A Study of Combinatorial Optimization

Anil Kumar Jhingonia
MS09015

A dissertation submitted for the partial fulfilment of
BS-MS dual degree in Science

Indian Institute of Science Education and Research Mohali
April 2014

Certificate of Examination

This is to certify that the dissertation titled "**A Study of Combinatorial Optimization** " submitted by **Mr. Anil Kumar Jhingonia (Reg. No. MS09015)** for the partial fulfillment of BS-MS dual degree programme of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.

Yashonidhi Pandey (Supervisor) Kapil H. Paranjape K. Gongopadhyay

Dated: April 25, 2014

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Yashonidhi Pandey at the Indian Institute of Science Education and Research Mohali.
This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgement of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the bibliography.

Anil Kumar Jhingonia
(Candidate)
Dated: April 25, 2014

In my capacity as the supervisor of the candidate's project work, I certify that the above statements by the candidate are true to the best of my knowledge.

Yashonidhi Pandey
(Supervisor)

Acknowledgement

I would like to express my gratitude to Yashonidhi Pandey, my project supervisor for his kind supervision and guidance in completing my dissertation. I would also like to thank IISER Mohali for their support.

I would like to thank David P. Williamson and R. Ravi for their valuable online notes on primal dual method, Jim Anderson whose lectures ppt on max flow problem helped me a lot in understanding the concepts. I would also like to thank Michel X. Goemans whose online lectures on combinatorial optimization helped me in understanding this subject.

Finally, I am grateful to my friends and family for giving me all the non-technical support and encouragement because of which I got the opportunity to learn mathematics.

Anil Kumar Jhingonia
MS09015
IISER Mohali

Contents

Abstract **2**

Introduction **3**

1 Computational complexity of Matrix Multiplication and Inversion **5**
1.1 Introduction 5
1.2 Asymptotic notation 7
1.3 Matrix multiplication is no harder than inversion 7
1.4 Symmetric and Positive definite matrix 8
1.4.1 Properties of Symmetric and Positive definite matrix 8
1.4.2 Schur complement 9
1.5 Inversion is no harder than multiplication 9
1.6 **Solving systems of linear equations** 12
1.6.1 Computing an LU decomposition 13
1.6.2 Computing an LUP-decomposition 15

2 Ellipsoid Algorithm **16**
2.1 Definitions 16
2.2 The Basic Ellipsoid Algorithm 18
2.3 Basic Ellipsoid Iterations 19
2.4 Separation Oracle 20
2.5 Correctness of the algorithm 20
2.6 Time complexity 24

3 Primal Dual Algorithm **25**
3.1 Introduction: 25
3.2 Duality 26
3.2.1 Relationships between the primal and the dual problems 26

3.3 Economic interpretation of dual problem 27
3.3.1 Production problem 27
3.3.2 Pricing problem 28
3.3.3 For 1×2 matrices(one resource and two products) 29
3.3.4 For 2×1 matrices (2 resource and 1 product) 31
3.4 Dual problem derived algebraically 32
3.5 Weak Duality theorem 33
3.6 Complementary Slackness 34
3.6.1 Complementary Slackness Theorem 35
3.7 Strong Duality theorem 35

4 Strong Duality and Max-Flow Min-Cut Problem **36**
4.1 The Primal-Dual Method Applied to Max-Flow Min-Cut Problem . . . 36
4.1.1 Max-Flow Problem 37
4.1.2 Min-Cut Problem 39
4.2 Strong Duality 46
4.3 Max-flow Min-cut Theorem 47

Bibliography **48**

Abstract

The primal-dual method is a standard tool in the design of algorithms for combinatorial optimization problems. It is a very powerful method. This method can be used to obtain a good approximation algorithm from which we can get a good combinatorial algorithm. It can also be used to prove good performance for combinatorial algorithms. Max-flow Min-cut is a very nice example of primal dual method. we would like to interpret its primal, then obtain its dual, interpret the dual and then prove the max-flow min-cut theorem using the strong duality.

Introduction

In this thesis, I would like to focus on some topics in combinatorial optimization. I have studied about the primal dual algorithm, strong duality, weak duality, max-flow min-cut problem, ellipsoid algorithm etc.. I have tried my best to describe these topics in a good manner.

In the first chapter we will talk about the complexity of matrix multiplication and inversion. The time complexity of matrix multiplication and inversion is equal. Usually, by the ordinary method we require $2n^3 - n^2 = O(n^3)$ scalar operations to compute the matrix product C = AB (n^3 multiplications and $n^3 - n^2$ additions). Then V. Strassen showed in his paper [1] that we can compute the matrix multiplication of two $n \times n$ in $O(n^{log_2 7}) \approx n^{2.807}$ time which is less than $O(n^3)$. Currently the best one is due to Coppersmith and Winograd and it works in $O(n^{2.376})$.[2]

In the second chapter ellipsoid algorithm is described. It is the first polynomial time algorithm on the feasibility problem of a system of linear inequalities. Leonid Khachyan showed that LP is solvable in polynomial time by a method of shrinking ellipsoids. This method is a modification of an algorithm introduced by Shor for non-linear optimization problems. Ellipsoid method reduces one combinatorial optimization problem to the other which is called the separation problem. Since this is the first polynomial time algorithm to solve the linear programming problems in worst case and its theoretical base is very strong, makes this method very important in combinatorial optimization.

Third chapter is on the primal dual algorithms. Primal dual method is a very important concept in combinatorial optimization. This method is very important as a means of devising algorithms for problems in combinatorial optimization. Some problems are easy to solve in their dual form in comparison to their primal (original) form. Weak duality gives an upper bound on the primal solution. And by strong

duality we can say that if there is an optimal solution for primal then so for the dual and vice versa, and these optimal solutions for the primal and dual would be equal and unique. Strong duality is very main concept in combinatorial optimization by which we can solve many optimization problems.

Chapter 4 is an application of strong duality. Max-flow min-cut problem is a very good example of primal dual method which can be solved by applying the strong duality on it. Max-flow min-cut theorem is a special case of strong duality.

