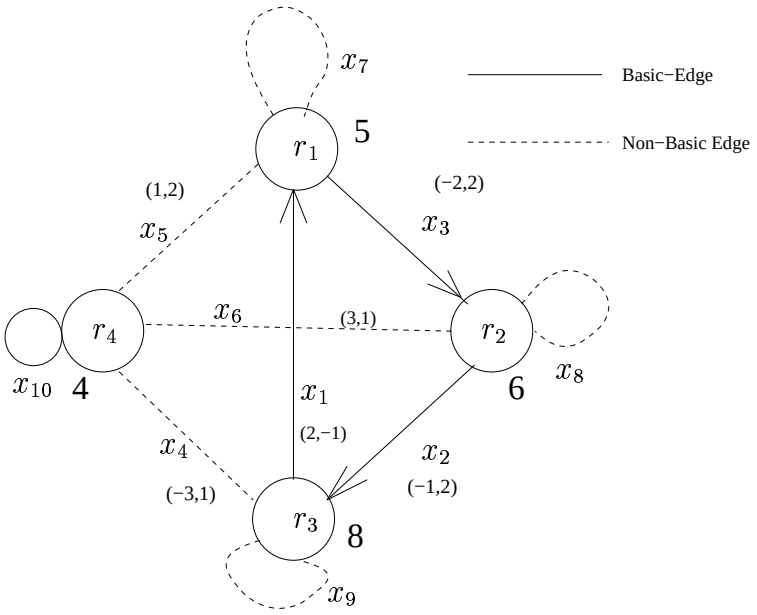


Figure 5.29: **The graph of vertex** 21

There is no new vertex found adjacent to V_{20}; p remains 32.

Now $r = 21$, the graph of B_{21} is illustrated in Figure 5.29 which consists of two components; one simple loop and a cycle. SOLVE (a) gives $x_{10} = 4$ from node r_4. SOLVE (c. parametrically) gives: Let $x_2 = \alpha$, then $x_1 = 2x_2 - 8 = 2\alpha - 8$ from node r_3 and $x_3 = \frac{5-x_1}{2} = \frac{4\alpha-21}{2}$ from node r_1. From node $r_2, x_2 = 2x_3 - 6 = 4\alpha - 27$. Hence $\alpha = 4\alpha - 27 \Rightarrow \alpha = 9$. So $x_2 = 9, x_1 = 18 - 8 = 10, x_3 = \frac{15}{2}$.
Vertex 21 is:

$$V_{21} : \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 10 \\ \frac{15}{2} \\ 9 \\ 4 \end{bmatrix} \tag{5.59}$$

Step 4 gives via SOLVE,

$$a_4' = \begin{bmatrix} -1 \\ -1 \\ -2 \\ 1 \end{bmatrix} \quad a_5' = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{6} \\ \frac{1}{3} \\ 2 \end{bmatrix}$$

Performing simplex pivots from V_{21} on each of these gives:

$\beta_{33} \leftarrow \beta_{21} \cup \{4\} - \{10\} = \{1, 3, 2, 4\}$

$\gamma_{33} \leftarrow \{5, 6, 7, 8, 9\} \quad \gamma_{21} \leftarrow \gamma_{21} - \{4\} = \{5, 6\}$

$\beta_{34} \leftarrow \beta_{21} \cup \{5\} - \{10\} = \{1, 3, 2, 5\}$

$\gamma_{34} \leftarrow \{4, 6, 7, 8, 9\} \quad \gamma_{21} \leftarrow \gamma_{21} - \{5\} = \{6\}$

We update a_6 using SOLVE to give:

$$a_6^{'} = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

Simplex pivoting gives x_3 as the exiting variable to give $\beta = \{1, 6, 2, 10\}$ which is identical to β_{24} so no new vertex is found and

$$\gamma_{21} \leftarrow \gamma_{21} - \{6\} = \emptyset, \quad \gamma_{24} \leftarrow \gamma_{24} - \{3\} = \{4, 5\}$$

We now have: $p = 34$, $L = \{(\beta_1, \emptyset), .., (\beta_{21}, \emptyset), ..(\beta_{34}, \gamma_{34})\}$.

Now $r = 22$, the graph of B_{22} is illustrated in Figure 5.30 which consists of one component; one loop joined to three adjacent edges. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1, $x_4 = 4$ from node r_4, $x_2 = \frac{8+x_1+3x_4}{2} = \frac{45}{4}$ from node r_2, and $x_8 = x_2 + 6 = \frac{69}{4}$ from node r_2.

Vertex 22 is:

$$V_{22} : \begin{bmatrix} x_1 \\ x_8 \\ x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{69}{4} \\ \frac{45}{4} \\ 4 \end{bmatrix} \tag{5.60}$$

As $\gamma_{22} = \{3, 5, 6\}$ we update a_3, a_5 using SOLVE to get:

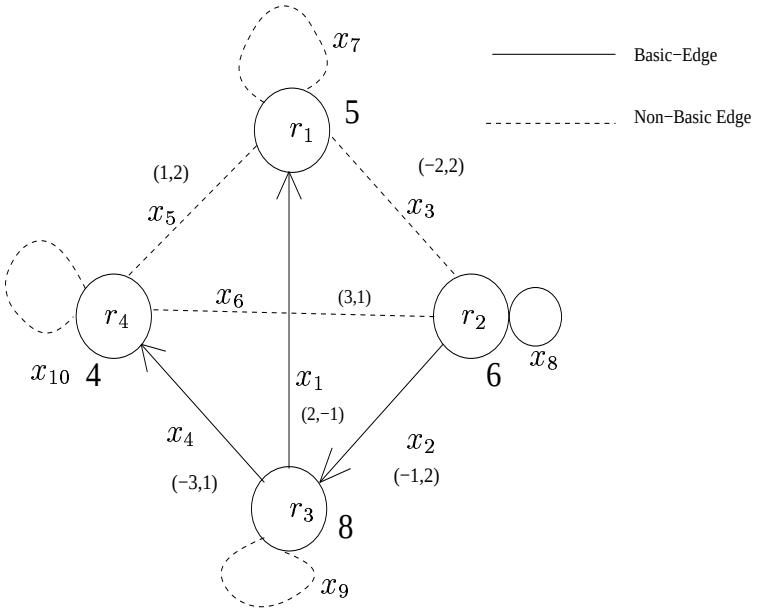


Figure 5.30: **The graph of vertex** 22

$$a_3' = \begin{bmatrix} -1 \\ \frac{3}{2}1 \\ -\frac{1}{2} \\ 0 \end{bmatrix} \quad a_5' = \begin{bmatrix} \frac{1}{2} \\ \frac{13}{4} \\ \frac{13}{4} \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_3 for x_8 to give $\beta = \{1, 3, 2, 4\}$ which is identical to β_{33} so no new vertex is found and

$$\gamma_{22} \leftarrow \gamma_{22} - \{3\} = \{5, 6\}, \qquad \gamma_{33} \leftarrow \gamma_{33} - \{8\} = \{5, 6, 7, 9\}$$

and x_5 for x_4 to give $\beta = \{1, 8, 2, 5\}$ which is identical to β_{23} so no new vertex is found and

$$\gamma_{22} \leftarrow \gamma_{22} - \{5\} = \{6\}, \qquad \gamma_{23} \leftarrow \gamma_{23} - \{4\} = \{3, 6\}$$

Step 4 gives via SOLVE,

$$a_6' = \begin{bmatrix} 0 \\ \frac{9}{2} \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

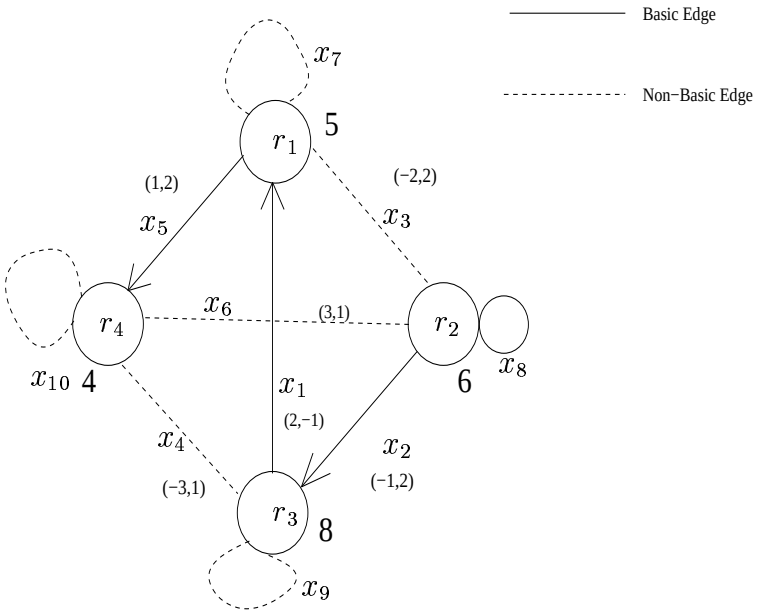


Figure 5.31: **The graph of vertex** 23

Performing a simplex pivot from V_{22} on this gives:

$\beta_{35} \leftarrow \beta_{22} \cup \{6\} - \{8\} = \{1, 6, 2, 4\}$

$\gamma_{35} \leftarrow \{3, 5, 7, 9, 10\}$ $\quad \gamma_{22} \leftarrow \gamma_{22} - \{6\} = \emptyset$

We have: $p = 35$, $L = \{(\beta_1, \emptyset), ..., (\beta_{22}, \emptyset), ..(\beta_{35}, \gamma_{35})\}$

Now $r = 23$, the graph of B_{23} is illustrated in Figure 5.31 which consists of one component; one loop joined to three adjacent edges. SOLVE (c) gives $2x_5 = 4$ or $x_5 = 2$ from node r_4, $x_1 = \frac{5-x_5}{2} = \frac{3}{2}$ from node r_1. $x_2 = \frac{8+x_1}{2} = \frac{19}{4}$ from node r_3. and $x_8 = x_2 + 6 = \frac{43}{4}$ from node r_2.

Vertex 23 is:

$$V_{23} : \begin{bmatrix} x_1 \\ x_8 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{3}{2} \\ \frac{43}{4} \\ \frac{19}{4} \\ 2 \end{bmatrix} \tag{5.61}$$

As $\gamma_{23} = \{3, 6\}$ we update a_3 using SOLVE to get:

$$a_3' = \begin{bmatrix} -1 \\ \frac{3}{2} \\ -\frac{1}{2} \\ 0 \end{bmatrix}$$

Simplex pivoting gives x_8 as the exiting variable to give $\beta = \{1, 3, 2, 5\}$ which is identical to β_{34} so no new vertex is found and

$$\gamma_{23} \leftarrow \gamma_{23} - \{3\} = \{6\}, \quad \gamma_{34} \leftarrow \gamma_{34} - \{8\} = \{4, 6, 7, 9\}$$

Step 4 gives via SOLVE,

$$a_6' = \begin{bmatrix} -\frac{1}{4} \\ \frac{23}{8} \\ -\frac{1}{8} \\ \frac{1}{2} \end{bmatrix}$$

Performing a simplex pivot from V_{23} on this gives:

$\beta_{36} \leftarrow \beta_{23} \cup \{6\} - \{8\} = \{1, 6, 2, 5\}$

$\gamma_{36} \leftarrow \{3, 4, 7, 9, 10\} \quad \gamma_{23} \leftarrow \gamma_{23} - \{6\} = \emptyset$

Now we have: $p = 36$, $L = \{(\beta_1, \emptyset), .., (\beta_{23}, \emptyset), ..(\beta_{36}, \gamma_{36})\}$

Now $r = 24$, the graph of B_{24} is illustrated in Figure 5.32 which consists of one component; one loop joined to three adjacent edges. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1, $x_2 = 8 + x_1 = \frac{21}{4}$ from node r_3, $x_6 = \frac{6+x_2}{3} = \frac{15}{4}$ from node r_2 and $x_{10} = 4 - x_6 = \frac{1}{4}$ from node r_4.

Vertex 24 is:

$$V_{24} : \begin{bmatrix} x_1 \\ x_6 \\ x_2 \\ x_{10} \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{15}{4} \\ \frac{21}{4} \\ \frac{1}{4} \end{bmatrix} \tag{5.62}$$

As $\gamma_{24} = \{4, 5\}$ we update a_4 using SOLVE to get:

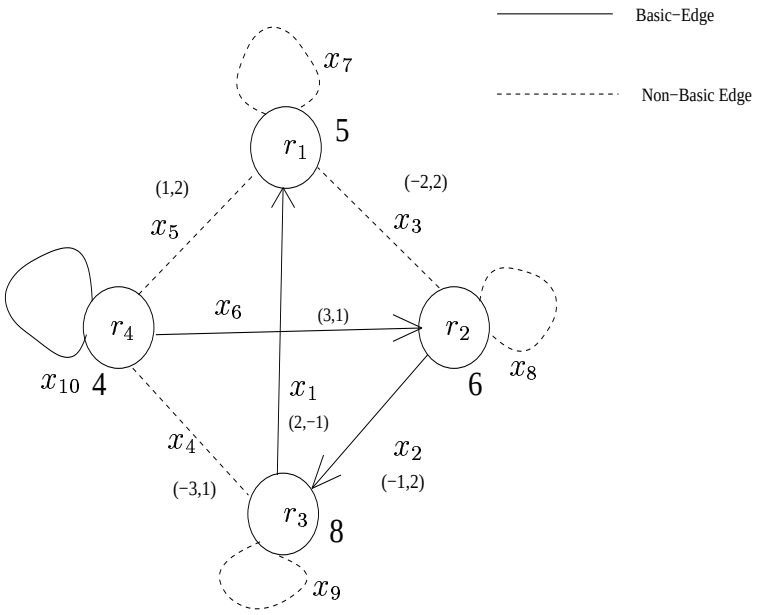


Figure 5.32: **The graph of vertex** 24

$$a_4' = \begin{bmatrix} 0 \\ -\frac{1}{2} \\ -\frac{3}{2} \\ \frac{3}{2} \end{bmatrix} \quad a_5' = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{12} \\ \frac{1}{4} \\ \frac{23}{12} \end{bmatrix}$$

Simplex pivoting exchanges x_4 for x_{10} to give $\beta = \{1, 6, 2, 4\}$ which is identical to β_{35} so no new vertex is found and

$$\gamma_{24} \leftarrow \gamma_{24} - \{4\} = \{5\}, \qquad \gamma_{35} \leftarrow \gamma_{35} - \{10\} = \{3, 5, 7, 9\}$$

and x_5 for x_{10} to give $\beta = \{1, 6, 2, 4\}$ which is identical to β_{36} so no new vertex is found and

$$\gamma_{24} \leftarrow \gamma_{24} - \{5\} = \emptyset, \qquad \gamma_{36} \leftarrow \gamma_{36} - \{10\} = \{3, 4, 7, 9\}$$

There is no new vertex found adjacent to V_{24}; p remains 36.

Now $r = 25$, the graph of B_{25} is illustrated in Figure 5.33 which consists of one component; one loop joined to three adjacent edges. SOLVE (c) gives $2x_3 = 6$ or $x_3 = 3$ from node r_2 while $x_4 = 4$ from node r_4, $x_1 = \frac{2x_3+5}{2} = \frac{11}{2}$ from node r_1 and $x_9 = x_1 + 3x_4 + 8 = \frac{51}{2}$ from node r_3.

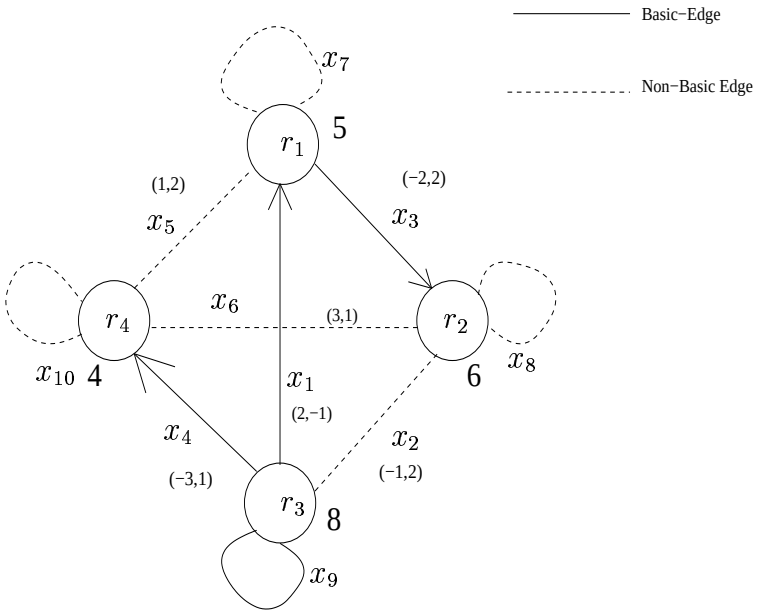


Figure 5.33: **The graph of vertex** 25

Vertex 25 is:

$$V_{25}: \begin{bmatrix} x_1 \\ x_3 \\ x_9 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{11}{2} \\ 3 \\ \frac{51}{2} \\ 4 \end{bmatrix} \tag{5.63}$$

As $\gamma_{25} = \{2, 5, 6\}$ we update a_2 using SOLVE to get:

$$a_2^{'} = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{3}{2} \\ 0 \end{bmatrix} \quad a_5^{'} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{13}{2} \\ 2 \end{bmatrix} \quad a_6^{'} = \begin{bmatrix} \frac{3}{2} \\ \frac{3}{2} \\ \frac{9}{2} \\ 1 \end{bmatrix}$$

Simplex pivoting exchanges x_2 for x_9 to give $\beta = \{1, 3, 2, 4\}$ which is identical to β_{33} so no new vertex is found and

$$\gamma_{25} \leftarrow \gamma_{25} - \{2\} = \{5, 6\}, \quad \gamma_{33} \leftarrow \gamma_{33} - \{9\} = \{5, 6, 7\}$$

and x_5 for x_4 to give $\beta = \{1, 3, 9, 5\}$ which is identical to β_{26} so no new vertex is found and

$$\gamma_{25} \leftarrow \gamma_{25} - \{5\} = \{6\}, \quad \gamma_{26} \leftarrow \gamma_{26} - \{4\} = \{2, 6\}$$

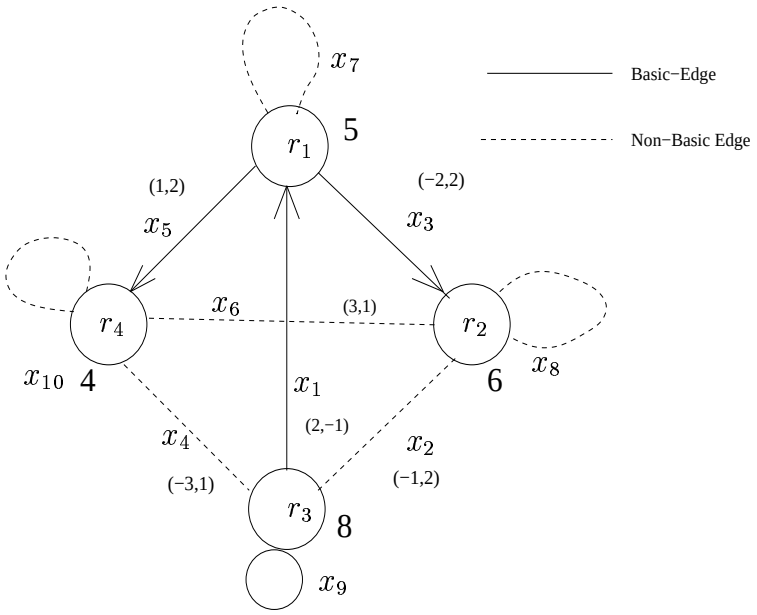


Figure 5.34: **The graph of vertex** 26

and x_6 for x_3 as the exiting variable to give $\beta = \{1, 6, 9, 4\}$ which is identical to β_{27} so no new vertex is found and

$$\gamma_{25} \leftarrow \gamma_{25} - \{6\} = \emptyset, \qquad \gamma_{27} \leftarrow \gamma_{27} - \{3\} = \{2, 5\}$$

There is no new vertex adjacent to V_{25}; p remains 36.

Now $r = 26$, the graph of B_{26} is illustrated in Figure 5.34 which consists of one component; a simple loop joined to three adjacent edges. SOLVE (c) gives $2x_3 = 6$ or $x_3 = 3$ from node r_2 while $x_5 = 2$ from node r_4, $x_1 = \frac{5+2x_3-x_5}{2} = \frac{9}{2}$ from node r_1, and $x_9 = 8 + x_1 = \frac{25}{2}$ from node r_3.

Vertex 26 is:

$$V_{26} : \begin{bmatrix} x_1 \\ x_3 \\ x_9 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{9}{2} \\ 3 \\ \frac{25}{2} \\ 2 \end{bmatrix} \tag{5.64}$$

As $\gamma_{26} = \{2, 6\}$ we update a_2 using SOLVE to get:

$$a_2' = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{3}{2} \\ 0 \end{bmatrix} \quad a_6' = \begin{bmatrix} \frac{5}{4} \\ \frac{3}{2} \\ \frac{5}{4} \\ \frac{1}{2} \end{bmatrix}$$

Simplex pivoting exchanges x_2 for x_9 to give $\beta = \{1, 3, 2, 5\}$ which is identical to β_{34} so no new vertex is found and

$$\gamma_{26} \leftarrow \gamma_{26} - \{2\} = \{6\}, \quad \gamma_{34} \leftarrow \gamma_{34} - \{9\} = \{4, 6, 7\}$$

and x_6 for x_3 to give $\beta = \{1, 6, 9, 5\}$ which is identical to β_{28} so no new vertex is found and

$$\gamma_{26} \leftarrow \gamma_{26} - \{6\} = \emptyset, \quad \gamma_{28} \leftarrow \gamma_{28} - \{3\} = \{2, 4\}$$

There is no new vertex found adjacent to V_{26}; p remains 36.

Now $r = 27$, the graph of B_{27} is illustrated in Figure 5.35 which consists of one component; a loop joined to three adjacent edges. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1 while $3x_6 = 6$ or $x_6 = 2$ from node r_2, $x_4 = 4 - x_6 = 2$ from node r_4 and $x_9 = 8 + x_1 + 3x_4 = \frac{33}{2}$ from node r_3.

Vertex 27 is:

$$V_{27} : \begin{bmatrix} x_1 \\ x_6 \\ x_9 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ 2 \\ \frac{33}{2} \\ 2 \end{bmatrix} \tag{5.65}$$

As $\gamma_{27} = \{2, 5\}$ we update a_2, a_5 using SOLVE to get:

$$a_2' = \begin{bmatrix} 0 \\ -\frac{1}{3} \\ 3 \\ \frac{1}{3} \end{bmatrix} \quad a_5' = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \frac{13}{2} \\ 2 \end{bmatrix}$$

Simplex pivoting exchanges x_2 for x_9 to give $\beta = \{1, 6, 2, 4\}$ which is identical to β_{35} so no new vertex is found and

$$\gamma_{27} \leftarrow \gamma_{27} - \{2\} = \{5\}, \quad \gamma_{35} \leftarrow \gamma_{35} - \{9\} = \{3, 5, 7\}$$

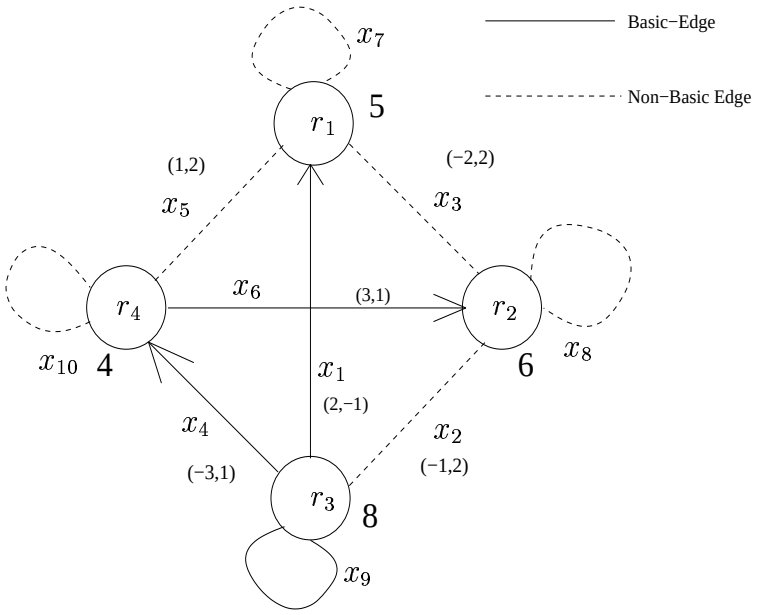


Figure 5.35: **The graph of vertex** 27

and x_5 for x_4 as the exiting variable to give $\beta = \{1, 6, 9, 5\}$ which is identical to β_{28} so no new vertex is found and

$$\gamma_{27} \leftarrow \gamma_{27} - \{5\} = \emptyset, \qquad \gamma_{28} \leftarrow \gamma_{28} - \{4\} = \{2\}$$

No new vertex is found adjacent to V_{27}; p remains 36.

Now $r = 28$, the graph of B_{28} is illustrated in Figure 5.36 which consists of one component; a loop joined to three adjacent edges. SOLVE (c) gives $3x_6 = 6$ or $x_6 = 2$ from node r_2 while $2x_5 = 4 - x_6$ or $x_5 = 1$ from node r_4, $x_1 = \frac{5-x_5}{2} = 2$ from node r_1. and $x_9 = 8 + x_1 = 10$ from node r_3.

Vertex 28 is:

$$V_{28} : \begin{bmatrix} x_1 \\ x_6 \\ x_9 \\ x_5 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 10 \\ 1 \end{bmatrix} \tag{5.66}$$

As $\gamma_{28} = \{2\}$ we update a_2 using SOLVE to get:

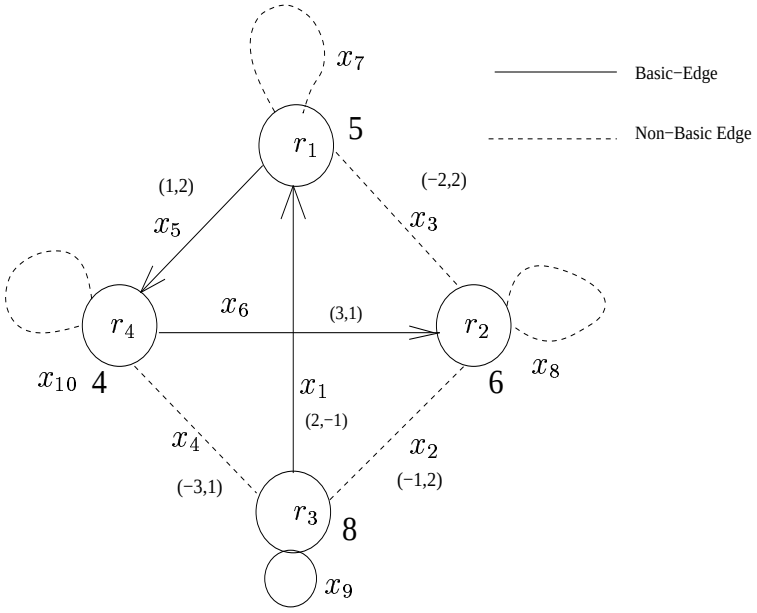


Figure 5.36: **The graph of vertex** 28

$$a_2' = \begin{bmatrix} -\frac{1}{12} \\ -\frac{1}{3} \\ \frac{23}{12} \\ \frac{1}{6} \end{bmatrix}$$

Simplex pivoting gives x_9 as the exiting variable to give $\beta = \{1, 6, 2, 5\}$ which is identical to β_{36} so no new vertex is found and

$$\gamma_{28} \leftarrow \gamma_{28} - \{2\} = \emptyset, \quad \gamma_{36} \leftarrow \gamma_{36} - \{9\} = \{3, 4, 7\}$$

There is no new vertex found adjacent to V_{28}; p remains 36.

Now $r = 29$, the graph of B_{29} is illustrated in Figure 5.37 which consists of one component; a loop joined to three adjacent edges. SOLVE (c) gives $x_4 = 4$ from node r_4 while $x_2 = \frac{8+3x_4}{2} = 10$ from node r_3, $x_3 = \frac{6+x_2}{2} = 8$ from node r_2 and $x_7 = 5 + 2x_3 = 21$ from node r_1.

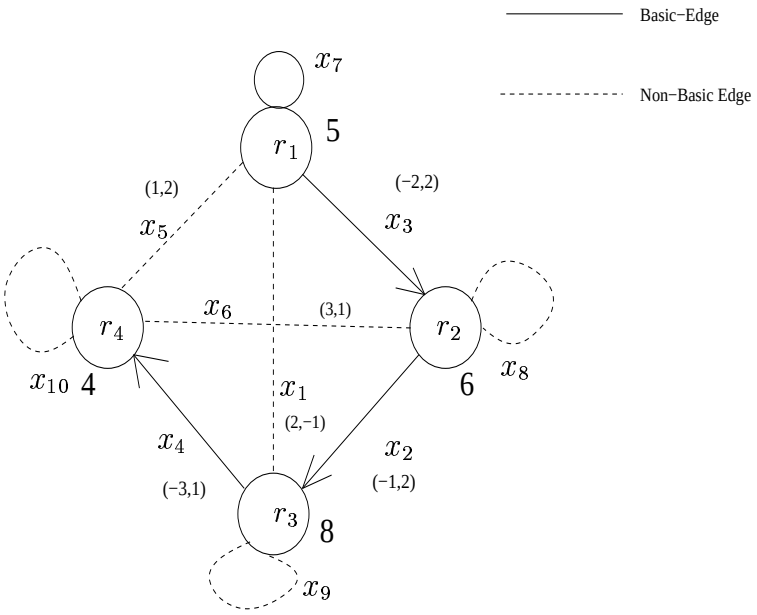


Figure 5.37: **The graph of vertex** 29

Vertex 29 is:

$$V_{29}: \begin{bmatrix} x_7 \\ x_3 \\ x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} 21 \\ 8 \\ 10 \\ 4 \end{bmatrix} \tag{5.67}$$

As $\gamma_{29} = \{1, 5, 6\}$ we update a_1 using SOLVE to get:

$$a_1' = \begin{bmatrix} \frac{3}{2} \\ -\frac{1}{4} \\ -\frac{1}{2} \\ 0 \end{bmatrix} \quad a_5' = \begin{bmatrix} 4 \\ \frac{3}{2} \\ 3 \\ 2 \end{bmatrix} \quad a_6' = \begin{bmatrix} \frac{9}{2} \\ \frac{9}{4} \\ \frac{3}{2} \\ 1 \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 3, 2, 4\}$ which is identical to β_{33} so no new vertex is found and

$$\gamma_{29} \leftarrow \gamma_{29} - \{1\} = \{5, 6\}, \qquad \gamma_{33} \leftarrow \gamma_{33} - \{7\} = \{5, 6\}$$

and x_5 for x_4 to give $\beta = \{1, 3, 2, 5\}$ which is identical to β_{30} so no new vertex is found and

$$\gamma_{29} \leftarrow \gamma_{29} - \{5\} = \{6\}, \qquad \gamma_{30} \leftarrow \gamma_{30} - \{4\} = \{1, 6\}$$

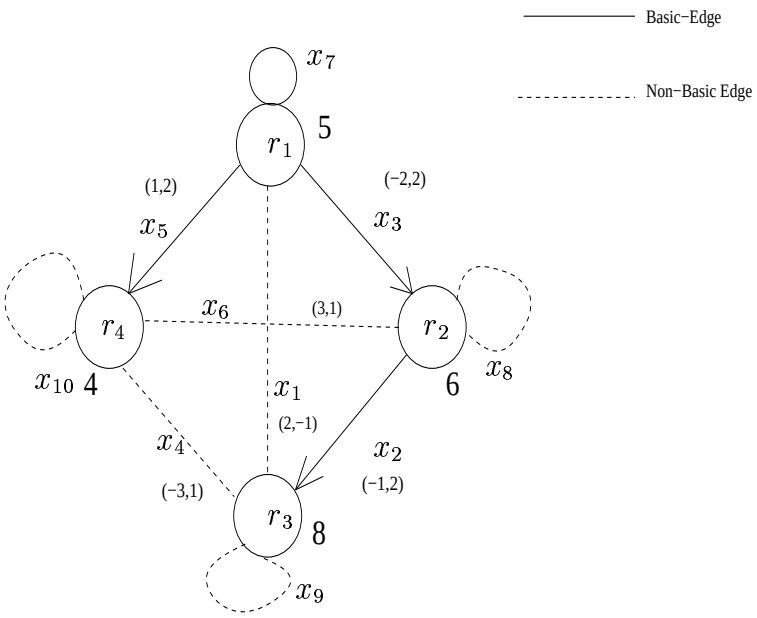


Figure 5.38: **The graph of vertex** 30

and x_6 for x_3 to give $\beta = \{7, 6, 2, 4\}$ which is identical to β_{31} so no new vertex is found and

$$\gamma_{29} \leftarrow \gamma_{29} - \{6\} = \emptyset, \qquad \gamma_{31} \leftarrow \gamma_{31} - \{3\} = \{1, 5\}$$

There is no new vertex found adjacent to V_{29}; p remains 36.

Now $r = 30$, the graph of B_{30} is illustrated in Figure 5.38 which consists of one component; a loop joined to three adjacent edges. SOLVE (c) gives $2x_5 = 4$ or $x_5 = 2$ from node r_4 while $2x_2 = 8$ or $x_2 = 4$ from node r_3, $x_3 = \frac{6+x_2}{2} = 5$ from node r_2 and $x_7 = 2x_3 - x_5 + 5 = 13$ from node r_1.

$$V_{30} : \begin{bmatrix} x_7 \\ x_3 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} 13 \\ 5 \\ 4 \\ 2 \end{bmatrix} \tag{5.68}$$

As $\gamma_{30} = \{1, 6\}$ we update a_1, a_6 using SOLVE to get:

$$a_1' = \begin{bmatrix} \frac{3}{2} \\ -\frac{1}{4} \\ -\frac{1}{2} \\ 0 \end{bmatrix} \quad a_6' = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 0 \\ \frac{1}{2} \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 3, 2, 5\}$ which is identical to β_{34} so no new vertex is found and

$$\gamma_{30} \leftarrow \gamma_{30} - \{1\} = \{6\}, \qquad \gamma_{34} \leftarrow \gamma_{34} - \{7\} = \{4, 6\}$$

and x_6 for x_3 to give $\beta = \{7, 6, 2, 5\}$ which is identical to β_{32} so no new vertex is found and

$$\gamma_{30} \leftarrow \gamma_{30} - \{6\} = \emptyset, \qquad \gamma_{32} \leftarrow \gamma_{32} - \{3\} = \{1, 4\}$$

No new vertex is found adjacent to V_{30}; p remains at 36.

Now $r = 31$, the graph of B_{31} is illustrated in Figure 5.39 which consists of two components; a simple loop and a cycle. SOLVE (a) gives $x_7 = 5$ from node r_1. SOLVE (c. parametrically) gives: Let $x_4 = \alpha$ then $x_6 = 4 - \alpha$ from node r_4 and $x_2 = 6 - 3\alpha$ from node r_2 so $x_4 = \frac{4-6\alpha}{3} \Rightarrow \alpha = \frac{4}{9}$. Hence $x_4 = \frac{4}{4}$, $x_2 = \frac{14}{3}$ and $x_6 = \frac{32}{9}$.

Vertex 31 is:

$$V_{31} : \begin{bmatrix} x_7 \\ x_6 \\ x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ \frac{32}{9} \\ \frac{14}{3} \\ \frac{4}{9} \end{bmatrix} \tag{5.69}$$

As $\gamma_{31} = \{1, 5\}$ we update a_1 using SOLVE to get:

$$a_1' = \begin{bmatrix} 2 \\ -\frac{1}{9} \\ -\frac{1}{3} \\ \frac{1}{9} \end{bmatrix} \quad a_5' = \begin{bmatrix} 1 \\ \frac{2}{3} \\ 2 \\ \frac{4}{3} \end{bmatrix}$$

Simplex pivoting exchanges x_1 for x_7 to give $\beta = \{1, 6, 2, 5\}$ which is identical to

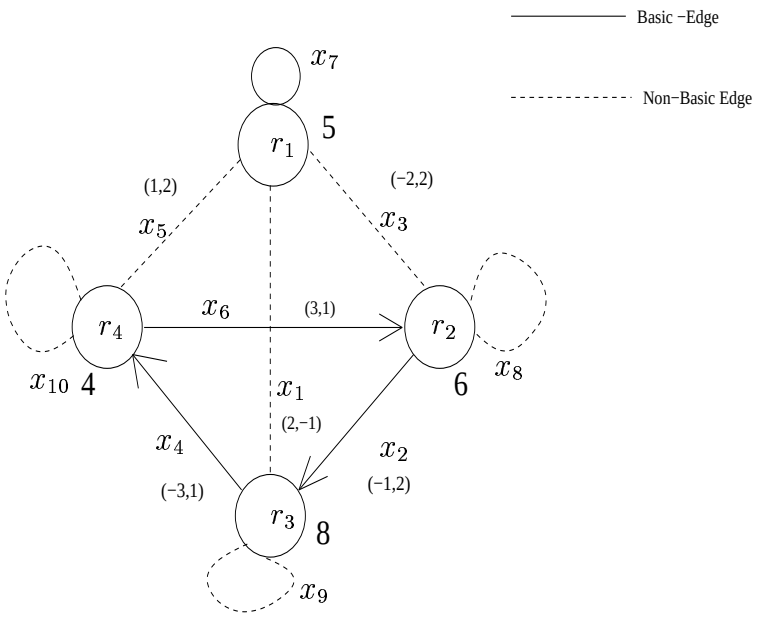


Figure 5.39: **The graph of vertex** 31

β_{35} so no new vertex is found and

$$\gamma_{31} \leftarrow \gamma_{31} - \{1\} = \{5\}, \qquad \gamma_{35} \leftarrow \gamma_{35} - \{7\} = \{3, 5\}$$

and x_5 for x_4 to give $\beta = \{7, 6, 2, 5\}$ which is identical to β_{32} so no new vertex is found and

$$\gamma_{31} \leftarrow \gamma_{31} - \{5\} = \emptyset, \qquad \gamma_{32} \leftarrow \gamma_{32} - \{4\} = \{1\}$$

There is no new vertex found adjacent to V_{31}; p remains at 36.