Chapter 1

Computational complexity of Matrix Multiplication and Inversion

In this chapter we discuss about the computational complexity of matrix multiplication and inversion. Matrix multiplication and inversion are considered as equally hard problems. They are equally hard in the sense that if we can multiply two $n \times n$ matrices in $Mult(n)$ time, then we can invert a non-singular $n \times n$ matrix in $O(Mult(n))$ time and similarly if we can invert a non-singular $n \times n$ matrix in $I(n)$ time then we can multiply two $n \times n$ matrices in time $O(I(n))$. At the end we briefly discuss LUP decomposition of a matrix.

1.1 Introduction

To understand this we talk about the procedure to compute the running time of matrix multiplication, addition, subtraction, division, inversion etc. Running time of an algorithm on a particular input is proportional the number of primitive operations executed. For simplicity we shall assume that the time taken to execute a particular operation is a fixed constant.
So, let A be a $n \times m$ matrix, B be a $m \times n$ matrix and C be their product. To get an element of product matrix(C) we have to multiply a row of matrix A to a column of matrix B that means that for any i,j, to compute the entry C_{ij} of C one needs m multiplications and some additions. Since there are n^2 entries in C, so the time taken to compute matrix C will be $O(mn^2)$. Hence to obtain the matrix C ,the computa-

tional cost is $O(mn^2)$.
So, for special case where m = n ,the computational cost of multiplication of two matrix is $O(n^3)$.
Similarly for addition of two $n \times n$ matrices, we will have to perform n^2 addition operations. So addition of two matrices takes time $O(n^2)$.

These are the estimated worst case running times of standard algorithms for matrix multiplications and additions. We will wonder if we get an efficient algorithm which can compute the matrix multiplication in less than $O(n^3)$ time. In the late 1960s, Strassen discovered an algorithm for multiplying 2×2 matrices using only 7 essential multiplications instead of 8. He then used this algorithm recursively to derive an algorithm for multiplying $n \times n$ matrices with $O(n^{log_2 7}) = O(n^{2.808})$.

Strassen's matrix multiplication algorithm :[3] Let A and B be $n \times n$ matrices and C be their product, where n is a power of 2. Then we write their product as $-$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22}. \end{bmatrix} \tag{1.1}$$

To compute this product, first compute the following products -
$m_1 = (a_{12} - a_{22})(b_{21} + b_{22})$,
$m_2 = (a_{11} + a_{22})(b_{11} + b_{22})$,
$m_3 = (a_{11} - a_{21})(b_{11} + b_{12})$,
$m_4 = (a_{11} + a_{12})b_{22}$,
$m_5 = a_{11}(b_{12} - b_{22})$,
$m_6 = a_{22})(b_{21} - b_{11})$,
$m_7 = (a_{21} + a_{22})b_{11}$.

Then compute the c_{ij}'s, using the formulas
$c_{11} = m_1 + m_2 - m_4 + m_6$,
$c_{12} = m_4 + m_5$,
$c_{ij} = m_6 + m_7$,
$c_{ij} = m_2 - m_3 + m_5 - m_7$.
Hence, Strassen had given this algorithm using only 7 multiplication and 18 additions/subtractions.

1.2 Asymptotic notation

We shall use three notations to describe the asymptotic running time. These are [4]:-

1. Θ notation :- This gives us an upper bound and a lower bound for the running time, which shows the worst case and best case running time. For a given function g(n), we denote by $\Theta(g(n))$ the set of functions,
   $\Theta(g(n)) = \{$f(n) : there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.

2. O notation :- This gives us an upper bound for running time which implies that the algorithm would not take more than this time. We denote by $O(g(n))$ the set of functions,
   $O(g(n)) = \{$f(n) : there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

3. Ω notation :- This gives us lower bound for the running time of an algorithm. We denote by $\Omega(g(n))$ the set of functions,
   $\Omega(g(n)) = \{$f(n) : there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.

1.3 Matrix multiplication is no harder than inversion

Theorem 1.3.1. *: If we can invert an $n \times n$ matrix in time $I(n)$, where $I(n) = \Omega(n^2)$ and $I(n)$ satisfies the regularity condition $I(3n) = O(I(n))$, then we can multiply two $n \times n$ matrices in time $O(I(n))$.*

Proof. : Here we want to compute the matrix C which is the product of the two $n \times n$ matrices A and B. Let D be a $3n \times 3n$ matrix defined by,

$$D = \begin{bmatrix} I(n) & A & 0 \\ 0 & I(n) & B \\ 0 & 0 & I(n) \end{bmatrix}$$

then we can find the inverse of D which is given by,

$$D^{-1} = \begin{bmatrix} I(n) & -A & AB \\ 0 & I(n) & -B \\ 0 & 0 & I(n) \end{bmatrix}$$

and hence, we can compute the product AB by taking the upper right $n \times n$ sub-matrix of D^{-1}. Here we take the matrix D as triangular because it is easy to handle the triangular matrices while doing a matrix multiplication or finding the inverse of a matrix. We can construct matrix D in $\Theta(n^2)$ time (we have to put $3n \times 3n = 9n^2$ elements in the matrix and for that we have to perform n^2 operations), which is $O(I(n))$ because we assume that $I(n) = \Omega(n^2)$, and we can invert D in $O(I(3n)) = O(I(n))$ time, by the regularity condition on $I(n)$. We thus have $Mult(n) = O(I(n))$[4]. □

In the Proof of the next theorem we use symmetric positive definite matrices, so first we shall define the symmetric positive definite matrices and their properties.

1.4 Symmetric and Positive definite matrix

Matrix 'A' is called symmetric if A = A^T.
Matrix 'A' is called real positive definite if and only if for all x $\neq$ 0, $x^T Ax > 0$.
These matrices are invertible , and LU decomposition can be performed on them without worrying about dividing by zero.

1.4.1 Properties of Symmetric and Positive definite matrix

1. Any positive definite matrix is nonsingular.

2. If A is symmetric positive definite matrix, then every leading sub-matrix of A is symmetric and positive definite.

We will use Schur complement lemma in this proof. So, we are stating the lemma here (without giving the proof).

1.4.2 Schur complement

Let A be a matrix, given by

$$A = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

then $S = H - GE^{-1}\mathrm{F}$ is called the **Schur complement** of A with respect to E.