Now $r = 32$, the graph of B_{32} is illustrated in Figure 5.40 which consists of one component; a loop joined to three adjacent edges. SOLVE (c) gives $2x_2 = 8$ or $x_2 = 4$ from node r_3 then $x_6 = \frac{6+x_2}{3} = \frac{10}{3}$ from node r_2, $x_5 = \frac{4-x_6}{2}$, or $x_5 = \frac{1}{3}$ from node r_4 and $x_7 = 5 - x_5 = \frac{14}{3}$ from node r_1.

Vertex 32 is:

$$V_{32} : \begin{bmatrix} x_7 \\ x_6 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{14}{3} \\ \frac{10}{3} \\ 4 \\ \frac{1}{3} \end{bmatrix} \tag{5.70}$$

As $\gamma_{32} = \{1\}$ we update a_1 using SOLVE to get:

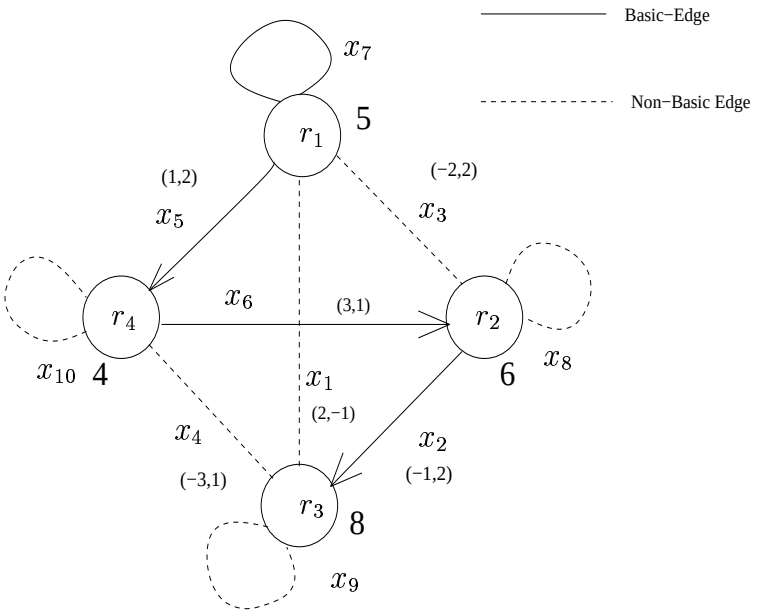


Figure 5.40: **The graph of vertex** 32

$$a_1^{'} = \begin{bmatrix} \frac{23}{12} \\ -\frac{1}{6} \\ -\frac{1}{2} \\ \frac{1}{12} \end{bmatrix}$$

Simplex pivoting gives x_7 as the exiting variable to give $\beta = \{1, 6, 2, 5\}$ which is identical to β_{36} so no new vertex is found and

$$\gamma_{32} \leftarrow \gamma_{32} - \{1\} = \emptyset, \qquad \gamma_{36} \leftarrow \gamma_{36} - \{7\} = \{3, 4\}$$

No new vertex is found adjacent to V_{32}; p remains 36.

Now $r = 33$, the graph of B_{33} is illustrated in Figure 5.41 which consists of one component; a cycle joined to an adjacent edge. SOLVE (c) gives $x_4 = 4$ from node r_4 (using parametric method) let $x_2 = \alpha$, from node r_3 $x_1 = 2\alpha - 20$, from node $r_1 x_3 = \frac{4\alpha - 45}{2}$ and, from node r_2, $x_2 = 4\alpha - 51 \Rightarrow \alpha = 17$, Hence $x_2 = 9, x_3 = \frac{23}{2}, x_1 = 14$.

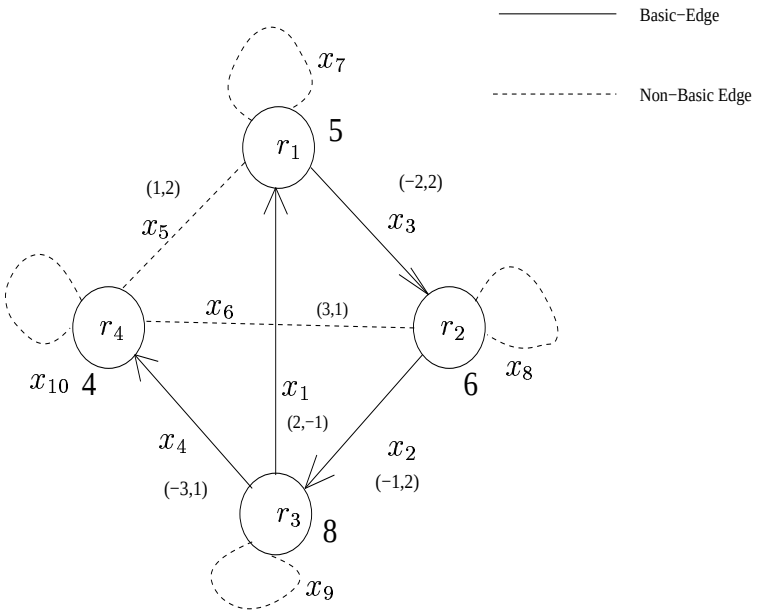


Figure 5.41: **The graph of vertex** 33

Vertex 33 is:

$$V_{33}: \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} 14 \\ \frac{23}{2} \\ 17 \\ 4 \end{bmatrix} \tag{5.71}$$

As $\gamma_{33} = \{5, 6\}$ we update a_5, a_6 using SOLVE to get:

$$a_5' = \begin{bmatrix} \frac{8}{3} \\ \frac{13}{6} \\ \frac{13}{3} \\ 2 \end{bmatrix} \quad a_6' = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 1 \end{bmatrix}$$

Simplex pivoting exchanges x_5 for x_4 to give $\beta = \{1, 3, 2, 5\}$ which is identical to β_{34} so no new vertex is found and

$$\gamma_{33} \leftarrow \gamma_{33} - \{5\} = \{6\}, \qquad \gamma_{34} \leftarrow \gamma_{34} - \{4\} = \{6\}$$

and x_6 for x_3 as the exiting variable to give $\beta = \{1, 6, 2, 4\}$ which is identical to β_{35} so no new vertex is found and

$$\gamma_{33} \leftarrow \gamma_{33} - \{6\} = \emptyset, \qquad \gamma_{35} \leftarrow \gamma_{35} - \{3\} = \{5\}$$

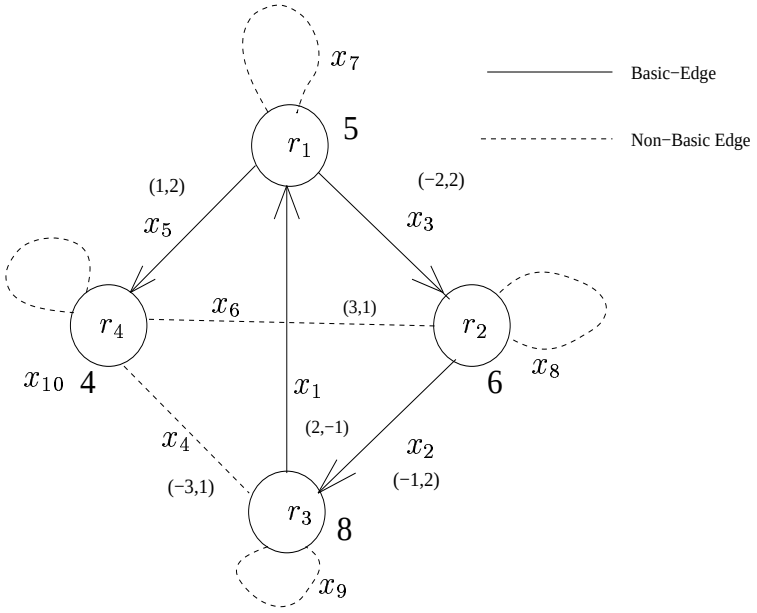


Figure 5.42: **The graph of vertex** 34

There is no new vertex adjacent to V_{33}; p remains 36.

Now $r = 34$, the graph of B_{34} is illustrated in Figure 5.42 which consists of one component; a simple cycle joined to an adjacent edge. SOLVE (c) gives $2x_5 = 4$ or $x_5 = 2$ from node r_4 (using parametric method) let $x_1 = \alpha$, then, from node r_1 $x_3 = \frac{2\alpha-3}{2}$, from node r_2 $x_2 = 2\alpha - 9$, and from node r_3 $x_1 = 4\alpha - 26 \Rightarrow \alpha = \frac{26}{3}$ Hence $x_1 = \frac{26}{3}, x_2 = \frac{25}{3}, x_3 = \frac{43}{6}$.

Vertex 34 is:

$$V_{34} : \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{26}{3} \\ \frac{43}{6} \\ \frac{25}{3} \\ 2 \end{bmatrix} \tag{5.72}$$

As $\gamma_{34} = \{6\}$ we update a_6 using SOLVE to get:

$$a_6' = \begin{bmatrix} \frac{5}{3} \\ \frac{23}{12} \\ \frac{5}{6} \\ \frac{1}{2} \end{bmatrix}$$

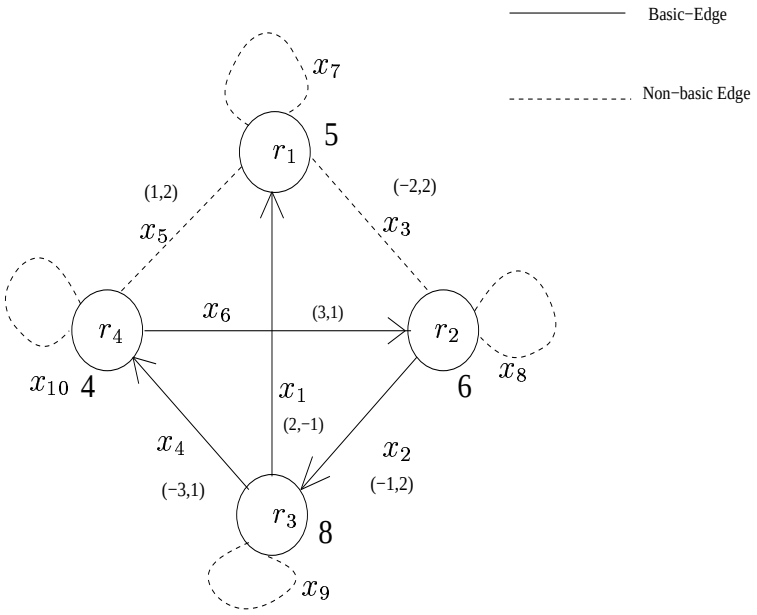


Figure 5.43: **The graph of vertex** 35

Simplex pivoting gives x_3 as the exiting variable to give $\beta = \{1, 6, 2, 5\}$ which is identical to β_{36} so no new vertex is found and

$$\gamma_{34} \leftarrow \gamma_{34} - \{6\} = \emptyset, \qquad \gamma_{36} \leftarrow \gamma_{36} - \{3\} = \{4\}$$

No new vertex is found adjacent to V_{34}; p remains 36.

Now $r = 35$, the graph of B_{35} is illustrated in Figure 5.43 which consists of one component; a cycle joined to an adjacent edge. SOLVE (c) gives $2x_1 = 5$ or $x_1 = \frac{5}{2}$ from node r_1, (using parametric method) let $x_4 = \alpha$, then from node r_4 $x_6 = 4 - \alpha$, from node r_2 $x_2 = 6 - 3\alpha$, and from node r_3 $x_4 = \frac{1-4\alpha}{2} \Rightarrow \alpha = \frac{1}{6}$ Hence $x_4 = \frac{1}{6}, x_2 = \frac{11}{2}, x_6 = \frac{23}{6}$.

Vertex 35 is:

$$V_{35} : \begin{bmatrix} x_1 \\ x_6 \\ x_2 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{23}{6} \\ \frac{11}{2} \\ \frac{1}{6} \end{bmatrix} \tag{5.73}$$

As $\gamma_{35} = \{5\}$ we update a_5 using SOLVE to get:

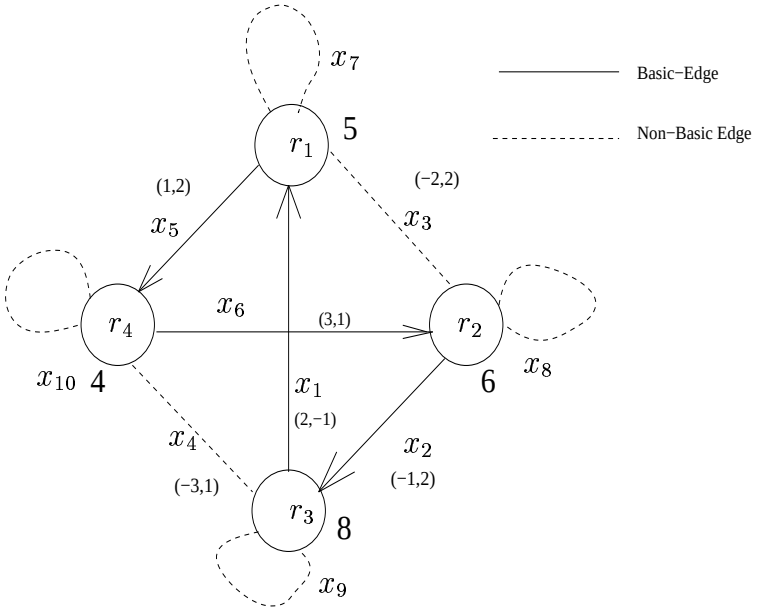


Figure 5.44: **The graph of vertex** 36

$$a_5' = \begin{bmatrix} \frac{1}{2} \\ \frac{13}{18} \\ \frac{13}{6} \\ \frac{23}{18} \end{bmatrix}$$

Simplex pivoting gives x_4 as the exiting variable to give $\beta = \{1, 6, 2, 5\}$ which is identical to β_{36} so no new vertex is found and

$$\gamma_{35} \leftarrow \gamma_{35} - \{5\} = \emptyset, \qquad \gamma_{36} \leftarrow \gamma_{36} - \{4\} = \emptyset$$

There is no new vertex found adjacent to V_{35}; p remains 36.

Now $r = 36$, the graph of B_{36} is illustrated in Figure 5.44 which consists of one component; a cycle. SOLVE (c. parametrically) gives: Let $x_1 = \alpha$, from node r_1 $x_5 = 5 - 2\alpha$, from node r_4 $x_6 = 4\alpha - 6$, from node r_2 $x_2 = 12\alpha - 24$ and from node $r_3 x_1 = 24\alpha - 56 \Rightarrow \alpha = \frac{56}{23}$. Hence $x_1 = \frac{56}{23}, x_2 = \frac{120}{23}, x_6 = \frac{86}{23}$.

The 36^{th} vertex is:

$$V_{36}: \begin{bmatrix} x_1 \\ x_6 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{56}{23} \\ \frac{86}{23} \\ \frac{120}{23} \\ \frac{3}{23} \end{bmatrix} \tag{5.74}$$

Now $r = 36$ but $\gamma_{36} = \emptyset$ so the algorithm terminates.

The complete spanning tree for the vertices of the polyhedron formed by inequalities (5.28) to (5.31) is given in Figure 5.45. The correctness of the vertices obtained has been verified by the dual *F-M* algorithm discussed in Chapter 4 and by Dyer's algorithm.

5.5 Complexity Analysis

The complexity of **EnumerateVertices**, in the absence of degeneracy, can be analysed as follows. In each loop, all computations are bounded by the number of edges in $G(B)$, i.e. $O(m)$, except for the step which identifies the edge to enter the new basis. This computation is $O(n)$, since n is the number of edges in $G(A)$. Assuming a good hash function for the test for adjacency, this step will also be $O(m)$ at worst. Thus each loop computation is $O(n)$ (assuming $n \geq m$). If there are v nondegenerate vertices in total, the complexity of the algorithm will therefore be $O(nv)$. This analysis excludes the time to find an initial basis.

5.6 Polyhedra with Unbounded Edges

EnumerateVertices as described in section 5.4 fails at Step 4 if $\exists i \in \gamma_r$ such that $a'_{ji} \leq 0, \forall i$ because there is then no valid primal simplex pivot. This characterises the existence of an unbounded edge incident at V_r. We can deal with this by modifying Step 4 to be:

$\forall j \in \gamma_r$,

determine a'_j using **SOLVE**$(G(B_r), a_j, a'_j)$

if $\{a'_{ji} : a'_{ji} > 0\} = \emptyset$ then

$\gamma_r \leftarrow \gamma_r - j$

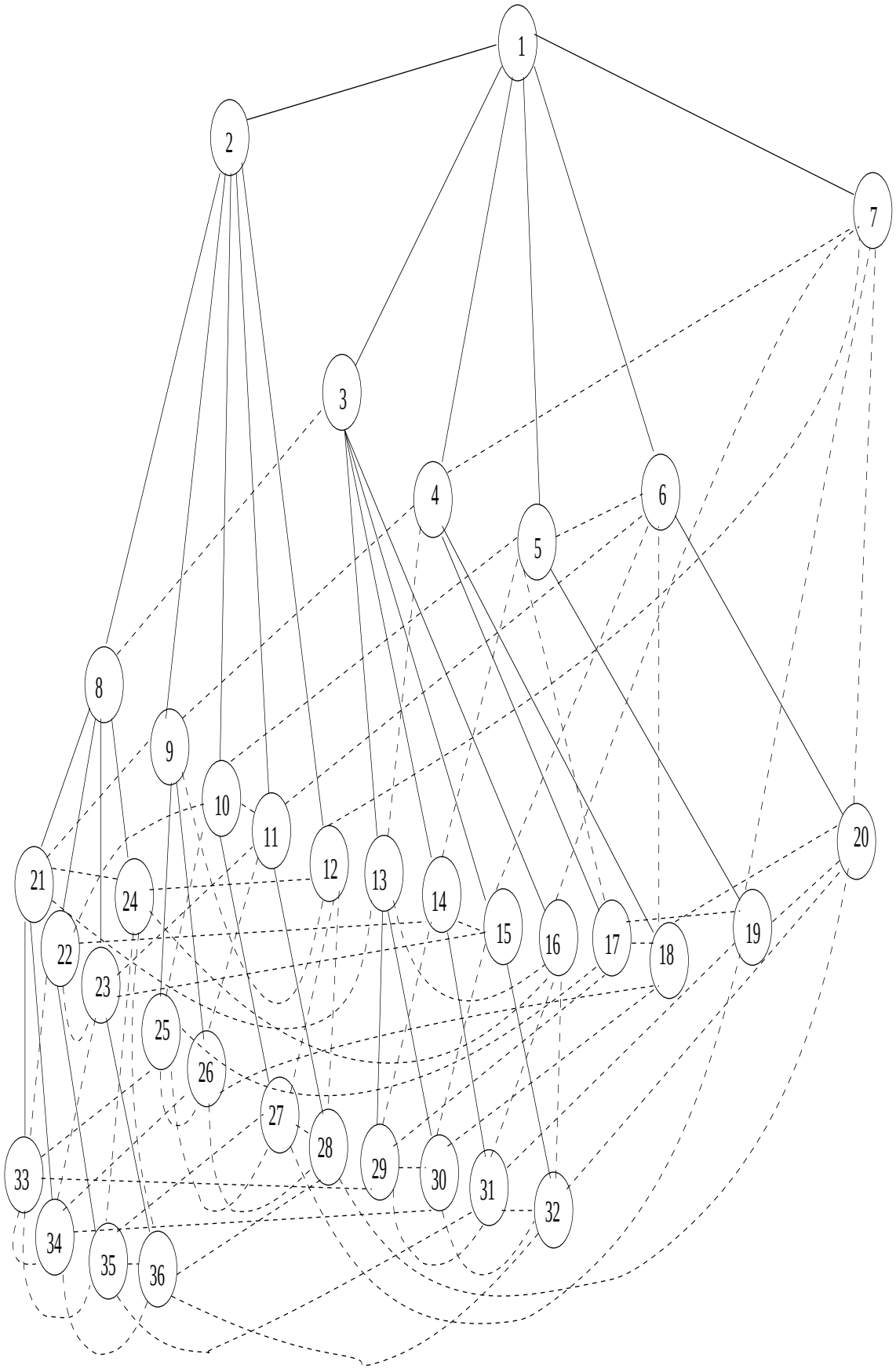


Figure 5.45: **The graph of spanning tree for the** 36 **vertices**

output edge details

else

< as before >. □

5.7 Degeneracy

It is well known that algorithms based on simplex pivoting experience difficulties in the presence of degeneracy. The primal simplex algorithm, for example, may cycle indefinitely, i.e. a sequence of basic feasible solutions giving the same value of the objective function may repeat unless degeneracy is dealt with [41]. It is to be expected therefore that BOP algorithms for vertex enumeration will be affected by degeneracy. There are two principal effects:

- the algorithm may fail because a feasible basis may have more than n adjacent bases, violating a fundamental assumption of BOP algorithms;
- multiple copies of the same vertex may be generated because there is now a many-one relationship between basic feasible solutions and vertices. This may be important if further processing of the vertices is to be undertaken.

The seriousness of these effects for a particular polyhedron is influenced by its degree of degeneracy. For a low degree of degeneracy explicit perturbation of the right hand sides, although inelegant, may be sufficient to resolve any practical difficulties with BOP algorithms. This is unlikely to be the case for a high degree of degeneracy. Unfortunately detecting in advance how degenerate a polyhedron is seems impossible. Indeed Dyer [24] has shown that deciding whether a polyhedron is not simple is NP-complete. Thus to be really useful BOP algorithms must be able to handle degeneracy and we discuss here modifications to the algorithm in section 5.4 to enable it to do so.

The first of the above effects can be resolved by finding a mechanism for ensuring the uniqueness of the simplex pivot in Step 4. Dyer [23] describes a randomised perturbation scheme which does so with probability 1, may be used with any BOP

algorithm and is not computationally expensive. We can incorporate this scheme in **EnumerateVertices** as follows:

- Prior to the start of the algorithm we calculate the perturbation vector h where
$$h_i = -\sum_{j=1}^{n} w_j a_{ji} - \sqrt{(\sum_{j=1}^{n} w_j^2 a_{ji}^2)}$$
and w_j is randomly generated;

- At Step 4,
$$k \leftarrow argmin\{\tfrac{V_{ri}}{a'_{ji}} : a'_{ji} > 0\}, \psi \leftarrow \{i : \tfrac{V_{ri}}{a'_{ji}} = \tfrac{V_{rk}}{a'_{jk}} \wedge a'_{ji} > 0\}$$
$$\text{If } |\psi| > 1, \quad \mathbf{SOLVE}(G(B_r), h, h'),\ k \leftarrow argmin\{\tfrac{h'_i}{a'_{ji}} : j \in \psi\}\ . \qquad \square$$

5.7.1 Redundancy

The method described by Dyer [23] for solving the problem of multiple copies (repetitions) of vertices relies on adjacency information, which is not stored in our algorithm. Chen et al. [10] have suggested using distance between degenerate vertices say, V_i and V_j; If $d = ||V_i - V_j|| < \epsilon$, for small $\epsilon > 0$ then $V_i \equiv V_j$. This requires storage and pairwise comparison of all degenerate BFS, which is relatively expensive. We suggest a new idea to be incorporated into step 3 in order to control redundancy problem as follows:

Solve-Redundancy:

In addition to storing β and γ for each vertex, we store δ and μ where $\mu_k \subseteq \beta_k$ and $\mu_k = \{j :\ j \in \beta_k,\ x_j \neq 0\}$ then:

$$\delta_k = \begin{cases} T & \text{if } V_k \text{ is degenerate and has been output} \\ F & \text{if } V_k \text{ is non-degenerate or a copy of a degenerate vertex which has already} \\ & \text{been output} \end{cases}$$

After determining V_r and μ_r from step 3 we add:

if $|\mu_r| = m$

Output V_r

$\delta_r \leftarrow F$

else

if $\mu_r \neq \mu_j \ \forall \ j \ s.t. \ \delta_j = T$

Output V_r

$\delta_r \leftarrow T$

else

$\delta_r \leftarrow F$. $\square$

Let us test the above modifications with an example from Dyer [23]. Determine all vertices to the system below:

$$x_1 + x_2 + 2x_4 \leq 2 \tag{5.75}$$

$$2x_1 + x_3 + x_4 \leq 2 \tag{5.76}$$

$$2x_2 + 2x_3 \leq 2 \tag{5.77}$$

$$\text{where } x_j \geq 0 \ (j = 1, 2, ..., 4)$$

We can write the system in the form of $Ax = b$, where:

$$A = \begin{bmatrix} 1 & 1 & 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.78}$$

$$x^T = [x_1, x_2, x_3, x_4, x_5, x_6, x_7] \tag{5.79}$$

$$b^T = [2, 2, 2] \tag{5.80}$$

$$\text{where } \ x_j \geq 0, \ (j = 1, 2, ..., 7)$$

Applying **EnumerateVertices** without modifications yields 15 basic feasible solutions as illustrated in **Table** 5.1:

Obviously, there are repetitions. Note that $|\mu_6| = |\mu_{10}| = |\mu_{11}| = |\mu_{12}| = |\mu_{13}| = |\mu_{14}| = |\mu_{15}| = 2 < 3$, meaning they are degenerate vertices. Whilst $|\mu_1| = |\mu_2| = |\mu_3| = |\mu_4| = |\mu_5| = |\mu_7| = |\mu_8| = |\mu_9| = 3$, meaning they are non-degenerate vertices . We therefore apply **Solve-Redundancy** as follows: when V_6 is generated it is found to be degenerate. As no other degenerate vertices have been found so

BFS	x_i	β	δ	μ	vertices
1	2,2,2	5,6,7	F	5,6,7	1
2	1,1,2	5,1,7	F	5,1,7	2
3	1,2,1	5,6,2	F	5,6,2	3
4	2,1,1	5,6,3	F	5,6,3	4
5	1,1,2	4,6,7	F	4,6,7	5
6	0,1,1	5,1,2	T	1,2	6
7	$\frac{3}{2},\frac{1}{2},1$	5,1,3	F	5,1,3	7
8	$\frac{2}{3},\frac{2}{3},2$	4,1,7	F	4,1,7	8
9	$\frac{1}{2},\frac{3}{2},1$	5,1,2	F	4,6,2	9
10	0,1,1	5,4,3	T	4,3	10
11	1,0,1	4,6,3	F	4,3	
12	0,1,1	4,1,2	F	1,2	
13	1,0,1	4,1,3	F	4,3	
14	0,1,1	2,4,3	F	4,3	
15	0,1,1	7,4,3	F	4,3	

Table 5.1: **Vertices of a degenerate polyhedron**

far, V_6 is output and $\delta_6 \leftarrow$ T. When V_{10} is generated, it is found to be degenerate and compared with V_6. As it is different it is output and $\delta_{10} \leftarrow$ T. When V_{11} is generated, μ_{11} is compared with μ_6 and μ_{10}. We find that $\mu_{10} = \mu_{11}$, so $\delta_{11} = F$ and V_{11} is not output. Similarly, $\mu_{13} = \mu_{14} = \mu_{15} = \mu_{10}$. When V_{12} is generated it is compared to μ_6, μ_{10} and we find $\mu_{12} = \mu_6$, so V_{12} is not output. Thus, there are 15 BFS but only ten distinct vertices.

5.8 Algorithm for 2 Non-zero per Row

We have seen from Proposition 5.2 that the basis structure for *LI(2)* is similar to that of its dual (2 non-zero per column), i.e. each component of the basis graph is either a tree or a graph with one cycle. Therefore, it seems likely that the vertices of *LI(2)* systems can be listed by a similar algorithm to that described in section

5.4.

5.9 Conclusion

In this chapter we have proved two important propositions which characterize the basis structure of *LI(2)* and dual *LI(2)* systems. Proposition 5.1 was used to develop a new BOP algorithm for enumerating vertices of simple, bounded polyhedra associated with dual *LI(2)* systems. An example of the application of this algorithm has been presented. Extensions to the algorithm to deal with unbounded edges and with degeneracy issues have been described.

Chapter 6

Counting Vertices of Polyhedra

6.1 Introduction

What are the differences between counting vertices and enumerating (listing) them? Is it possible to count vertices of general polyhedra in polynomial time? These questions will be explored in this chapter. We may recall in Chapter 1 that various vertex enumeration algorithms were reviewed, the most successful appeared to be those of Avis and Fukuda [4], Chen et al. [10] and Dyer [24]. However, these algorithms are inefficient with polyhedra that have a high degree of degeneracy. Provan [79] presented another algorithm that is considered to be an improvement on existing algorithms, his algorithm enumerates vertices of transportation and assignment polyhedra associated with Network LP structures in time *quadratic* in the number of vertices.

The difference between counting vertices of polyhedra and enumerating them is that counting problems are concerned with the exact or approximate number of vertices of the polyhedron, whilst vertex enumeration is concerned with listing the vertices.

Counting problems that can be solved exactly in polynomial time have few examples. Jerrum and Liptak [51] review two counting algorithms that exhibit such behaviour: the counting of spanning trees in connected loop-free undirected graphs and the number of perfect matchings in a planar graph. Therefore, research in this area is mainly concerned with problems of approximate counting and integration. The problems we address tend to be complete for the complexity class of counting problems known as $\#P$. Formally, a counting problem $f : \Sigma^* \longrightarrow \mathbb{N}$ is said to belong to the complexity class $\#P$ if there exists a polynomial time predicate $\chi : \Sigma^* \times \Sigma^* \longrightarrow \{0,1\}$ and a polynomial p such that $\forall x \in \Sigma^*$, $f(x) = |\{w \in \Sigma^* : \chi(x,w) \wedge |w| \leq p(|x|)\}|$. (See [45, 48, 51, 53]).

Once it is established that a counting problem is $\#P$-complete then the chances of developing a polynomial time algorithm for the exact counting (of the problem) is indeed very small. In such a situation attention should be focused on designing a polynomial-time algorithm for "*Approximate Counting*". We will define this precisely later. Approximate counting algorithms often proceed by simulating a suitable random walk on the set of interest, Ω, (e.g. set of feasible solutions to a combinatorial optimization problem). The random walk is said to be *rapidly mixing* if it gets close to the uniform distribution after a polynomial number of steps. If it can be shown that the counting problem is "*self-reducible*" and there exists a rapidly mixing time *Markov Chain* (MC) that has uniform distribution over Ω then *Markov Chain Monte Carlo* (MCMC) provides a polynomial-time algorithm for approximate counting.

MCMC provides an algorithm for the following general computational task. Let Ω be a very large (but finite) set of combinatorial structures (such as the set of possible configurations of a physical system, or the set of feasible solutions to a combinatorial optimization problem, such as *0-1*-Knapsack problems), and let π be a probability distribution on Ω (usually uniform). The task is to sample an element of Ω at random according to the distribution π. In addition to their inherent interest, combinatorial sampling problems of this kind have many computational

applications.

The remainder of this chapter is divided into four sections; section 6.2 deals with uniform sampling for approximate counting; section 6.3 deals with approximation using MCMC and other sections deal with new approximate counting methods for the vertices of certain polyhedra.

6.2 Uniform Sampling for Approximate Counting

We will review briefly (see Jerrum and Wagner [53]) how almost uniform sampling can be used for approximate counting, and survey how it might be achieved using Markov Chain simulation. One of the counting problems that can be approximated using uniform sampling and MC simulation is the *0-1*-Knapsack problem.

Example: Let $a = (a_1, a_2, ..., a_n)$ and $b \in \mathbb{N}$ (set of natural numbers). We estimate the number N of *0-1*-vectors, $x \in \{0,1\}^n$ satisfying the inequality $a.x = \sum_{i=1}^{n} a_i x_i \leq b$. If the vector a of size n (items) is to be packed into a Knapsack of capacity b, then the quantity to be estimated, $\Omega = \{x : ax \leq b\}$ can be interpreted as the number of combinations of items that can be fitted into the Knapsack, which we shall refer to as "*Knapsack solutions*".

Definition 1: A *Probabilistic Turing machine T* is one equipped with special coin tossing states. Each coin-tossing state q has two possible successor states qh and qt. When T enters state q, it moves on the next step to state qh with probability $\frac{1}{2}$ and to state qt with probability $\frac{1}{2}$.

There are various types of Probabilistic Turing machines which can be used to approximate functions with a high probability, leading to various randomised complexity classes.

Definition 2: A *randomised approximation scheme* (RAS) for the counting problem $f : \Sigma^* \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, is a randomised algorithm that takes as input an instance $x \in \Sigma^*$ and an error tolerance $\epsilon > 0$, and outputs a number $N \in \mathbb{N}$ (a random variable for 'coin tosses' made by the algorithm) such that, for every instance x, the following holds:

$$\Pr[e^{-\epsilon} f(x) \leq N \leq e^{\epsilon} f(x)] \geq \frac{3}{4} \tag{6.1}$$

A *fully polynomial randomised approximation scheme* (*fpras*) is a randomised approximation scheme that runs in time bounded by a polynomial in $|x|$ and ϵ^{-1}.

Remarks:

1. We can replace $\frac{3}{4}$ in (6.1) by any constant in the open interval $(\frac{1}{2}, 1)$.

2. (6.1) is essentially equivalent to $(1 - \epsilon)f(x) \leq N \leq (1 + \epsilon)f(x)$, and the requirement of a randomised approximation scheme is often specified in this fashion.

We define the *total variation distance* between probability distributions π and π' on a countable set Ω by

$$D_{tvd}(\pi, \pi') = \frac{1}{2} \sum_{w \in \Omega} |\pi(w) - \pi'(w)| = max_{A \subseteq \Omega} |\pi(A) - \pi'(A)| \tag{6.2}$$

Definition 3: A *sampling problem* is defined by a relation $S \subseteq \Sigma^* \times \Sigma^*$ between probability instances x and "solution" $w \in S(x)$ where $S(x) = \{w : xSw\}$.

Definition 4: An *almost uniform sampler* for a solution set $S \subseteq \Sigma^* \times \Sigma^*$ is a randomised algorithm that takes as input an instance $x \in \Sigma^*$ and sampling tolerance $\delta > 0$, and outputs a solution $W \in S(x)$ (a random variable over the 'coin tosses' made by the algorithm) such that the variation distance between the distribution of W and the uniform distribution on $S(x)$ is at most δ.

Definition 5: An almost uniform sampler is *fully polynomial* if it runs in time bounded by a polynomial in x and $\log(\delta)$. We abbreviate fully-polynomial almost

uniform sampler as *fpaus*.

Remarks:

1. There is no conflict in the definitions of *fpras* and *fpaus*. The running-time dependence on the tolerance (ϵ or δ, respectively) is not the same: it is polynomial in ϵ^{-1} versus $\log(\delta)$ respectively.

2. The difference between exact and almost uniform sampling seems to be less important. However, some technical difficulties arise when one attempts to define exactly uniform sampling: e.g. when $|S(x)| = 3$, if we use the probabilistic Turing machine as our model of computation. (See [92])

In the case of *self-reducible* problems (see Jerrum et al. [52]) approximate counting can be reduced to almost uniform sampling by using a witness-checking predicate χ, and we consider the associated counting and sampling problems, $f : \Sigma^* \to \mathbb{N}$ as:

$$f(x) = |\{w \in \Sigma^* : \chi(x, w) \wedge |w| \leq p(|x|)\}| \tag{6.3}$$

where $\mathbb{N}$ is a set of natural numbers, and $S \subseteq \Sigma^* \times \Sigma^*$. The computational complexity of approximating $f(x)$ and sampling almost uniformly from $S(x)$ are closely related. In particular, f admits an *fpras* if and only if S admits an *fpaus*. Note that this is not true in general (see Dyer et al. [20]). The most powerful technique for approximate sampling seems to be Markov Chain Monte Carlo.