Lemma 1.4.1. *If A is a symmetric positive definite matrix and A_k is a leading $k \times k$ sub-matrix of A, then the Schur complement S of A with respect to A_k is symmetric and positive definite.*

1.5 Inversion is no harder than multiplication

Theorem 1.5.1. *: If we can multiply two $n \times n$ matrices in time Mult(n), and Mult(n) $= \Omega(n^2)$ and Mult(n) satisfies two regularity conditions, which are as follows : −*

1. *$Mult(n + k) = O(Mult(n))$ for $0 \leq k \leq n$*
2. *$Mult(n/2) \leq cMult(n)$ for some constant $c < 1/2$.*

Then we can compute the inverse of any real non-singular $n \times n$ matrix in time $O(Mult(n))$.

Proof. : − we are going to prove this theorem only for real matrices. If n is not a power of 2, then since there exists a power of 2, say 2^k, where k is a natural number, between n and 2n, we may embed the given matrix into one of size 2^k. By the first regularity condition we can say that this embedding only augments the running time by only a constant factor. Hence, without loss of generality, we may suppose that n is a power of 2. Thus we have to prove our claim only for the case, when n is a power of 2. Now to prove this, we have to consider two cases-

1. When the $n \times n$ matrix A is symmetric and positive definite, or
2. when the $n \times n$ matrix A is invertible but not symmetric and positive definite.

So,

Case-1 :When the $n \times n$ matrix A is symmetric and positive definite:

Partition A and A^{-1} into four $n/2 \times n/2$ sub-matrices,

$$A = \begin{bmatrix} B & C^T \\ C & D \end{bmatrix}$$

and

$$A^{-1} = \begin{bmatrix} R & T \\ U & V \end{bmatrix}$$

$S = D - CB^{-1}C^T$ is the **Schur complement of A with respect to B** we can write A^{-1} as-

$$A^{-1} = \begin{bmatrix} B^{-1} + B^{-1}C^T S^{-1} CB^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{bmatrix} \tag{1.2}$$

and because A is symmetric and positive definite, B and S are also symmetric and positive definite matrices and so by the definition of a symmetric and positive definite matrix B and S are invertible ,so B^{-1} and S^{-1} exists and are symmetric as well. So,
$(B^{-1})^T = B^{-1}$ and $(S^{-1})^T = S^{-1}$
Now , we can compute the elements of A^{-1} as follows-

1. Compute B^{-1} from B.
2. Compute the product W = CB^{-1} and then compute its transpose $W^T = B^{-1}C^T$.
3. Compute the product X = WC^T = C$B^{-1}C^T$ and then compute the Schur complement of A, S = D - X = D - C$B^{-1}C^T$.
4. Now compute S^{-1} and set V to S^{-1}.
5. Compute the product Y = S^{-1}W and then compute its transpose $Y^T = W^T S^{-1}$, set T to -Y^T and U to -Y.
6. Compute the product Z = $W^T Y$ and set R to $B^{-1} + Z$.

Hence, the total time to compute the inverse of an $n \times n$ matrix A is −

$$I(n) \leq 2I(n/2) + 4Mult(n/2) + O(n^2)$$

where, 2I(n/2) is the cost of inverting two $n/2 \times n/2$ sub-matrices, 4Mult(n/2) is the cost of multiplying four $n/2 \times n/2$ sub-matrices and $O(n^2)$ is the cost of performing a constant number of additions, subtractions and transpose on $n/2 \times n/2$ sub-matrices.

By the second regularity condition of Mult(n),

$$I(n) = 2I(n/2) + \Theta(Mult(n)) = O(Mult(n))$$

Hence we are done with case-1.

Case-2 :When the $n \times n$ matrix A is invertible but not symmetric and positive definite.
Because A is non-singular, the matrix $A^T A$ is symmetric and positive definite (by the properties of non-singular matrix). So, instead of inverting A, we will invert the matrix $A^T A$. Again we are assuming that n is a exact power of 2. We can write A^{-1} as-

$$A^{-1} = (A^T A)^{-1} A^T \tag{1.3}$$

since,

$$((A^T A)^{-1} A^T)A = (A^T A)^{-1}(A^T A) = I_n \tag{1.4}$$

and we know that every matrix has a unique inverse, therefore we compute the A^{-1} using following steps-

1. Multiply A^T by A to get $A^T A$.
2. Compute the inverse of $A^T A$ using the above divide and conquer algorithm.
3. Multiply the result by A^T.

We can see that each of these three steps takes $O(Mult(n))$ time, and hence we can invert any non-singular real matrix in $O(Mult(n))$ time.

Thus we are done with case-2 also.

□

Proof of equation(3) [3]:-

$$A^{-1} = \begin{bmatrix} B^{-1} + B^{-1}C^T S^{-1} C B^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1} C B^{-1} & S^{-1} \end{bmatrix}$$

We can write A as -

$$A = \begin{bmatrix} B & C^T \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CB^{-1} & I \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & B^{-1}C^T \\ 0 & I \end{bmatrix} \tag{1.5}$$

where S is the Schur complement of A with respect to B.
Hence,

$$A^{-1} = \begin{bmatrix} I & -B^{-1}C^T \\ 0 & I \end{bmatrix} \begin{bmatrix} B^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CB^{-1} & I \end{bmatrix}, \tag{1.6}$$

$$A^{-1} = \begin{bmatrix} B^{-1} + B^{-1}C^T S^{-1} CB^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{bmatrix}. \qquad \square$$

1.6 Solving systems of linear equations

In this we discuss the LUP-decomposition method to solve the systems of the linear equations. We call the three matrices L,U and P, the LUP-decomposition of a matrix A, such that

$$PA = LU \tag{1.7}$$

where, L represents a unit lower triangular matrix, U represents an upper triangular matrix and P represents a permutation matrix.

We use this method because one can solve the linear systems easily when they are triangular. Any $n \times m$ matrix A, over any field can be written as a product, LUP, of three matrices, where L, U and P are as mentioned above.
Now we talk about how we use these triangular matrices to solve the systems of linear equations. Once we have the LUP decomposition of a matrix A, we can solve the equation $Ax = b$, where A is an $n \times n$ matrix and x, b are n-vectors, by solving only triangular linear systems, as follows [4]: −

1. We multiply both sides of equation $Ax = b$ by permutation matrix P, which yields

$$PAx = Pb \tag{1.8}$$

2. By equation (1.7),

$$LUx = Pb \tag{1.9}$$

This equation can be solved by solving two triangular linear systems.

3. Define Ux = y, where x is the desired solution vector. Then equation (1.9) becomes,

$$Ly = Pb \tag{1.10}$$

We solve this lower triangular system firstly for the unknown vector y by the method called "forward substitution".

4. We then solve the upper triangular system

$$Ux = y \tag{1.11}$$

for the unknown vector x by the method called "back substitution".

Since the permutation matrix P is invertible, multiplying both sides of equation (1.7) by P^{-1} gives $P^{-1}PA = P^{-1}LU$, so that

$$A = P^{-1}LU \tag{1.12}$$

Hence, the vector x is the solution to Ax = b.

$Ax = P^{-1}LUx$ (by equation (1.12))
$= P^{-1}Ly$ (by equation (1.11))
$= P^{-1}Pb$ (by equation (1.10))
= b

1.6.1 Computing an LU decomposition

This is the special case of LUP decomposition where P is absent or $P = I_n$ (identity matrix). So, by eq. (1.7)

$$A = LU \tag{1.13}$$

Therefore, the two matrices L and U are called the **LU decomposition** of the matrix A. Rest of the procedure is same to find the unknown vector x. We use Gaussian elimination method to produce the LU decomposition. In this method we subtract the first equation from the second equation to remove the variables of the first equation from the second equation. Similarly, we subtract the second equation from the third

equation to remove the variables of the second equation. We repeat this procedure until the remaining system is in an upper triangular form. This upper triangular matrix is the matrix U and the matrix L is comprised of the row multipliers that cause variables to be eliminated.

Now suppose we have an $n \times n$ matrix A, we divide this matrix into four parts to compute its LU decomposition.[4]

$$A = \begin{bmatrix} a_{11} & w^T \\ v & A^{'} \end{bmatrix}$$

where, v is a column (n-1) vector, w^T is a row (n-1) vector, and $A^{'}$ is an $(n-1) \times (n-1)$ matrix. We can factor A as

$$A = \begin{bmatrix} a_{11} & w^T \\ v & A^{'} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A^{'} - vw^T/a_{11} \end{bmatrix} \tag{1.14}$$

The $(n-1) \times (n-1)$ matrix $A^{'} - vw^T/a_{11}$ is the **schur complement** of A with respect to a_{11}. Because schur complement is non-singular, so we can now recursively find an LU decomposition for it. Let us say that
$A^{'} - vw^T/a_{11} = L^{'}U^{'}$
where $L^{'}$ is unit lower triangular matrix and $U^{'}$ = upper triangular matrix. Then, using matrix algebra we have

$$A = \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A^{'} - vw^T/a_{11} \end{bmatrix} \tag{1.15}$$

$$= \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & L^{'}U^{'} \end{bmatrix} \tag{1.16}$$

$$= \begin{bmatrix} 1 & 0 \\ v/a_{11} & L^{'} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & U^{'} \end{bmatrix} \tag{1.17}$$

$$= LU \tag{1.18}$$

thereby rendering our LU decomposition. Since $L^{'}$ is a unit lower triangular matrix, so is L, and because $U^{'}$is upper triangular, so is U.

"Here a_{11} cannot be zero otherwise this method doesn't work. Similarly, the upper left most entry of the schur complement $A^{'} - vw^{T}/a_{11}$ is also cannot be zero, since we divide by it in the next step of the recursion. The elements by which we divide during LU decomposition are called pivots, and they occupy the diagonal elements of the matrix U. The reason we include a permutation matrix P during LUP decomposition is that it allows us to avoid dividing by zero. When we use permutations to avoid division by zero (or by small numbers, which would contribute to numerical instability), we are pivoting."[4]

1.6.2 Computing an LUP-decomposition

The procedure to compute the LUP decomposition of a matrix is almost same as LU decomposition. The only difference is that in LU decomposition, in solving a system of linear equation Ax = b, we pivot on off diagonal elements of A to avoid dividing by zero but in LUP decomposition, we also use to avoid dividing by a small number, even if A is nonsingular, because numerical stabilities can result. We therefore pivot on a large value. To do this, we interchange the rows with the rows whose first element has largest absolute value in each step. Rest of the procedure is same as the LU-decomposition

Chapter 2

Ellipsoid Algorithm

The ellipsoid algorithm is the first polynomial time algorithm discovered for linear programming. It was proposed by Russian mathematicians D. B. Yudin and A. S. Nemirovskii and then clarified by N. Z. Shor in 1977 for general convex optimization problems and applied to linear programming by Khachyan in 1979. Khachyan modified the method to obtain a polynomial time algorithm for the feasibility problem for a system of linear inequalities[5]. He proved that LP is solvable in polynomial time by a method of shrinking ellipsoids.

The problem being considered by the ellipsoid algorithm is[6] −
"Given a bounded convex set $P \in \mathbb{R}^n$, find $x \in P$".
The input for the ellipsoid algorithm is a convex set and the output is a point from the set, provided it is nonempty (we return empty if the set is empty). It is possible to determine feasibility or infeasibility of a system of linear inequalities in polynomial time with this algorithm. Formally, the ellipsoid algorithm checks if a given convex set $P \subseteq \mathbb{R}^n$ is empty.

To understand the ellipsoid method, we need some definitions that will be useful throughout the chapter −

2.1 Definitions

- **Linear Programming** : − It can be described by the following (Primal) optimization problem

$$\begin{array}{rl} \text{maximize} & c^T x \\ \text{subject to} & Ax \leq b \\ \text{and} & x \geq 0 \end{array}$$

In words we can describe this as "The process of maximizing a linear objective function, subject to a finite number of linear equality and inequality constraints".