Definition 6: A sequence $(X_t \in \Omega)_{t=0}^{\infty}$ of random variables is a Markov Chain (MC) with state space Ω, if:

$$\Pr[X_{t+1} = y | X_t = x_t, X_{t-1} = x_{t-1}, ..., X_0 = x_0] = \Pr[X_{t+1} = y | X_t = x_t] \tag{6.4}$$

$\forall\ t \in \mathbb{N}$ and $\forall\ x_t, x_{t-1}, ..., x_o \in \Omega$. Equation (6.4) encompasses the *Markovian Property*, where the MCs' antecedents prior to time t are forgotten. We are interested in *time-homogeneous* MCs, i.e., ones for which the right-hand side of (6.4) is independent of t; in this case, we set:

$$P(x, y) := \Pr[X_{t+1} = y | X_t = x] \tag{6.5}$$

define the *transition matrix* of the MC. The t-step transition probabilities P^t are then given inductively by:

$$P^t(x,y) := \begin{cases} I(x,y) & \text{if } t = 0 \\ \sum_{y' \in \Omega} P^t(x,y')P(y',y) & \text{if } t > 0 \end{cases}$$

where I is the identity matrix $I(x,y) = \delta_{xy}$. In another words, $P^t(x,y)[X_t = y | X_0 = x]$ and P^t implies the same matrix multiplications. A *stationary distribution* of an MC with transition matrix P is a probability distribution $\pi : \Omega \to [0,1]$ satisfying:

$$\pi(y) = \sum_{x \in \Omega} \pi(x)P(x,y) \tag{6.6}$$

Thus, if X_0 is distributed over π then so is X_1 (indeed, so is X_t $\forall t \in \mathbb{N}$). A finite MC always has at least one stationary distribution. An MC is *irreducible* if for all $x, y \in \Omega$ there exists a $t \in \mathbb{N}$ such that $P^t(x,y) > 0$; it is *aperiodic* if $gcd\{t : P^t(x,x) > 0\} = 1$ $\forall x \in \Omega$ and P^t implies the usual matrix multiplication. A finite-state MC is *ergodic* if it is both irreducible and aperiodic. An ergodic MC has a unique stationary distribution π; moreover the MC approaches π in the sense that $P^t(x,y) \to \pi(y)$ as $t \to \infty$ for all $x \in \Omega$, [53]. Informally, an ergodic MC eventually "forgets" its starting state. Computation of the stationary distribution is often facilitated by the following lemma.

Lemma 1 (*Jerrum* and *Sinclair*): Suppose P is the transition matrix of an MC. If a distribution $\pi' : \Omega \to [0,1]$ satisfies detailed balance w.r.t. P,

$$\pi'(x)P(x,y) = \pi'(y)P(y,x), \forall\ x, y \in \Omega \text{ and } \sum_{x \in \Omega} \pi'(x) = 1 \tag{6.7}$$

then, π' is a stationary distribution of the MC. If the MC is ergodic, then clearly, $\pi' = \pi$ is the unique stationary distribution.

Proof :

See [53], we just need to check that π' is invariant. Suppose X_0 is distributed as π'. Then,

$$\Pr(X_1 = y) = \sum_{x \in \Omega} \pi'(x)P(x,y) = \sum_{x \in \Omega} \pi'(y)P(y,x) = \pi'(y) \tag{6.8}$$

QED.

Remarks:

An MC for which (6.7) holds is said to be *time-reversible*. Clearly, Lemma 1 cannot be apply to non-time-reversible MCs. In practice, this is not a problem, since the MCs we consider are time reversible. It is difficult in general even to determine the stationary distribution of large non-time-reversible MCs, unless there is some special circumstance, e.g. symmetry, that can be taken into consideration. All the usual methods for constructing MCs with specified stationary distributions produce time-reversible MCs.

6.3 Approximation using Markov Chain Monte Carlo Method (MCMC)

Jerrum and Sinclair [48] have correctly observed that there is "no efficient deterministic algorithm which is known to accurately count Knapsack solutions and there is convincing complexity-theoretic evidence that none exists". In fact, Valiant [97] and Garey and Johnson [37] have even shown that the problem of counting Knapsack solutions is $\#P$-complete. Hence the need for approximations. A simple (non-polynomial time) method is based on selecting uniformly at random (u.a.r) a $0,1$-vector, $x \in \{0,1\}^n$ from the set of extreme points (corners) of an n-dimensional boolean hypercube. If $a.x \leq b$, then return 2^n solutions, but return 0 otherwise. The outcome is a random variable with an exact *expectation* N (number of $0,1$-vectors).

Suppose $\Omega = \{x \in \{0,1\}^n : \ a.x \leq b, \ a \in \mathbb{N}^n \ , \ b \in \mathbb{N}\}$ is the set of all solutions (quantities to be estimated) to the Knapsack problem. The Markov Chain $\mathcal{M}_{knap}$ over Ω is defined to be a random walk on the boolean hypercube truncated by the hyperplane $H' = \{x : a.x = b\}$. Jerrum and Sinclair [48] have shown that provided $\mathcal{M}_{knap}$ is "rapidly mixing", i.e. close to a static (stationary) state, and if the *Markov Chain Simulation* or total number of trials is assumed to be $17\epsilon^{-2}n^3$ (polynomial

in input length n, and ϵ^{-1}, for $0 < \epsilon \leq 1$) then:

$$P[(1 - \frac{\epsilon}{2})|\Omega(b)|^{-1} \leq Z \leq (1 + \frac{\epsilon}{2})|\Omega(b)|^{-1}] \geq \frac{3}{4} \tag{6.9}$$

which is an *fpras* for the problem, where Z(random variable, similar to N in (6.1)) is the product of the sample means of the trial.

The problem of whether $\mathcal{M}_{knap}$ is *rapidly mixing* for the *0-1*-Knapsack problem has been resolved recently. Morris and Sinclair [70] have provided an *fpras* for the problem. The Markov Chain Simulation approach described above is also used by Dyer et al. [33].

We will later outline an approximation scheme for counting the vertices of *0-1*-Knapsack polytopes.

6.4 Approximately Counting Vertices of Polyhedra

We have seen that it is possible to approximately count using uniform sampling. Vertex counting for general polyhedra has been shown to be NP-hard by Dyer [24]. As such it is of importance to explore the possibilities of exact or approximate counting vertices of polyhedra. Is it also $\#P$-complete?

The basic problem is to count the vertices of a polyhedron P defined by linear inequalities:

$$P = \{x : \sum_{j=1}^{n} a_{ij}x_j \leq b_i,\ i = 1, 2, ...m\} \tag{6.10}$$

Like many other counting problems, this has been shown to be $\#P$-hard by Dyer [24] and a survey is available in [45, 48]. Therefore, we attempt to develop a fully-polynomial randomised approximation scheme (*fpras*). For simplicity, we divide the problems into two classes: (1) polyhedra with polynomially many inequalities and (2) polyhedra with exponentially many inequalities.

6.5 Polyhedra with Polynomially Many Inequalities

In this case we are concerned with approximation algorithms for polyhedra that have non-exponential growth in the number of inequalities and that are polynomial in the number of variables n. We will present four different sub-classes of this type, polyhedra associated with: (I) *0-1*-Permanent, (II) *Down-Sets* in a Partial Order, (III) *0-1*-Knapsack problem and (IV) $2 \times n$ transportation problem.

6.5.1 Counting vertices of polyhedra associated with *0-1*-Permanent

What is meant by *0-1-Permanent*?

Definition 7: The permanent of an $n \times n$ non-negative matrix $A = (a(i,j))$ is defined as:

$$per(A) = \sum_{\pi} \prod_{i} a(i, \pi(i)) \tag{6.11}$$

where the sum is over all permutations π of $\{1, 2, ..., n\}$. If A is a $0, 1$ matrix, then the permanent is called *0-1*-Permanent. This can then be visualized as an adjacency matrix of a bipartite graph $G_A = (V_1, V_2, E)$. It is clear that the permanent of A can then be equivalent to the number of perfect matchings in G_A.

The problem of counting *0-1*-Permanent was proved to be $\#P$-complete by Valiant [96]. Since the discovery of this result focus has shifted to finding efficient approximation algorithms that may have precise performance guarantees.

Suppose we are to count the vertices of a polyhedron formed by:

$$\sum_{i=1}^{n} a_{ij} x_{ij} = 1, (j = 1, 2, ..., n) \tag{6.12}$$

$$\sum_{j=1}^{n} a_{ij} x_{ij} = 1, (i = 1, 2, ..., n) \tag{6.13}$$

$$x_{ij} \geq 0$$

Jerrum et al. [49] have presented a polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. In particular, after several attempts, Jerrum et al. [49] have resolved the question of existence of an *fpras* for the permanent of a general 0, 1-matrix. The main result of their paper can be summarised in the following theorem:

Theorem 1 (*Jerrum*, *Sinclair* and *Vigoda*): There exists a fully-polynomial randomised approximation scheme for the permanent of an arbitrary $n \times n$ matrix A with non-negative entries.

In the *0-1* case, the above result can be interpreted as counting approximately the number of *0-1*-vertices of the above polyhedron. This can be summarised by the following claim and proposition, which we present without proof.

Claim 6.1: There is a *1-1* correspondence between the number of perfect matchings and the *0-1*-vertices of the polyhedra formed from (6.12) and (6.13).

Proposition 6.1: There exists a fully-polynomial randomised approximation scheme for counting the *0-1*-vertices of a polyhedron formed by the permanent of (6.12) and (6.13).

6.5.2 Partial order (PO)

In this case we will be concerned with two situations: (a) Down-Sets in a Partial Order and (b) Independent sets in a bipartite graph.

(a) Down-Sets in a Partial Order (PO)

A binary relation R on a set A is called *reflexive*, if , $\forall\ a \in A$, aRa. A relation R on A is called *transitive*, if for $a, b, c\ \in A$, aRb and $bRc \Rightarrow aRc$. A relation R on A is called *antisymmetric*, if for any $a, b\ \in A$, aRb and $bRa \Rightarrow a = b$.

Definition 8: A relation R on a set A is called a partial order if it is reflexive, antisymmetric and transitive.

An example is $\leq$ on the set of natural numbers.

Definition 9: Suppose we have a partial order (PO) on the set $U = \{1, 2, ..., n\}$. A Down-Set S is a subset, $S \subseteq U$, such that if $j \in S$ and $i \leq j$ then $i \in S$ (where $\leq$ is the given partial order).

Definition 10: An $m \times n$ integral matrix A is *totally unimodular* (TU) if the determinant of each square submatrix of A is equal to 0,1, or -1.

It is clear from definition 10 above that we must have $a_{ij} = 0, 1$, or -1 if A is TU; that is, A is a $(-1, 0, 1)$ matrix.

For a given partial order $\leq$, consider the polyhedron in $\mathbb{R}^n$:

$$P^{(0,1)} = \{x \ : x_i \leq x_j \text{ if } i \leq j, \text{ where } 0 \leq x_k \leq 1, \ \forall k\}. \tag{6.14}$$

The coefficient matrix, $A^{(0,1)}$ is a two per row matrix, each with one $+1$ and one -1.

The problem of counting all vertices of $P^{(0,1)}$ is also known to be $\#P$-complete. So we can only attempt to develop an *fpras* to do approximation. To do this we need to show that all vertices of $P^{(0,1)}$ are *0-1*. To do this we show that the co-efficient matrix $A^{(0,1)}$ of $P^{(0,1)}$ is TU. We use some results from Nemhauser and Wolsey [75].

Proposition 6.2 (*Nemhauser* and *Wolsey*): If the $(-1, 0, 1)$ matrix A has no more than two nonzero entries in each column, and if $\sum_i a_{ij} = 0$ if column j contains two nonzero coefficients, then A is TU.

Proposition 6.3 (*Nemhauser* and *Wolsey*): If A is TU, then the transpose A^T is TU.

It is worth noting that a network matrix has 2 non-zeros per each column, one is $+1$ and the other is -1. By Proposition 6.2 it is TU. From the above, we have:

Proposition 6.4: If the $(-1, 0, 1)$ matrix B has no more than two nonzero entries in each row and each row contains one $+1$ and one -1, then B is TU.

If the coefficient matrix of a polyhedron is TU, then the polyhedron has integral vertices.

Proposition 6.5 (*Nemhauser* and *Wolsey*): If A is TU, then the polyhedron $P(b) = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ has integral vertices for all $b \in Z^m$ for which it is not empty.

Proof :
We consider the linear program with constraint set $Ax + Iy = b$, $x \in \mathbb{R}^n_+$ and $y \in \mathbb{R}^m_+$, where A is TU and b is integral. Let $(A, I) = (A_B, A_N)$, where A_B is a basis matrix for the linear program. But a matrix obtained by a pivot operation on A is TU, which means A_B^{-1} is an integral matrix. Thus, $A_B^{-1}b$ is integral, so the correspondence between basic feasible solutions and vertices yields the result. QED

We can combine Propositions 6.4 and 6.5 to get the following result.

Proposition 6.6: If B has at most two nonzero entries in each row, at most one $+1$ and one -1, then it is TU, and the polyhedron $P^B = \{x \in \mathbb{R}^n_+ : Bx \leq b\}$ is integral for all $b \in Z^m$ for which it is not empty.

Hence, we have:

Proposition 6.7: For a given partial order, the polyhedron $P^{(0,1)}$ is integral with *0-1* vertices.

Theorem 2: There is a *1-1* correspondence between Down-Sets and vertices of $P^{(0,1)}$.

Proof :
If S is a Down-Set, let the vector x^s be defined by: $x_i^s = 1$ if $i \in S$, $x_i^s = 0$ otherwise. Since S is a Down-Set, x^s lies in $P^{(0,1)}$. Also, it exactly satisfies n of the inequalities $0 \leq x_i \leq 1$, and hence is a vertex. Clearly this argument can be reversed, since all vertices of $P^{(0,1)}$ have coordinates 0 or 1. QED

Let us examine the relation between Down-Sets and Independent Sets in bipartite graphs.

(b) Independent Set (IS)
Definition 11: An Independent Set (IS) of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in E is incident on at most one vertex in V'.

The problem of computing a *maximum* independent set in a graph is a NP-hard problem. (See [37]).

A (directed) bipartite graph G is a graph whose nodes are partitioned into two sets such that all the arcs in the graph are directed from a node in the first set (known as an origin) to a node in the second set (known as a destination). The graph is complete if all possible arcs are present. Figure 6.1 shows a bipartite graph G with a partial order (PO) and the relationship between an Independent set (IS) and the corresponding Down-Set S. The partial order (PO) on the graph S such that $a < e$ because there is an arc between a and e. Similarly, $b < e$, $b < f$, $c < f$, $c < g$, $d < g$, and $d < h$. For example the IS can be chosen to be $\{a, f, g\}$ which is complemented on the bottom to give the Down-Set $\{b, c, d, f, g\}$.

Dyer and Greenhill [21] have shown that the problem of counting Independent sets

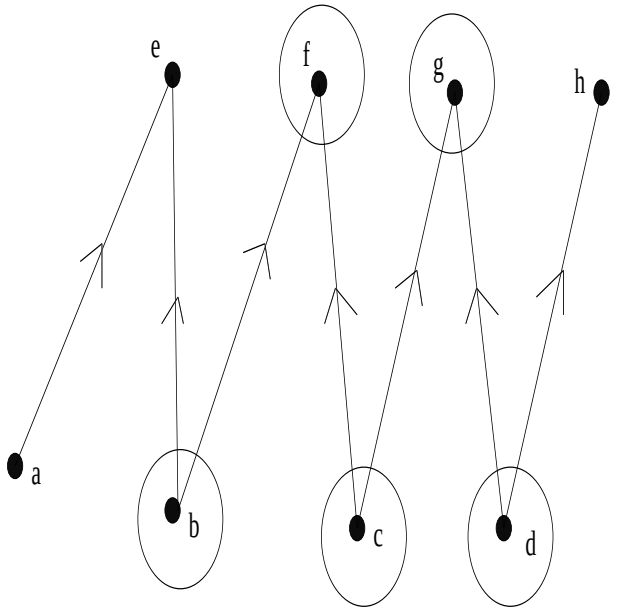


Figure 6.1: **Complementary relation between elements of *IS* and *S***

in graphs with maximum degree 3 is $\#P$-complete. Luby and Vigoda [62] describe a Markov Chain for Independent sets in graphs of maximum degree 4 which is rapidly mixing.

Let us investigate how we can develop an *fpras* for counting vertices of polyhedra associated with Independent sets in bipartite graphs.

Claim 6.2: Any Independent Set (IS) in a bipartite graph G can be complemented to give a Down-Set S in the associated partial order.

Proof : (See Dyer et al. [32])
This follows directly from the fact that $\forall x \in IS$ there exists no $y \in IS$ such that $x < y$, i.e. there is no edge between x and y. Also $\forall y \in S$, $x \in S$ iff $x < y$ so there is no edge between y and any $x \notin S$. QED

From the above discussion and results, we see that the following polyhedron that has IS as its set of vertices is integral. Define the polyhedron associated with

Independent Sets (IS) in a bipartite graph $G = (V, E)$ as:

$$P^{IS} = \{x : \ x_i + y_j \leq 1, \ \forall \ (i,j) \in E, i = 1, ..., m, j = 1, ..., n\}$$

Corollary 1: If A^{IS} is a matrix associated with Independent set IS, then it is TU.

Proof :
For j on the bottom put $x_j = 1 - y_j$ $(j = 1, ..., m)$ then we have the polyhedron for the corresponding partial order. QED

Comments:
From Corollary 1 and Proposition 6.7 we conclude that the Down-Sets and Independent Sets are sets of integral vertices. The matrices associated with polyhedra whose sets of solutions (vertices) are Down-Sets or independent sets are TU, because they have at most two nonzero per row/column that are $+1$ and -1. It is not yet known whether there is an *fpras* for Down-Set or Independent Set. The concluding result of this subsection can be presented below.

Open Problem: Is there an *fpras* for counting Down-Sets in a partial order or Independent Sets in a bipartite graph? (See Dyer et al. [32])

The next related problem is that of counting the vertices of *0-1*-Knapsack Problem. This has also resisted attack until recently.