- **Convex Set** : − A convex set[7] $P \subseteq \mathbb{R}^n$ is a set of points such that $\forall x, y \in P$ $\lambda x + (1 - \lambda)y \in P$, where $\lambda \in [0, 1]$. A convex body is a closed and bounded convex set. The whole space $\mathbb{R}^n$ is trivially an infinite convex set.

- **Ellipsoid** : − An ellipsoid E(a,A) is defined as

$$E(a, A) = \{x \in \mathbb{R}^n : (x - a)^T A^{-1}(x - a) \leq 1\}$$

where 'a' is a center and 'A' is a positive definite matrix.
-one important fact about a positive definite matrix A, is that there exists B, such that $A = B^T B$ and hence $A^{-1} = B^{-1}(B^{-1})^T$ and $x^T Ax > 0$ for all nonzero $x \in \mathbb{R}^n$.

- **Ball** : − A (closed) ball B(a,r) (in $\mathbb{R}^n$) centered at $a \in \mathbb{R}^n$ with radius r is the set

$$B(a, r) = \{x \in \mathbb{R}^n : x^T x \leq r^2\}.$$

The set B(0,1) is called the unit ball. The unit sphere is the set

$$S_n = \{x \in \mathbb{R}^n : x^T x \leq 1\}.$$

- **Hyperplane** : − A hyperplane is defined to be the set of points satisfying the linear equation

$$ax = b$$

where $a, x, b \in \mathbb{R}^n$. A hyperplane of an n-dimensional space, is a flat subset with dimension n-1. Hyperplane separates the space into two half spaces.

- **Affine Transformation** : − The transformation $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ defined as

$$T(x) = t + Q.x$$

for each $x \in \mathbb{R}^n$, is called an affine transformation, where Q is an $'n \times n'$ nonsingular matrix and t is an n vector.

$*$ If T is an affine transformation, then $T(S_n)$ is called an ellipsoid. Alternatively

$$T(S_n) = \{y \in \mathbb{R}^n : (y-t)^T B^{-1}(y-t) \leq 1\}$$

where B is a positive definite matrix such that $B = QQ^T$ and $x^T Bx > 0$ for all nonzero $x \in \mathbb{R}^n$. So, we can say that ellipsoids are just the affine transformations of unit spheres.

Lemma 2.1.1. *Ellipsoids are the affine transformations of unit spheres.*

Proof. $E = T(S_n)$
$= \{T(x) \mid x \in S_n\}$
$= \{t + Q.x \mid \|\mathbf{x}\| \leq 1\}$
$= \{y \mid \|Q^{-1}(y-t)\| \leq 1\}$
$= \{y \mid (y-t)^T (Q^{-1})^T Q^{-1}(y-t) \leq 1\}$
$= \{y \mid (y-t)^T B^{-1}(y-t) \leq 1\}$, where $B = QQ^T$ □

Lemma 2.1.2. *If $S \subseteq S'$, then $T(S) \subseteq T(S')$.[8]*

Lemma 2.1.3. *Suppose that a subset S of $\mathbb{R}^n$ has volume V. Then T(S) has volume $V.|det(Q)|$.[8]*

Before going further we should know that in this algorithm why we use only ellipsoids as the fundamental geometric object of our investigation. Ellipsoids comprise the simplest class of n-dimensional convex sets which is closed under non-singular affine maps. if we replace ellipsoid by some other geometric object then this makes the formulas and the analysis of the algorithm substantially more unpleasant than those for the ellipsoids.

2.2 The Basic Ellipsoid Algorithm

It is an iterative method for minimizing convex function. This method finds an optimal solution of a linear optimization problem, in a finite number of steps. This method generates a sequence of ellipsoids whose volume uniformly decreases at every step, thus enclosing a minimizer of a convex function.

1. The idea behind the ellipsoid algorithm is that we start with an initial big ellipsoid that ensures P is contained in it, i.e. the solution set of $Ax \leq b$.

2. In each step, the center of the ellipsoid is a candidate for a feasible point of the problem. We then check, if the center of the ellipsoid is in P. If it is, we are done, we found a point in P. If the center a_k of E_k is not in P, we get a violated inequality $Ax > b$, satisfied by a_k, not satisfied by P (if P is nonempty).

3. Because of the inequality violation we construct another ellipsoid of smaller volume which has a different center. Let E_{k+1} be the minimum volume ellipsoid containing $E_k \cap \{x : c^T x \leq c^T a_k\}$, where $c^T x \leq c^T a_k$ be the half space through a_k that contains the convex set P. Apparently each of the constraints in a linear program defines half spaces (which are parts of $\mathbb{R}^n$ bounded on one side by hyperplanes) and the solution lies in the intersection of these half spaces.[7]

4. We repeat the procedure until either a feasible point is found or a maximum number of iterations is reached. This means that after enough iterations either a solution must be found or it is certain that the ellipsoid has become too small to contain a solution through successive shrinkings and we report that the inequality set has no feasible point.

Thus we can observe that in each step of the ellipsoid algorithm the new ellipsoid shrinks in volume. This implies that if the convex set P has positive volume, we will finally find a point in P. So, now we are in a situation to deal with the case when volume of the convex set P is zero i.e. when P has just a single point. On the other hand it is also very important to discuss when we can stop and be ensure that either we have a point in P or we certain that P is empty[6].

2.3 Basic Ellipsoid Iterations

Let E_0,E_1,E_2,.....,E_j,..... be the sequence of ellipsoids generated by the ellipsoid algorithm. Each of the ellipsoid contains a point which satisfies the constraint equation $A^T x \leq b$, if one exists. Let E_j be defined as follows[9, 8] :

$$E_j = \{x \in \mathbb{R}^n : (x - t_j)^T {B_j}^{-1} (x - t_j) \leq 1\}$$

. Then we can obtain the variables for the next iteration by using the following formulas.

$$t_{j+1} = t_j - \frac{\rho.B_j.a}{\sqrt{a^T B_j a}}$$

,

$$B_{j+1} = \eta(B_j - \frac{\gamma(B_j a)(B_j a)^T}{a^T B_j a})$$

where, $\rho = \frac{1}{n+1}$, $\gamma = \frac{2}{n+1}$ and $\eta = \frac{n^2}{n^2-1}$. ρ is defined as the step parameter, while γ and η are the dilation and expansion parameters respectively.

Lemma 2.3.1. *Let $a \in \mathbb{R}^n$ be a vector of length $\|a\|$. There is a rotation R such that [8]*

$$Ra = (\|a\|,0,.....,0).$$

If $P \subseteq \mathbb{R}^n$ is a convex set and $x \in \mathbb{R}^n$ is a point, then one of the following holds[7] −

1. $x \in P$
2. There exists a hyperplane that separates x from P.

This motivates the following definition of a polynomial time separating oracle.

2.4 Separation Oracle

To make the ellipsoid method executable, we need to be able to decide, given $x \in \mathbb{R}^n$ whether $x \in P$ or find a inequality. To do this, we require a separation oracle for P. A polynomial time separating oracle for a convex set P is a procedure which given x either decide x is a feasible point for LP i.e. $x \in P$ or returns a hyperplane separating x from P .

2.5 Correctness of the algorithm

The correctness of the ellipsoid algorithm follows from the next theorem.[8]

Theorem 2.5.1. *Let B_j be a positive definite matrix. Let $t_j \in \mathbb{R}^n$ and let 'a' be any nonzero n-vector. Let B_{j+1} and t_{j+1} be as in step-3 (after j^{th} iteration) of the ellipsoid algorithm. Then the following holds : −[8]*

1. *B_{j+1} is positive definite (or equivalently $E_{j+1} = \{x \in \mathbb{R}^n : (x-t_{j+1})^T {B_{j+1}}^{-1}(x-t_{j+1}) \leq 1\}$ is an ellipsoid.)*

2. *The semiellipsoid $\frac{1}{2}E_j[a] = \{x \in \mathbb{R}^n : (x-t_j)^T {B_j}^{-1}(x-t_j) \leq 1, a^T(x-t_j) \leq 0\}$ is a subset of E_{j+1}.*

3. *The volume of E_j and E_{j+1} satisfy*

$$\frac{vol(E_{j+1})}{vol(E_j)} < e^{\frac{-1}{2(n+1)}}$$

To prove this theorem we need two auxiliary lemmas.

Lemma 2.5.2. *Consider the sphere S_n and the set $E = \{x \in \mathbb{R}^n : (x-t)' B^{-1}(x-t) \leq 1\}$, where $t = (-1/(n+1), 0, ..., 0)'$ and $B = diag(n^2/(n+1)^2, n^2/(n^2-1), ..., n^2/(n^2-1))$.[8]*

1. *B is positive definite (and hence E is an ellipsoid).*

2. *The hemisphere $\frac{1}{2}S_n = \{x \in \mathbb{R}^n : x'x \leq 1$ and $x_1 \leq 0\}$ is a subset of E.*

3. *The volumes of S_n and E satisfy*

$$\frac{vol(E_{j+1})}{vol(S_n)} < e^{\frac{-1}{2(n+1)}}$$

Proof. 1. We can write B as, $B = QQ^T$,where[8]

$$Q = diag\left(\frac{n}{(n+1)}), c, ..., \frac{n}{\sqrt{n^2-1}}\right)$$

2. Suppose that $x \in \frac{1}{2}S_n$. Then

$$\begin{aligned}(x-t)' B^{-1}(x-t) &= \frac{(n+1)^2}{n^2}\left(x_1 + \frac{1}{(n+1)}\right)^2 + \frac{n^2-1}{n^2}\sum_{i=2}^{n} {x_i}^2 \\ &= \frac{n^2-1}{n^2}x'x + \frac{2n+2}{n^2}{x_1}^2 + \frac{2n+2}{n^2}x_1 + \frac{1}{n^2} \\ &\leq 1 + \frac{2n+2}{n^2}{x_1}^2 + x_1 \qquad \text{(because } x \in S_n\text{)} \\ &\leq 1 \qquad \text{(because } x \in \frac{1}{2}S_n\text{)}\end{aligned}$$

3. By lemma (2.1.3), where Q is as in part (1).
   Since Q is diagonal, det Q $= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2}$.

   Since $\forall x > 0$, $1 + x \leq e^x$, $1 - x \leq e^{-x}$

   so, $\frac{n}{n+1} = 1 - \frac{1}{n+1} \leq e^{-1/(n+1)}$

   $\Rightarrow \frac{n^2}{n^2-1} = 1 + \frac{1}{n^2-1} \leq e^{1/(n^2-1)}$.

   Hence, $detQ < exp\left(\frac{n-1}{2(n^2-1)} - \frac{1}{n+1}\right) < 2^{-1/2(n+1)}$

□

Lemma 2.5.3. *Let B_j be a positive definite matrix, $t_j \in \mathbb{R}^n$, and let 'a' be any nonzero n-vector. Let B_{j+1} and t_{j+1} be obtained as in step-3 of the ellipsoid algorithm. Let $\frac{1}{2}S_n$ and E be as in previous lemma. Then there exist an affine transformation T such that[8]*

1. $T(S_n) = \{x \in \mathbb{R}^n : (x - t_j)' {B_j}^{-1}(x - t_j) \leq 1\}$;
2. $T(E) = \{x \in \mathbb{R}^n : (x - t_{j+1})' {B_{j+1}}^{-1}(x - t_{j+1}) \leq 1\}$;
3. $T(\frac{1}{2}S_n) = \{x \in \mathbb{R}^n : (x - t_j)' {B_j}^{-1}(x - t_j) \leq 1, a'(x - t_j) \leq 0\}$

Proof. By hypothesis, B_j is positive definite and hence $B_j = QQ^T$ for some nonsingular matrix Q. Also by lemma (2.3.1) there exists a rotation R^T such that R^TQ^Ta = ($\|Q^Ta\|$,0,.....,0)'. The transformation T is defined thus $T(x) = t_j + QRx$. Now, we shall check the three conditions.[8]

1. $T(S_n) = \{T(x) : x'x \leq 1\}$
   $= \{x : (T^{-1}(x))'T^{-1}(x) \leq 1\}$
   $= \{x : (x - t_j)'(Q^{-1})^T RR^T Q^{-1}(x - t_j) \leq 1\}$
   $= \{x : (x - t_j)' {B_j}^{-1}(x - t_j) \leq 1\}$