6.5.3 *0-1*-Knapsack problem

The *0-1*-Knapsack problem, which is also called *binary* Knapsack problem, is that of finding all solutions to the system:

$$\sum_{i=1}^{n} a_i x_i \leq b \tag{6.15}$$

where a_i, b are integers and $x_i \in \{0, 1\}$. This problem is known to be $\#P$-complete (see [48, 50] for survey). The *0-1*-Knapsack problem has been quoted as an open problem in [33, 50] since it is among a small handful of canonical problems that

resisted attack. Recently, Morris and Sinclair [70] have solved the problem concerning the mixing-time of a symmetric random walk on the n-dimensional cube truncated by a hyperplane. They show that it is polynomial in n. As a result they successfully obtain an *fpras* for counting the feasible solutions of a *0-1*-Knapsack problem.

Figure 6.2 is a 3-dimensional illustration of *0-1*-hypercube truncated by a hyperplane. The problem in this case is trying to approximate the number of vertices that lie on the hyperplane.

Let $a = (a_i)_{i=1}^n$ and b be real numbers, then the set of solutions to (6.15) can be denoted by Ω:

$$\Omega = \{x : a'x \leq b\} \tag{6.16}$$

where $x = (x_i)_{i=1}^n$ is a *0-1*-vector and $a'x \equiv \sum_{i=1}^n a_i x_i$. From a geometrical viewpoint, Ω can be viewed as the set of vertices of the n-dimensional cube $\{0,1\}^n$ which lie on one side of the hyperplane $a'x = b$. From a combinatorial point of view Ω is the set of feasible solutions to the *0-1*-Knapsack problem defined by a and b. We can therefore denote a_i as the weights of a set of n items, and b as the capacity (weight limit) of a Knapsack. Then there is *1-1* correspondence between $x \in \Omega$ and subsets set of items X whose aggregated weight does not exceed the Knapsack capacity, given by $X = \{i : x_i = 1\}$. The weight of X can then be written as $a(X) = \sum_{i \in X} a_i$. The problem of computing $|\Omega|$ is $\#P$-complete, so the aim is to find a good approximation algorithm.

Virtually, all known approximation algorithms proceed by simulating a suitable random walk on the set of interest Ω. The walk is constructed so that it converges to a uniform distribution over Ω; simulation of the walk for sufficiently many steps therefore allows one to sample (almost) uniformly from Ω, and thus to approximate $|\Omega|$.

Morris and Sinclair [69, 70] have recently presented an *fpras* for computing $|\Omega|$ in

the general case, their result can be summarised as follows:

Theorem 3 (*Morris* and *Sinclair*): Let Ω be the set of solutions to an arbitrary instance of the *0-1*-Knapsack problem. The mixing time of the random walk on G_Ω is $\tau_{mix} = O(n^{4.5})$.

The results of Morris and Sinclair [70] can be adapted to solve the problem of counting all vertices of a polyhedron P, where $P = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j \leq b\}$ and $0 \leq x_j \leq 1$.

Thus, is there any approximation scheme for counting all vertices of P, including the non-integer vertices, in polynomial time?

We attempt to solve this problem. Let us assume that we are dealing with non-degenerate *0-1*-Knapsack Problems. Define the Knapsack polytope as:

$$P^{(NK)} = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j \leq b\} \tag{6.17}$$

where $0 \leq x_j \leq 1$, a_j and b are integers; $j = 1, 2, ..., n$. Clearly the above polyhedron $P^{(NK)}$ is non-degenerate. The method can easily be extended if $P^{(NK)}$ is degenerate.

The problem can be answered if there is a relationship between the number of non-integer and integer vertices of $P^{(NK)}$. The following observations answer this question.

Proposition 6.8: Let $P^{Kp} = \{x \in \mathbb{R}^n : \sum_{i=1}^{n} a_i x_i \leq b\}$ be a polyhedron where $0 \leq x_i \leq 1$. The number of non-integer vertices of P^{Kp} is less than or equal to n times the number of integer vertices.

Proof :
This is true as every non-integer vertex is adjacent on $P^{(Kp)}$ to an integer vertex, and every integer vertex has at most n neighbours. *QED*

Let InV be the number of integer vertices of $P^{(Kp)}$; $NonIV$ be the number of non-integer vertices and ToN be total number of vertices of $P^{(Kp)}$.

Theorem 4: $InV \leq ToN \leq (n+1)InV$.

Proof :
Clearly $InV + NonIV = ToN$, so $InV \leq ToN$.
Also, by Proposition 6.8, $NonIV \leq n \times InV$ so

$$ToN = InV + NonIV \leq InV + n \times InV$$
$$\Rightarrow ToN \leq InV(1+n)\ QED$$

Since the number of integer vertices can be approximated and the total is polynomially bounded by this, we can have the following conclusion.

Theorem 5: There exists an *fpras* for counting all vertices of non-degenerate *0-1*-Knapsack polyhedra.

Proof :
This follows from general results of Jerrum and Sinclair [48].

6.5.4 $2 \times n$ transportation polyhedra

In Chapter 2 we have implemented Provan's algorithm for enumerating vertices that are associated with Network LP structure. The algorithm of Provan was confirmed to be quadratic empirically. Here we will be concerned with counting vertices of transportation polyhedra. Like other polyhedra the problem of counting all vertices of transportation polyhedra is $\#P$-complete, see [24]. We attempt to explore how to approximately count vertices of a special class of transportation polytopes, known as $2 \times n$ transportation polyhedra. Let us briefly introduce some definitions that will be used later.

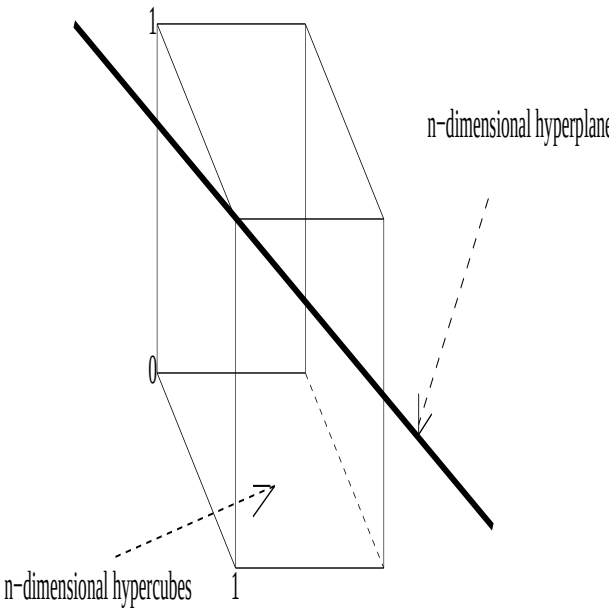


Figure 6.2: **A $3-D$ *0-1*-Knapsack cube truncated by a hyperplane**

Definition 12: Consider m origin points, where i has a supply s_i of units of a particular item (commodity). In addition, there are n destination points, where destination j requires d_j units of commodity. We assume that s_i, $d_j > 0$. Associated with each link (i, j), from origin i to destination j, there is a unit cost c_{ij} for transportation. The problem is to determine a feasible "shipping pattern" from origins to destinations that minimizes the total transportation cost. This problem is known as the Hitchcock or the transportation problem. We assume that the total supply equals total demand, i.e, the problem is balanced. In another words, the problem is that of counting vertices of:

$$P^{(transp)} = \{x : A^{(transp)}x = b\} \tag{6.18}$$

where $A^{(transp)}$ is a $(m+n) \times (m \times n)$ node-arc incidence matrix, it is a 2 nonzero per column matrix, each nonzero is either $+1$ and -1. As we said earlier, the problem of counting vertices of $P^{(transp)}$ is $\#P$-complete, (Dyer [24]).

Definition 13: A contingency table is a matrix of non-negative integers with prescribed positive row and column sums. In a more technical term, let $r = (r_1, r_2, ..., r_m)$ and $s = (s_1, s_2, ..., s_n)$ be two positive integer partitions of the posi-

tive integer $\mathbb{N}$. The set $\Sigma_{r,s}$ of contingency tables with these row and column sums is defined by:

$$\Sigma_{r,s} = \{Z \in \mathbb{N}_0^{m \times n} : \sum_{j=1}^{n} Z_{ij} = r_i, for\ 1 \leq i \leq m, \sum_{i=1}^{m} Z_{ij} = s_j, for\ 1 \leq j \leq n\} \tag{6.19}$$

The problem of approximately counting the number of contingency tables with given row and column sums is known to be $\#P$-complete even when m, n equals 2 [22]. Hence the need for an approximation. Dyer and Greenhill [26] have employed a path coupling method and show that the MC for 2 row contingency tables is rapidly mixing. Cryan and Dyer [15], Cryan et al. [16, 17] and Dyer [31] show a similar result for any fixed m. The papers [16, 17, 31] were written after the work for this thesis was completed.

The following results give upper bounds for approximately counting contingency table. Let $\mathcal{M}(\sum_{r,s})$ denote a Markov chain with state space $\sum_{r,s}$. If X_t is the state of the chain in time t then at time $t+1$ the state is determined by:

- Select (j_1, j_2) u.a.r. such that $1 \leq j_1 \leq j_2 \leq n$,

- Choose $x \in T_X(j_1, j_2)$ u.a.r. and let,

$$X_{t+1}(k,j) = \begin{cases} x(k,l) & \text{if } j = j_l \text{ for } l \in \{1,2\} \\ X_t(k,j) & \text{otherwise} \end{cases}$$

Clearly $\mathcal{M}(\sum_{r,s})$ is aperiodic and the following is due to Dyer and Greenhill [26].

Theorem 6 (*Dyer* and *Greenhill*): Let $r = (r_1, r_2)$ and $s = (s_1, s_2, ..., s_n)$ be two positive integer partitions of the positive integer N. The Markov chain $\mathcal{M}(\sum_{r,s})$ is rapidly mixing with mixing time $\tau(\epsilon)$ satisfying

$$\tau(\epsilon) \leq \frac{n(n-1)}{2} \log(N\epsilon^{-1})$$

Note that since the Markov Chain is rapidly mixing then there can be an *fpras* for the counting problem.

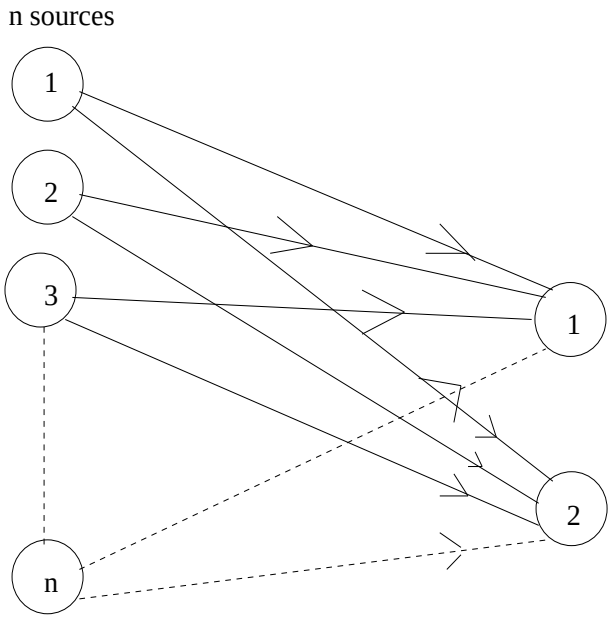


Figure 6.3: **Graph of a $2 \times n$ transportation problem**

Theorem 7 (*Dyer* and *Greenhill*): Let $\tau(\epsilon)$ denote the mixing time of the Markov chain $\mathcal{M}(\sum_{r,s})$. Then

$$\frac{n(n-1)}{6} \log(\frac{n-1}{8}) \leq \tau(\epsilon)$$

We will explore the problem of counting vertices of $2 \times n$ transportation polyhedra. Figure 6.3 is a graph of a $2 \times n$ transportation problem, and 6.4 is a graph of $m \times 2$ transportation problem.

6.5.5 Approximately counting vertices of $2 \times n$ general transportation polytopes

As we have stated earlier the coefficient matrices of $2 \times n$ transportation polytopes are identical to those of contingency tables. Here, we are interested on how to approximately count (by means of an *fpras*) vertices of transportation polyhedra

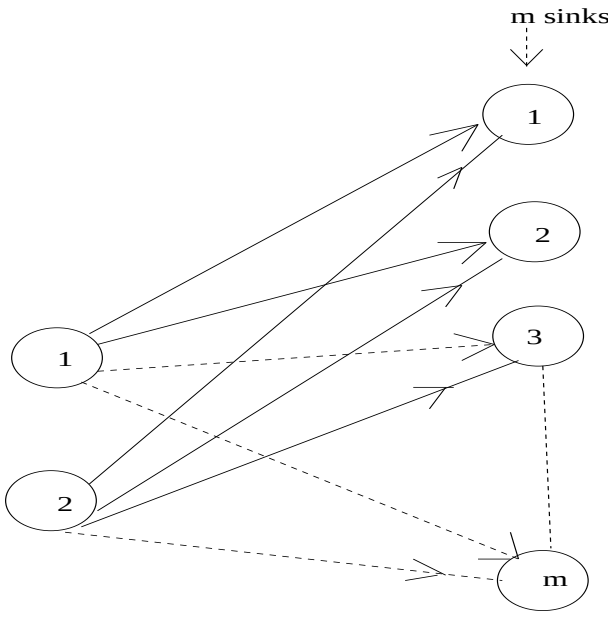


Figure 6.4: **Graph of an $m \times 2$ transportation problem**

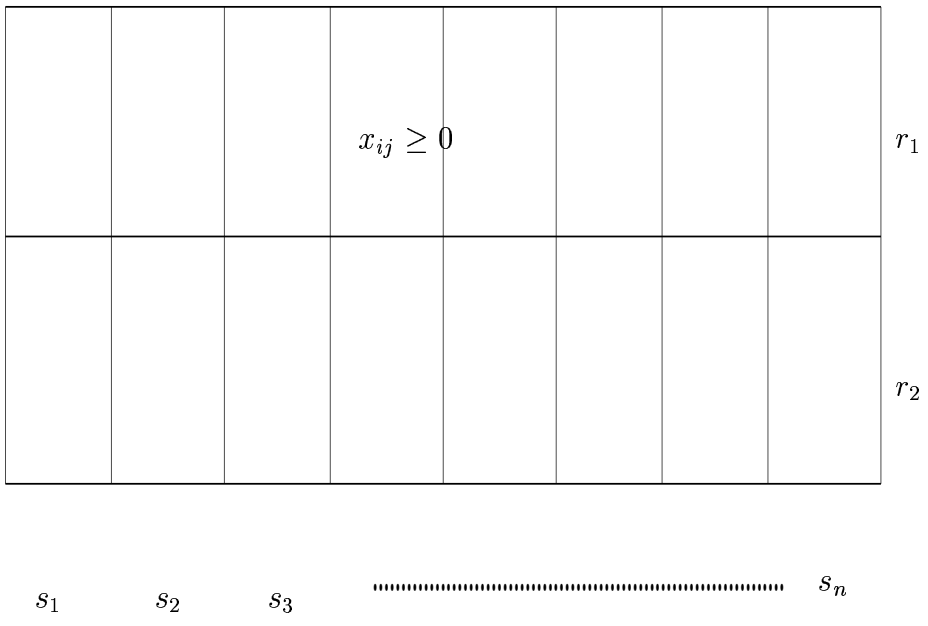


Figure 6.5: **Table of a $2 \times n$ general transportation problem**

formed by the constraints:

$$\sum_{j=1}^{n} x_{ij} = r_i, (i = 1, 2) \tag{6.20}$$

$$x_{1j} + x_{2j} = s_j \quad (j = 1, 2, ..., n) \tag{6.21}$$

$$x_{ij} \geq 0 \tag{6.22}$$

where r_1 and r_2 are the two sources (supplies) and $s_1, s_2, ..., s_n$ are the destinations (demands). We assume that the sum of supply and sum of demand is the same and denoted by N, and x_{ij} are shipments from supply i to destination j.

We assume without loss of generality that $r_1 \leq r_2$ and $s_1 \leq s_2 \leq s_3..... \leq s_n$. Let us transform the above problem into one similar to the Knapsack problem and attempt to find a scheme for approximate counting.

Let

$$y_j = \frac{x_{1j}}{s_j}, \tag{6.23}$$

then $0 \leq y_j \leq 1 (j = 1, 2, ..., n)$.

Substituting (6.23) into (6.20) we have,

$$\sum_{j=1}^{n} s_j y_j = r_1 \tag{6.24}$$

From (6.23), $x_{1j} = s_j y_j$ and from (6.21) we have:

$$x_{2j} = s_j - x_{1j} = s_j - s_j y_j = s_j (1 - y_j) \tag{6.25}$$

As such, for $i = 2$ and $r_1 + r_2 = N$ we have,

$$\sum_{j=1}^{n} s_j (1 - y_j) = r_2 \Longleftrightarrow \sum_{j=1}^{n} s_j y_j = N - r_2 = r_1 \tag{6.26}$$

Let us choose y_n as a slack variable. Then from (6.26) we have

$$s_n y_n + \sum_{j=1}^{n-1} s_j y_j = r_1$$

or,

$$s_n y_n = r_1 - \sum_{j=1}^{n-1} s_j y_j \tag{6.27}$$

or

$$y_n = \frac{r_1 - \sum_{j=1}^{n-1} s_j y_j}{s_n} \tag{6.28}$$

From $0 \leq y_n \leq 1$, we can have two cases, case (a) for $0 \leq y_n$ we have:

$$y_n \geq 0 \Rightarrow \frac{r_1 - \sum_{j=1}^{n-1} s_j y_j}{s_n} \geq 0 \Rightarrow \sum_{j=1}^{n-1} s_j y_j \leq r_1 \tag{6.29}$$

case (b) $y_n \leq 1$ we can have,

$$y_n \leq 1 \Rightarrow \frac{r_1 - \sum_{j=1}^{n-1} s_j y_j}{s_n} \leq 1 \Rightarrow r_1 - s_n \leq \sum_{j=1}^{n-1} s_j y_j \tag{6.30}$$

combining (6.29) and (6.30) we have:

$$r_1 - s_n \leq \sum_{j=1}^{n-1} s_j y_j \leq r_1 \tag{6.31}$$

where $0 \leq y_j \leq 1$ for $j = 1, 2, ..., n$, (6.31) resembles the Knapsack problem except for the fact that it has a lower as well as an upper bound. The gap between the upper and lower bounds in (6.31) can itself be bounded as follows:

We know that $r_1 + r_2 = N$ and also $s_1 + s_2 + ... + s_n = N$. Thus, we have $r_1 \leq \frac{N}{2}$ since it is the smallest row sum and $s_n \geq \frac{N}{n}$ since it the largest column sum. Combining these results we have,

$$2r_1 \leq N \leq ns_n \Rightarrow 2r_1 \leq ns_n \Rightarrow s_n \geq \frac{2}{n} r_1 \tag{6.32}$$

so we have;

$$r_1 - s_n \leq r_1 - \frac{2}{n} r_1 = (1 - \frac{2}{n}) r_1. \tag{6.33}$$

The polyhedron is sketched in Figure 6.6, where:

$$a_1 = |\{y \in \{0,1\}^{n-1} : \sum s_j y_j < r_1 - s_n\}| \tag{6.34}$$

$$a_3 = |\{y \in \{0,1\}^{n-1} : \sum s_j y_j > r_1\}| \tag{6.35}$$

and

$$a_2 = 2^n - a_1 - a_3 \tag{6.36}$$

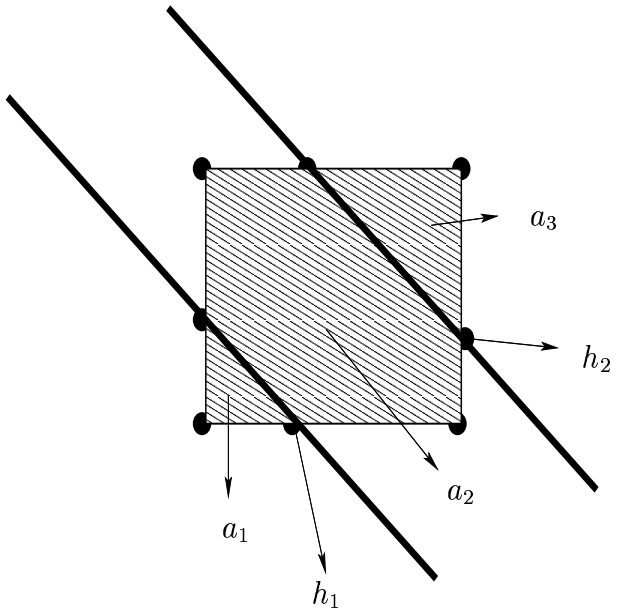


Figure 6.6: **A hypercube truncated by two hyperplanes**

Figure 6.6 consists of two hyperplanes that truncate a hypercube, which we may conveniently call the lower and upper bounds respectively. Let h_1 be the number of vertices on the lower hyperplane and h_2 be the number of vertices on the upper hyperplane.