2. First notice that

$$B_{j+1} = \frac{n^2}{n^2-1}\left[B_j - \frac{2}{n+1}\frac{B_j a a' {B_j}^T}{a' B_j a}\right]$$

$$= \frac{n^2}{n^2-1}\left[B_j - \frac{2}{n+1}\frac{QRR^TQ^Taa'QRR^TQ^T}{a'QRR^TQ^Ta}\right]$$

$$= \frac{n^2}{n^2-1}\left[B_j - \frac{2}{n+1}\frac{QR(\|Q^Ta\|^2,0,.....,0)R^TQ^T}{\|Q^Ta\|^2}\right]$$

(Because $R^TQ^Ta = (\|Q^Ta\|,0,.....,0)'$)

$$= \frac{n^2}{n^2-1}\left[B_j - \frac{2}{n+1}QRdiag(1,0,.....,0)R^TQ^T\right]$$

$$= \frac{n^2}{n^2-1}QR\ \text{diag}\left(\frac{n-1}{n+1},1,.....,1\right)R^TQ^T$$

$$= QRBR^TQ^T$$

where B is as in lemma (2.5.2). Also,

$$(x-t_{j+1}) = \left(x - t_j + \frac{QRR^TQ^Ta}{(n+1)\sqrt{a'QRR^TQ^Ta}}\right)$$

$$= \left(x - t_j + \frac{QR(\|Q^Ta\|,0,.....,0)'}{(n+1)\|Q^Ta\|}\right)$$

$$= QR(T^{-1}(x) - t)$$

Therefore,

$T(E) = \{T(x) : (x-t)'B^{-1}(x-t) \leq 1\}$

$= \{x : (T^{-1}(x)-t)'B^{-1}(T^{-1}(x)-t) \leq 1\}$

$= \{x : (x-t_{j+1})'(Q^{-1})^TRB^{-1}R^TQ^{-1}(x-t_{j+1}) \leq 1\}$

$= \{x : (x-t_{j+1})'{B_{j+1}}^{-1}(x-t_{j+1}) \leq 1\}$

3. The condition in (3) now follows easily from the condition in (1) and lemma (2.1.2) by observing that

$$T(\{x \in \mathbb{R}^n : x_1 \leq 0\}) = \{x \in \mathbb{R}^n : a'(x-t_j \leq 0)\}.$$

□

Proof of theorem (2.5.1)[8]

Proof. 1. By (2) of lemma (2.5.3) $T(E) = E_{j+1}$, also by (1) of lemma (2.5.2) $E = T'(S_n)$ for some affine transformation T'. Hence $E_{j+1} = T.T'(S_n)$ is an ellipsoid (the comparison of two affine transformations is also an affine transformation).

2. $E_j[a] = T\left(\frac{1}{2}S_n\right)$ by (3) of lemma (2.5.3) and $\frac{1}{2}S_n \subseteq E$. Hence $E_j[a] \subseteq T(E) = E_{j+1}$.

3. By lemma (2.1.3) and by (3) of lemma (2.5.2)

$$\frac{Vol(E_{j+1})}{Vol(E_j)} = \frac{Vol(T(E))}{Vol(T(S_n))} = \frac{det(QR)Vol(E)}{det(QR)Vol(S_n)} < 2^{-1/2(n+1)}$$

□

2.6 Time complexity

The time complexity of the ellipsoid algorithm depends on the time taken by the separation oracle, time required to find E_{j+1} and the ratio $\frac{V_u}{V_l}$, where V_u is the upper bound on the volume of the convex set P and V_l is the lower bound on Vol(P).

For linear programming, time taken by the separation oracle is $O(mn)$ as all we have to do is check whether the convex set P satisfies all the constraints and returns a hyperplane (if it exists).

Lemma 2.6.1. *The minimum volume ellipsoid surrounding a half ellipsoid can be calculated in polynomial time and*

$$Vol(E_{j+1}) \leq (1 - \frac{1}{2n})Vol(E_i)$$

.

If the while loop iterates t times , then $(1-\frac{1}{2n})^t \leq \frac{V_u}{V_l} \Rightarrow$ t $= O(nlog(\frac{V_u}{V_l}))$. Suppose we need L bits to represent our input, we can choose[7] $V_l = 2^{-c_1 nL}$ and $V_u = 2^{c_2 nL}$ for some constants c_1, c_2. This gives us the time t $= O(n^2L)$. Hence, the ellipsoid algorithm terminates after $O(n^2L)$ iterations where each iteration taking polynomial time.

The ellipsoid algorithm is theoretically very important and valuable tool to "analyzing the complexity of optimization problems" but in practice, it is not very useful. This method is very slow, compared to the other methods like simplex method, interior point method, karmakars method etc. But since it was the first polynomial time algorithm and its theoretical base is very strong, makes this method incredibly useful.

Chapter 3

Primal Dual Algorithm

The Primal dual method is a standard tool to understand the combinatorial optimization problems that can be formulated as linear programmes and to design the algorithms. This method was proposed by Dantzig, Ford and Fulkerson.

3.1 Introduction:

Linear programming (LP or linear optimization) is a technique for find out a way to achieve the best outcome (such as maximum profit or lowest cost) in a given mathematical model for some list of requirements represented as linear relationships.

More formally, it is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints.

A generic LP can be expressed as:

$$\begin{array}{rl} \text{maximize} & c^T x \\ \text{subject to} & Ax \leq b \\ \text{and} & x \geq 0 \end{array}$$

where, x is a vector of variables, c and b are vectors of coefficients, and A is a matrix of coefficients.
The expression $c^T x$ is called the objective function which we have to maximize or minimize. The constraints are represented by the inequalities $Ax \leq b$ and $x \geq 0$, over which the objective function is to be optimized.

3.2 Duality

We may view the optimization problems in either of two aspects - the Primal problem or the dual problem .