We can now derive an *fpras*. Let a_1 denote the number of vertices of the Knapsack problem bounded above by the lower hyperplane whose number of vertices is represented by h_1. Let a_3 represents the number of vertices of a Knapsack problem bounded below by the upper hyperplane whose number of vertices is h_2. Let a_2 denote the number of integer vertices that lie between the lower and upper hyperplanes (these are represented in (6.34), (6.35) and (6.36)). It is possible to approximate a_1, a_3, h_1, h_2, $h_2 + a_1 + a_2$ and $h_1 + a_2 + a_3$ since these are all associated with Knapsack problems or Knapsack Polytopes.

We wish to approximate the number of vertices $h_1 + a_2 + h_2$ of the "*middle*" polytope (i.e. the approximate number of integer vertices bounded below by the lower hyperplane and above by the upper hyperplane combined with approximate number of vertices on both the hyperplanes.)

We will use Harper, Lindsey, Bernstein and Hart's theorem which is cited in [61] to deduce our result. Let Q_n be the discrete cube and A be a subset of Q_n. Let $\partial_e A$ be the edge boundary i.e. the number of edges of Q_n which have exactly one end-point in A.

Theorem 8(*Harper et al.*): Let $A \subset Q_n$ and I be the set of the first $|A|$ elements of Q_n in binary order (an order β is binary if for any $x, y \in A$, x precedes y if $max(x \triangle y) \in y$, i.e. the greatest element which is one of x and y but not the other is actually in y). Then $|\partial_e A| \geq |\partial I|$. In particular, if $|A| = 2^r$ then $|\delta_e A| \geq 2^r(n-r)$. This implies that, if $|A| \leq 2^{n-1}$ then $|\partial_e A| \geq |A|$.

We can relate to above theorem with the notations we have been using, that is, in the non-degenerate case, h_1 is equivalent to $|\partial_e A|$. Since a_3 and a_1 are mirror images, by symmetry we can assume without loss of generality $a_3 \geq a_1$, but since $a_1 + a_2 + a_3 = 2^n$ it follows that $a_1 \leq 2^n - a_1$, i.e. $a_1 \leq 2^{n-1}$. So, in the non-degenerate case:

$$h_1 \geq a_1, \ and, \ h_1 \leq n a_1 \tag{6.37}$$

Since we can approximate $h_2 + a_1 + a_2$ we can use this to get the desired result, from (6.37), i.e. provided $n \geq 4$:

$$h_2 + h_1 + a_2 \geq h_2 + a_1 + a_2 \tag{6.38}$$

Also

$$h_2 + h_1 + a_2 \leq h_2 + n a_1 + a_2 \leq n(h_2 + a_1 + a_2) \tag{6.39}$$

Thus, the number of vertices within or on the lower and upper hyperplanes is polynomially related to the quantity $h_2 + a_1 + a_2$ which is approximately countable by an *fpras*. It now follows from results of Sinclair and Jerrum [48] that $h_2 + h_1 + a_2$ has an *fpras* (note that the required self-reducibility property is satisfied here by the $\{0, 1\}$ constraints).

In the degenerate case, we still have $h_1 \leq na_1$ and $h_1 \geq \frac{a_1}{n}$, so the results follows similarly.

The method cannot easily be extended to work for $n = 3$ or $m = 3$ or higher fixed values because the relationship to the Knapsack problem breaks down.

6.6 Polyhedra with Exponentially Many Inequalities

Matroids and submodular functions (functions over sets that satisfy some properties) are known to be building blocks of some combinatorial optimization problems whose generalizations are both network flow problems and the spanning tree problems.

Matroids are also considered to be prototypes of independence systems and *0-1*-integer programs that have special properties which could be used to obtain efficient algorithms for the corresponding optimization problems. Let us review some definitions.

Definition 14: Let $S = \{1, 2, ..., n\}$ be a finite set and let $\mathcal{F}$ be a set of subsets of S. $\mathcal{I} = (N, \mathcal{F})$ is an *independence system* if $F_1 \in \mathcal{F}$ and $F_2 \subseteq F_1 \Longrightarrow F_2 \in \mathcal{F}$. Elements of $\mathcal{F}$ are called independent sets, and the remaining subsets of S are called dependent sets.

Definition 15: Given an independence system $\mathcal{I} = (S, \mathcal{F})$ we say that $F \in \mathcal{F}$ is a *maximal independent set* if $F \bigcup \{j\} \notin \mathcal{F}$, $\forall j \in S \backslash F$. A maximal independent set T is *maximum* if $|N| \leq |T|$ $\forall N \in \mathcal{F}$. We denote $m(T) = max_{S \subseteq T}\{|N| : N \in \mathcal{F}\}$ and $T \subseteq S$ denote the size of a maximum cardinality independent set T, $m(T) \leq |T|$ and $\mathcal{F} = \{T \subseteq S : m(T) = |T|\}$.

Based on above definitions and notations the definition of matroid becomes appar-

ent, that is:

Definition 16: $M = (S, \mathcal{F})$ is a *matroid* if M is an independence system in which for any subset $T \subseteq S$ every independence set in T that is maximal in T (a base) has cardinality $m(T)$.

In this subsection we are going to present the problem of counting vertices of polyhedra whose sets of constraints grow exponentially. We will explore two classes of problems: (I) *Matroids*, (we consider two sub-classes (Ia) *Regular Matroids* and (Ib) *Balanced Matroids*) and (II) *Matchings* in *non-bipartite* graphs.

One of the examples of a matroid is the graphic matroids, whose bases are spanning trees of a graph; another is vectorial matroids whose S is a set of vectors over a field and the bases are the maximum cardinality linearly independent subsets of the vectors.

Definition 18: *Regular matroids* are the subclass of matroids vectorial over every field.

Edmonds [34] has introduced the bases-exchange graph of a matroid $\mathcal{M}$ denoted by $G(\mathcal{M})$ as the graph whose vertex-set is the collection of bases B_1 and B_2 connected by an edge if and only if B_2 can be obtained from B_1 by fundamental operation of removing and adding one of the ground-set elements: $B_2 = B_1 - \{e\} \bigcup \{f\}$. Let us present some results:

6.6.1 Matroids

The problem of counting the vertices of *Matroid Polytopes* which are of the form:

$$P(\mathcal{M}) = \{x \in \mathbb{R}^n_+ : \sum_{j \in S} x_j \leq m(S) \ for \ S \subseteq N\} \tag{6.40}$$

where for any matroid $\mathcal{M} = (N, \mathcal{F})$ with rank function m. It follows that $T \in \mathcal{F}$

$iff\ |T| \leq m(T)\ iff\ \sum_{j\in S} x_j^T = |S \bigcap T| \leq m(S),\ \forall S \subseteq N$. Edmonds shows that $P(\mathcal{M})$ is integral. That is:

Theorem 9 (*Edmonds*): $P(\mathcal{M})$ is an integral polytope.

By above theorem we have the following results.

Proposition 6.9: There is a *1-1* correspondence between the vertices of $P(\mathcal{M})$ in (6.40) and bases β associated with its matroids.

We will be interested in two types of graphical matroids, these are regular and balanced matroids:

Ia. Regular Matroid:

We have seen that a matrix is totally unimodular if all its subdeterminants are -1, 0 or $+1$. Totally unimodular matrices are known to be matrix representations of regular matroids, [84]. Let A be a totally unimodular $m \times n$ matrix with columns a_1, a_2,..., a_n, and as assumed above, let β be the set of bases of the associated regular matroid. If A has full row rank, that is $\beta \neq \emptyset$, Dyer and Frieze [25] have defined a *simple* (*natural*) *random walk* on β sequence B_0, B_1, B_2, ..., B_r,..... $\in \beta$ as follows. At B_r, randomly choose columns $a \in B_r$, $a' \in A - B_r$. Let $B'_r = B_r \bigcup \{a'\} - \{a\}$. If $B'_r \in \beta$ then $B_{r+1} = B'_r$, otherwise $B_{r+1} = B_r$. The following result is due to Dyer and Frieze [25].

Theorem 10 (*Dyer* and *Frieze*): For any $B \in \beta$,

$$|Pr[B_r = B] - |\beta|^{-1}| \leq (1 - \frac{1}{8m^4n^4})^r$$

Theorem 10 is used to show how to estimate $|\beta|$ which is the set of bases of the regular matroids associated with the totally unimodular matrix A. Dyer and Frieze [25] have also shown that the cardinality of β is efficiently obtainable using Binet-Cauchy formula for the product of 2 rectangular matrices.

Proposition 6.10: If A is a totally unimodular matrix representation of regular matroid $\mathcal{M}$ and β is the set of its bases, then the number of vertices of the polyhedron $P(\mathcal{M})$ can be exactly counted in polynomial time.

Proof :
The proof follows from Theorem 10 and the number of the bases, $|\beta|$. The *1-1* correspondence between basic feasible solutions and vertices completes the proof. QED

Ib. Balanced Matroids:
Feder and Mihail [35] have defined balanced matroids in simple terms to be those matroids whose minors satisfy the property that, for any randomly chosen basis, the presence of an element can only make any other element's presence less likely. We introduce some definitions and terminologies before presenting an important result from Feder and Mihail [35] with which we are to present an approximation of counting vertices of polyhedra associated with balanced matroids.

Definition 19: Let B be a basis chosen uniformly at random from β and for $e \in S$, let e denote the event $e \in B$. The matroid $\mathcal{M}(S, \beta)$ is said to satisfy the negative correlation property if the inequality

$$Pr[ef] \leq Pr[e]Pr[f]$$

holds $\forall e, f \in S$. Note that $Pr[e|f] \leq Pr[e]$. Graphic matroids and regular matroids are known to be negatively correlated.

Definition 20: A minor of a matroid is obtained by repeatedly performing the operation of choosing an element $e \in S$ and then selecting either those bases that contain e or those that do not. For graphic matroids, this operation corresponds to contraction or deletion of selected edges from the graph.

Definition 21: A matroid $\mathcal{M}(S, \beta)$ is balanced if all its minors, including $\mathcal{M}$ itself, satisfy negative correlation.

What are the natural and modified random walks?

Theorem 11 (*Feder* and *Mihail*): For the natural random walk on any balanced matroid the total variation distance can be bounded by ϵ in time $t = O(n \log m + \log \epsilon^{-1})n^2 m$.

where n is the rank, m is the cardinality of the ground-set, and ϵ is a bound on total variation distance. The result applies to graphic and regular matroids specifically.

Theorem 11b (*Feder* and *Mihail*): For the modified random walk on any balanced matroid, the total variation distance can be bounded by ϵ in time $t = O(n \log m + \log \epsilon^{-1})n^3$.

Open Problem: Give an *fpras* for arbitrary matroids.

6.6.2 Matchings in non-bipartite graphs

We briefly mentioned that the number of perfect matchings in a planar graph and spanning trees can be counted exactly in polynomial time. However, it is not yet known whether there is a polynomial algorithm for counting perfect matchings in non-bipartite graphs. The challenge is, can we develop an *fpras* for the non-bipartite graphs? Matching problems involve choosing a subset of the edges subject to degree constraints on the nodes. Before then let us review some definitions.

Definition 22: If $G(V, E)$ is a graph with set of nodes V and set of edges E, the number of edges that meet a node i is called the degree of node i.

Definition 23: A *matching* $M \subseteq E$ is a subset of edges with the property that each node in the subgraph $G(M) = (V, M)$ is met by no more than one edge. Every graph has a matching namely $M = \emptyset$ (because $\emptyset$ is a subset of every set). The

simplest case is *1-matching*, and generalization of which is the *b-matchings* in which node i is met by no more than b_i, which is a positive integer.

Definition 24: In 1-matching each node is met by exactly one edge. In some cases each node i is met by at least b_i edges. These type of problems are called node covering by edges.

An integer programming formulation of the weighted *0-1,* b-matching optimization problem is:

$$Max\ Cx \tag{6.41}$$

$$Ax \leq b,\ x \in B^n \tag{6.42}$$

where A is the node-arc incidence matrix of the graph, $|E| = n$, and $x_e = 1$ means that e is in the matching. The important property of A for matching problems is that each of its columns contains exactly two $1's$; in another words, $\sum_i a_{ij} = 2$, $\forall j \in E$. We should note that if the graph is bipartite, then A is totally unimodular so that the extreme points of the polyhedron;

$$P^{(match)} = \{x \in \mathbb{R}^n_+ : Ax \leq b\} \tag{6.43}$$

are b-matchings. But, if G contains an odd cycle, the constraint set of the linear programming relaxation may contain fractional extreme points.

A non-empty polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ with $rank(A) = n$ is called integral iff all its vertices are integral.

Define a polytope associated with a graph $G = (V, E)$ and matchings $M \subseteq E$ as follows:

$$P(M) = \{x \in \mathbb{R}^n_+ : \sum_{e \in \delta(v)} x_e \leq 1\ for\ v \in V, \sum_{e \in E(U)} x_e \leq \lfloor \frac{|U|}{2} \rfloor, \forall\ odd\ sets\ U \subseteq V\} \tag{6.44}$$

where $U \subseteq V$ is an odd set if $|U| \geq 3$ and is odd, and $e \in E$. Here $\delta(v)$ is the set of edges meeting vertex v. Edmonds has shown that the polytope $P(M)$ is integral

with $0-1$ vertices.

Theorem 12 (*Edmonds*): The polytope $P(M)$ is the convex hull of incidence vectors of matchings.

Proposition 6.14: There exists an *fpras* for counting the vertices of the polyhedron $P(M)$ defined in (6.44).

Proof :
The result follows from the work of Jerrum and Sinclair.

Open Problem: If $\sum_{e\in\delta(v)} x_e = 1$ in (6.44), it is still unknown whether there exist *fpras* for counting vertices of $P(M)$ if G is not bipartite. (If G is bipartite it follows from Jerrum et al. [49]).

6.7 Conclusion

In this chapter we reviewed approximate counting procedures. We discussed different *fpras* methods for counting vertices for some special classes of polyhedra associated with IS; Down-Set, 0-1 Knapsack, $2 \times n$ transportation, Matroids and matchings in non-bipartite graphs. These are mainly based on existing results [21, 26, 35, 46, 47, 51, 53, 75, 97].

Chapter 7

Conclusion and Future Work

7.1 Introduction

In this chapter we give some concluding remarks about the work that has been reported in this thesis and suggest some ways in which it can be extended. The VE problem has been the object of substantial research effort and, as discussed in Chapter 1, a considerable number of algorithms for its solution have been published. Despite this the VE problem still cannot be regarded as an "easy" problem. There are several reasons for this, in particular the fact that there can be no general VE algorithm which is polynomial in the input size. Some BOP algorithms, e.g. Dyer [24], have been shown to have good theoretical performance for simple polyhedra and good empirical performance for both simple polyhedra and those with a low degree of degeneracy. Unfortunately, the many-to-one relationship between basic feasible solutions and vertices leads to significant difficulties for such algorithms on highly degenerate polyhedra. Thus there is still plenty of opportunity for research in vertex enumeration. Chapters 2, 4, 5 and 6 of this thesis describe some original contribution to this research.

7.2 Achievements of the Thesis

In Chapter 1 the aims of the research were listed as:

1. To develop VE algorithms for some special classes of polyhedron;
2. To investigate whether vertices of some special classes of polyhedra can be approximately counted.

Here we summarise the achievements of the thesis in trying to meet those aims:

- **Achievement 1:** Provan [79] describes a pivoting algorithm and proves that its running time is quadratic in the number of vertices, even when as is frequently the case for networks, the polyhedron is degenerate. In Chapter 2 we discuss the non-trivial task of turning Provan's high level description of his primal algorithm into an implementation which allows the performance of his algorithm to be evaluated empirically. The results tend to confirm the quadratic nature but are perhaps disappointing in that even low dimension networks are computationally challenging. This is in part due to the fact that each vertex is generated several times. There does not appear to be any way in which we can tell in advance that a cycle will lead to an already discovered vertex without performing flow adjustment around the cycle, hashing into the hash table and performing an expensive arc by arc check. It does not seem possible to avoid this by use of a device such as the 'gamma-set' as edges of the network may correspond to different edges in the pseudo-graph at different points in the execution of the algorithm. In addition memory requirements of the algorithm are substantial due to the number of cycles which arise.

- **Achievement 2:** Williams suggested the use of dual *F-M* for vertex enumeration. In [98] he shows via an example, how this might be done. The approach is attractive as it is not significantly affected by degeneracy. We provide a complete algorithmic description of the method. Our computational experience suggests that this method is unlikely to be competitive with other VE algorithms due to the growth in the number of intermediate variables. We analyse the corresponding intermediate constraints growth for *LI(2)* systems

and prove that, provided the associated graph is connected, the growth is linear in the number of constraints and exponential in the number of variables.

- **Achievement 3:** In Chapter 5 we prove two propositions on the structure of bases for *LI(2)* and dual *LI(2)* systems. We exploit the graph representation of the basis implied by the proposition for dual *LI(2)* to develop a BOP algorithm for enumerating their vertices. We later show how to deal with degeneracy issues in the algorithm.

- **Achievement 4:** In Chapter 6 we turn our attention to counting the vertices of a polyhedron, without listing them. As this is an $\#P$-complete problem [24], we have developed *fpras* for approximately counting the vertices of polyhedra associated with some special classes as described in Chapter 6.

7.3 Further Work

As we have observed Provan's algorithm [79] faces great challenge with even relatively low dimension degenerate polyhedra. There is no way in which to tell whether cycles will lead to new vertices or not. Future suggested work may involve investigating whether or not a pattern can be developed and incorporated into Provan's algorithm that can detect when a given cycle leads to new or old discovered vertex, without having to perform flow adjustment.

The BOP algorithm developed in Chapter 5 can be implemented and evaluated empirically. It would be appropriate to compare the computational performance of this algorithm with general VE codes to assess the extent of any gain made by exploiting the dual *LI(2)* structure. Due to time constraints we were unable to implement the algorithm, but some components of the implementation are:

1. Code that finds a basic feasible solution (initial vertex) to a dual *LI(2)* system, using LP if necessary.

2. Code that finds the components of the basis graph. The strongly connected components' code used in the Provan's code discussed in Chapter 2 may be

modified and adapted.

3. Binary encoding for storing the β sets and γ sets. Modification of and extraction from β, γ is then easy in any language which supports bit manipulation.

4. Code to implement the SOLVE routine. This is potentially the most important component as it is called several times per vertex, in order to perform the necessary simplex pivots. It may be sub-divided into four sub-routines:

   - For simple loops assign the associated variable (edge) its corresponding nodes values (trivial).
   - For a tree, identify the root and leaf nodes, and start with node that has degree 1 and move upward to the root. Code similar to that in Provan's algorithm for trees in pseudo-vertices can be employed.
   - For a graph with one cycle, assign edges value by starting at nodes with degree 1 and moving upward until the cycle is reached, i.e. there are no nodes of degree 1.
   - For a cycle, break the cycle by assigning an arbitrary value to one of the edge and solve for the others parametrically.

5. Binary encoding can also be used for the μ sets. Their construction and comparison again requires bit manipulation routines. As γ sets are only stored for the 'master' copy of a degenerate vertex, they could be stored in a separate structure which is pointed to by the vertex list.

6. Hash tables code can be used for accounting procedure, via a simple hashing of the binary encoded β set.

As we noted in Chapter 5 the similar basis structures of *LI(2)* and dual *LI(2)* suggest that a similar BOP algorithm can be developed for *LI(2)*. This requires further investigation.

In Chapter 6 some open questions are raised as to whether there are *fpras* for counting Down-Sets with partial order and Independent Set in a bipartite graph? and

whether there exist *fpras* for arbitrary matroids. If $\sum_{e \in \delta(v)} x_e = 1$ in (6.44) of Chapter 6, it is still unknown whether there exist *fpras* for counting vertices of $P(M)$ as described in (6.44). These questions need further investigation.

In the process of completing this thesis, Dyer [31] has developed a new, more efficient *fpras* for counting the *0-1*-Knapsack solutions based on dynamic programming. The algorithm uses dynamic programming and "provide a *deterministic* relative approximation". He also used what are called "dart throwing" techniques for the provision of arbitrary approximations. It is recommended that further research for counting should explore using dynamic programming on the other vertex counting problems discussed in Chapter 6.

Bibliography

[1] A V Aho, J E Hopcroft, and J D Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass, 1974.

[2] W Altherr. An algorithm for enumerating all vertices of a convex polyhedron. *Computing (Austria)*, 15:181–193, 1975.

[3] B Aspvall and Y Shiloach. Polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 9:827–845, 1980.

[4] D Avis and K Fukuda. A pivoting algorithm for convex hulls and vertex enumeration for arrangements and polyhedra. *Discrete Computational Geometry*, 8:295–313, 1992.

[5] M L Balinski. An algorithm for finding all vertices of convex polyhedral sets. *Journal of the Society of Industrial and Applied Mathematics*, 9:72–88, 1961.

[6] D W Barnette. The minimum number of vertices of a simple polytope. *Israel Journal of Mathematics*, 10:121–125, 1971.

[7] R G Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2:103–107, 1977.

[8] C Burdet. Generating all the faces of a polyhedron. *SIAM Journal on Applied Mathematics*, 26:479–489, 1974.

[9] M J Carrillo. An algorithm for finding the vertices of a convex polyhedron. Working paper, University of Texas at Dallas, September 1977.

[10] P-C Chen, P Hansen and B Jaumard. On-line and off-line vertex enumeration by adjacency lists. *Operations Research Letters*, 10:403–409, 1991.

[11] P-C Chen, P Hansen and B Jaumard. Partial pivoting in vertex enumeration. Rutcor research report 10-92, Rutgers University, March 1992.

[12] N V Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities. *USSR Computational Mathematics and Mathematical Physics*, 5:228–233, 1965.

[13] E Cohen and N Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23:1313–1347, 1994.

[14] T H Cormen, C E Leiserson, and R L Rivest. *Introduction to Algorithms.* MIT Press / McGraw-Hill, 1990.

[15] M Cryan and M Dyer. A polynomial-time algorithm to approximately count contingency tables when the number of rows is constant. In 34^{th} *Annual ACM Symposium on the Theory of Computing (STOC 02)*, pages 240–249, 2002.

[16] M Cryan, M Dyer, H Muller and L Stougie. Random walks on the vertices of transportation polytopes with constant number of sources. *Submitted July 2002*, 2002.

[17] M Cryan, M Dyer, L Goldberg, M Jerrum and R Martin. Rapidly mixing Markov Chains for sampling contingency tables with a constant number of rows. In *Proceedings of the* 43^{rd} *Annual IEEE Symposium on Foundations of Computer Science (FOCS 02)*, to appear.

[18] G Dahl and S Storoy. Decomposed enumeration of extreme points in the linear programming problem. *BIT*, 15:151–157, 1975.

[19] G B Dantzig. *Linear Programming and Extensions.* Princeton University Press, N.J., 1963.

[20] M Dyer, L A Goldberg, and M Jerrum. Counting and sampling H-colourings. In Rolim, J D P and Vadharn, S, editor, *Randomization and Approxima-*

tion Techniques: 6th International Workshop, RANDOM 2002, pages 51–67. Springer-Verlag, 2002.

[21] M Dyer and C Greenhill. On Markov chains for independent sets. *Journal of Algorithms*, 35:17–49, 2000.

[22] M Dyer, R Kannan, and J Mount. Sampling contingency tables. *Random Structures and Algorithms*, 10:487–506, 1997.

[23] M E Dyer. *Vertex Enumeration in Mathematical Programming: Methods and Applications*. PhD thesis, The University of Leeds, October 1979.

[24] M E Dyer. The complexity of vertex enumeration methods. *Mathematics of Operations Research*, 8:381–402, 1983.

[25] M E Dyer and A Frieze. Random walks, totally unimodular matrices, and a randomised dual simplex algorithm. *Mathematical Programming*, 64:1–16, 1994.

[26] M E Dyer and C Greenhill. Polynomial-time counting and sampling of two-rowed contingency tables. *Theoretical Computer Science*, 246:265–278, 2000.

[27] M E Dyer and L G Proll. Vertex enumeration in convex polyhedra: a comparative computational study. In T B Boffey, editor, *Proceedings of the CP77 Combinatorial Programming Conference*, pages 23–43, University of Liverpool, Liverpool, 1977.

[28] M E Dyer and L G Proll. Eliminating extraneous edges in Greenberg's algorithm. *Mathematical Programming*, 19:106–110, 1980.

[29] M E Dyer and L G Proll. An improved vertex enumeration algorithm. *European Journal of Operational Research*, 9:359–368, 1982.

[30] M E Dyer and LG Proll. An algorithm for determining all extreme points of a convex polytope. *Mathematical Programming*, 12:81–96, 1977.

[31] Martin Dyer. Approximate counting by dynamic programming. School of computing research report 2002.16, University of Leeds, November 2002.

[32] M E Dyer, L A Goldberg, C S Greenhill and M R Jerrum. On the Relative Complexity of Approximate Counting Problems. In Jansen, K and Khuller, S, editor, *Approximation Algorithms for Combinatorial Optimization: 3^{rd} International Workshop, APPROX 2000*, pages 108–119. Springer-Verlag, 2002.

[33] M Dyer, R Kannan, A Kapoor, L Perkovic and U Vazirani. A mildly exponential time algorithm for approximating the number of solutions to a multidimensional Knapsack problem. *Combinatorics, Probability and Computing*, 2:271–284, 1993.

[34] J Edmonds. Submodular functions, matroids and certain polyhedra. In *Combinatorial Structures and their Applications, Proceedings, Calgary International Conference*, 1969.

[35] T Feder and M Mihail. Balanced matroids. In *24th ACM Symposium on the Theory of Computing*, pages 26–38, 1992.

[36] A M Galperin. The general solution of a finite system of linear inequalities. *Mathematics of Operations Research*, 1:185–196, 1976.

[37] M R Garey and D S Johnson. *Computers and Intractability.* Freeman, 1979.

[38] R Graham. An efficient algorithm for determining the convex hull of a planar set. *Information Processing Letters*, 1:132–133, 1972.

[39] H Greenberg. An algorithm for determining redundant inequalities and all solutions to convex polyhedra. *Numerische Mathematik*, 24:19–26, 1975.

[40] B Grunbaum. *Convex Polytopes.* John Wiley and Sons, 1967.

[41] G Hadley. *Linear Programming.* Addison-Wesley, 1962.

[42] D S Hochbaum and J Naor. Simple and fast algorithms for linear and integer program with two variables per inequality. *SIAM Journal of Computing*, 23:1179–1192, 1994.

[43] R Horst and H Tuy. *Global Optimization: Deterministic Approaches.* Springer-Verlag, 2nd edition, 1992.

[44] G R Jahanshahlou and G Mitra. Two algorithms for finding all the vertices of a convex polyhedron. *Colloquia Mathematica Socistatis Janos Bolyai*, 12:503–535, 1974.

[45] M Jerrum. The computational complexity of counting. In M Jerrum, editor, *Counting*, International Congress of Mathematicians, pages 1407–1416, Birkhauser, Basel 1995, 1995.

[46] M Jerrum and A Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18:1149–1178, 1989.

[47] M Jerrum and A Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22:1087–1116, 1993.

[48] M Jerrum and A Sinclair. The Markov Chain Monte Carlo method: an approach to approximate counting and integration. In D Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 482–520. PWS publishing, Boston, 1996.

[49] M Jerrum, A Sinclair, and E Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. Report tr00-079, Electronic Colloquium on Computational Complexity (ECCC), 2000.

[50] M R Jerrum. Mathematical foundations of the Markov Chain Monte Carlo method. In J Ramirez-Alfonsin M Habib, C McDiarmid and B Reeds, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16, pages 116–165. Springer-Verlag, 1998.

[51] M R Jerrum and Z Liptak. Two good counting algorithms. In *Counting, Sampling and Integrating: Algorithms and Complexity*, pages 1–12. http://www.dcs.ed.ac.uk/home/mrj/pubs.html, 2001.

[52] M R Jerrum, L G Valiant, and V V Vazirani. Random generation of combinatorial structures from uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[53] M R Jerrum and U Wagner. Sampling and counting. In *Counting, Sampling and Integrating: Algorithms and Complexity*, pages 25–34. http://www.dcs.ed.ac.uk/home/mrj/pubs.html, 2001.

[54] L G Kachiyan. A polynomial time algorithm in linear programming. *Doklady Akademiia Nauk SSSR Noaia Seriia*, 20:191–194, 1979.

[55] J L Kennington and R V Helgason. *Algorithms for Network Programming.* John Wiley and Sons, Inc., 1980.

[56] D B Khang and O Fujiwara. A new algorithm to find all vertices of a polytope. *Operations Research Letters*, 8:261–264, 1989.

[57] M J L Kirby, H R Love, and K Swarup. A cutting plane algorithm for extreme point mathematical programming. *Cahiers du Centre d'Etudes de Recherche Operationnelle*, 14:27–42, 1971.

[58] D E Knuth. *The Art of Computer Programming Vol. 3: Sorting and Searching.* Addison-Wesley, 1973.

[59] V G Kuznetsov. Algorithms for finding the general solution of a system of linear inequalities. *USSR Computational Mathematics and Mathematical Physics*, 6:197–205, 1966.

[60] L L and I Barany. Borsuk's theorem and the number of facets of centrally symmetric polytopes. *Acta Mathematica Academy Sci. Hung*, 40, 1982.

[61] I Leader. Discrete isoperimetric inequalities. In *Proceedings of Symposia in Applied Mathematics*, volume 44, pages 57–80, San Francisco, California, 1991.

[62] M Luby and E Vigoda. Approximately up to four. In *29th Annual Symposium on the Theory of Computing, ACM, El Paso, Texas*, pages 682–687, 1997.

[63] J J Jarvis M S Bazaraa and H D Sherali. *Linear Programming and Network Flows.* John Wiley and Sons, 1977.

[64] M Manas and J Nedoma. Finding all vertices of a convex polyhedron. *Numerische Mathematik*, 12:226–229, 1968.

[65] T H Mattheiss. An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. *Operations Research*, 21:247–260, 1973.

[66] T H Mattheiss. Computational results on an algorithm for finding all vertices of a polytope. Working paper no. 1, College of Commerce and Business Administration, University of Alabama, October 1978.

[67] T H Mattheiss and D S Rubin. A survey and comparison of methods for finding all vertices of convex polyhedral sets. *Mathematics of Operations Research*, 5:167–185, 1980.

[68] P McMullen and G C Shepard. Convex polytopes and the upper bound conjecture. *London Mathematical Society Lecture Notes Series 3 Cambridge University Press, London*, 1971.

[69] B Morris and A Sinclair. Random walks on truncated cubes and sampling *0-1* Knapsack solution. submitted.

[70] B Morris and A Sinclair. Random walks on truncated cubes and sampling *0-1* Knapsack solution (preliminary version). In *Proceedings of the 41th IEEE conference on Foundations of Computer Science*, pages 230–241, 1999.

[71] T S Motzkin, H Raiffa, G L Thompson and R M Thrall. The double description method. In *Contributions to the Theory of Games*, volume 2, pages 51–73. Princeton University Press, Princeton, 1953.

[72] K G Murty. An algorithms for ranking all assignments in increasing order of cost. *Operations Research*, pages 682–687, 1969.

[73] C G Nelson. An $n^{O(logn)}$ algorithm for the two variables-per-constraint linear programming satisfiability problem. Report no. stan-cs-76-689, Stanford University, 1978.

[74] C G Nelson and D C Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Systems and Languages*, 1:245–257, 1979.

[75] G L Nemhauser and L A Wolsey. *Integer and Combinatorial Optimzation.* John Wiley and Sons, Inc., 1999.

[76] V A Nicula and L V Nicula. An algorithm for the determination of the general non-negative solution of a system of linear equations. In *Proceedings of the 4th Conference on Probability*, pages 557–570, Brasvo, Rumania, 1971.

[77] S B Ong, M E Dyer, and L G Proll. A comparative study of three vertex enumeration algorithms. Technical report, School of Computer Studies, University of Leeds, 1996.

[78] M A Pollatschek and B Avi-Itzhak. Sorting feasible basic solutions of a linear program. *Paper presented at the 3rd Annual Israel Conference on Operations Research*, 1969.

[79] J Scott Provan. Efficient enumeration of the vertices of polyhedra associated with network LP's. *Mathematical Programming*, 64:47–64, 1994.

[80] R C Read and R E Tarjan. Bounds on backtrack algorithms for listing cycles, paths and spanning trees. *Networks*, 5:237–252, 1975.

[81] Cooper Redwine. *Upgrading to Fortran 90.* Springer-Verlag, 1995.

[82] E Ya Remez and A S Shteinberg. A theorem of convex polyhedra in connection with the problem of finding the set of solutions to a system of linear inequalities. *Ukrainskii Mathematicheskii Zhurnal*, 19:74–89, 1967.

[83] D S Rubin. Vertex generation and cardinality constrained linear programs. *Operations Research*, 23:555–564, 1975.

[84] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1986.

[85] R Seidel. A convex hull algorithm optimal for point sets in even dimension. Technical report, University of British Columbia, 1981.

[86] R Seidel. Constructing higher dimensional convex hull at logarithmic cost per face. In *18th ACM Symposium on the Theory of Computing*, pages 404–413, 1986.

[87] Raimund Seidel. The upper bound theorem for polytopes: an easy proof of its asymptotic version. *E-mail Communications to Ziegler*, 1994.

[88] Raimund Seidel. Linear programming and convex hulls made easy. In *6th Annual Symposium on Computational Geometry Barkeley,California*, pages 211–215, June 6-8,1990.

[89] A Shefi. *Reduction of linear inequality constraints and determination of all feasible extreme points.* PhD thesis, Stanford University, 1969.

[90] R Shostak. Deciding linear inequalities by computing loop residues. *Journal of the Association for Computing Machinery*, 28:769–779, 1981.

[91] G J Silverman. Computational considerations in extreme point enumeration. Technical Report G320-2649, IBM Los Angeles Scientific Centre, 1971.

[92] A Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach.* Birkhauser Boston Inc., 1993.

[93] G Swart. Finding the convex hulls facet by facet. *Algorithms*, 6:17–48, 1985.

[94] K Tone. An algorithm for finding all extremal rays of polyhedral convex cones with some complementarity conditions. *Journal of the Operations Research Society of Japan*, 18:97–109, 1975.

[95] H Uzawa. A theorem on convex polyhedral cones. In L Hurwicz K J Arrow and H Uzawa, editors, *Studies in Linear and Non-Linear Programming*, pages 23–31. Stanford University Press, Stanford, CA, 1958.

[96] L G Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[97] L G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

[98] H P Williams. Fourier's method of linear programming and its dual. *American Mathematical Monthly*, 93:681–694, 1986.

[99] H P Williams. *Model Solving in Mathematical Programming*. John Wiley and Sons, 1993.

[100] H P Williams. personal communication, 2000.

[101] G M Ziegler. *Lectures on Polytopes*. Springer-Verlag, 1994.