Every primal problem can be converted into a dual problem. The solution of the dual problem provides an upper bound to the optimal value of the primal problem. However, in general the optimal values of the primal and dual problems need not be equal.
In matrix form, primal problem can be expressed as -

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ \text{and} \quad & x \geq 0 \end{aligned}$$

and the corresponding symmetric dual problem can be expressed as -

$$\begin{aligned} \text{minimize} \quad & b^T y \\ \text{subject to} \quad & A^T y \geq c \\ \text{and} \quad & y \geq 0 \end{aligned}$$

3.2.1 Relationships between the primal and the dual problems

[10]

1. **Dual of a dual problem is the original primal problem.**

   Proof. : Let the primal problem is

   $$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ \text{and} \quad & x \geq 0 \end{aligned}$$

   and its dual is

   $$\begin{aligned} \text{minimize} \quad & y^T b \\ \text{subject to} \quad & A^T y \geq c \\ \text{and} \quad & y \geq 0 \end{aligned}$$

We can rewrite the dual as a maximization problem

$$\begin{aligned} \text{maximize} \quad & -y^T b \\ \text{subject to} \quad & -A^T y \leq -c \\ \text{and} \quad & y \geq 0 \end{aligned}$$

take its dual-

$$\begin{aligned} \text{minimize} \quad & -c^T z \\ \text{subject to} \quad & -(A^T)^T z \geq -b \\ \text{and} \quad & z \geq 0 \end{aligned}$$

□

2. Every feasible solution for a primal problem gives a bound on the optimal value of the objective function of its dual.

3. Primal problem has an optimal solution if and only if its dual problem has an optimal solution.

Now, to understand the primal and the dual LP problems, we first take an example called Production problem as our primal problem and then we will go through its dual, called Pricing problem.

3.3 Economic interpretation of dual problem

3.3.1 Production problem

Suppose a firm produces chairs and tables. We know that a generic Primal LP can be expressed as:

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ \text{and} \quad & x \geq 0 \end{aligned}$$

Let's assume that
c = Per unit price of a chair or a table
b = Total amount of resource (woods or metal)
x = Production of chairs or tables
a_{ij} = Units of resource i used in making per unit of product j, and
r_j \$ = Per unit profit for product j
$j \in \{C(chair), T(table)\}$
$i \in \{w(wood), m(metal)\}$

We define the matrices as-

$$A = \begin{bmatrix} w_C & w_T \\ m_C & m_T \end{bmatrix}, x = \begin{bmatrix} C \\ T \end{bmatrix}, b = \begin{bmatrix} b_w \\ b_m \end{bmatrix}, c = \begin{bmatrix} c_C \\ c_T \end{bmatrix} \tag{3.1}$$

where,
w_C = wood in chair, m_C = metal in chair, w_T = wood in table, m_T = metal in table, b_w = total amount of woods, b_m = total amount of metal, c_C = price of a chair, c_T = price of a table.
Therefore, by the inequality

$$Ax \leq b$$

$$\begin{bmatrix} w_C & w_T \\ m_C & m_T \end{bmatrix} \begin{bmatrix} C \\ T \end{bmatrix} \leq \begin{bmatrix} b_w \\ b_m \end{bmatrix} \tag{3.2}$$

Production problem : How much of tables or chairs should be produce that yields the maximum profit ? or in other words determine an allocation of resources to products that yields the maximum profit.

3.3.2 Pricing problem

Suppose there is an another firm who wants to buy all the resources which is in our case, woods and metal. Should the first firm sell all its resources or in which proportion should the first firm sell its resources or should the firm continue to be

with making chairs and tables ? All depends on by which way the firm can get the more profit.
On the other hand the second firm would like to buy all the resources at the minimum price so that they could sell it in market at some higher price and can get more profit. Suppose y_i denotes the per unit price for resource i offered by the buyer then when is such price acceptable to the first firm ?
If the first firm accept the prices y_i, then all the resource will be sold off for $\sum_i b_i y_i$ dollar.
Hence, the dual of production problem which is the point of view of the buyer -

$$\begin{array}{cc} \text{minimum} & y^T b \\ \text{subject to} & A^T y \geq c \\ \text{and} & y \geq 0 \end{array}$$

Pricing problem : Find equilibrium price at which the deal can be closed.
y_i, is called the shadow price for the resource i.

Objective imposed by buyer's view point : Minimize total price paid

$$p^* = min \sum_i b_i y_i.$$

Constraints imposed by seller's present performance : For each product j, total price paid for resources used per unit $\geq$ unit profit for product j:

$$w_C y_w + m_C y_m \geq c_C,$$

$$w_T y_w + m_T y_m \geq c_T.$$

3.3.3 For 1×2 matrices(one resource and two products)

Suppose the firm produces tables and chairs and use only woods as resource. Then our primal problem would be :

$$\begin{array}{cc} \text{maximize} & P_t T + P_c C \\ \text{subject to} & W_t T + W_c C \leq W \\ \text{and} & T, C \geq 0 \end{array}$$

where,
T = total number of tables, C = total number of chairs, P_t = price per unit of table,

P_c = price per unit of chair, W = total amount of wood, W_t = wood used in per unit of table, W_c = wood used in per unit of chair.

Then the dual of this primal problem according to the pricing problem would be :

$$\begin{array}{cc} \text{minimize} & WP_W \\ \text{subject to} & P_W \geq \{P_c/W_c, P_t/W_t\} \\ \text{and} & P_W \geq 0 \end{array}$$

where P_W is the per unit price of the wood.
Dual problem says that the suppose an another firm want to buy all the resources which is, in our case, wood. Then the buyer would like to buy all the resources at the minimum price. From the point of view of the seller, the selling price of wood, used in per chair or table should be greater or equal to the per unit price of a chair or table. So,

$$W_c P_W \geq P_c \tag{3.3}$$

or

$$W_t P_W \geq P_t \tag{3.4}$$

then

$$P_W \geq P_c/W_c$$

and

$$P_W \geq P_t/W_t$$

Suppose the seller gets more profit in making chairs, but because of customers he has to make tables also. So, when he will sell his resource he would like to maximize his profit. So,

$$P_W \geq max(P_c/W_c, P_t/W_t) \tag{3.5}$$

Therefore, when he will make only chairs

$$P_W = P_c/W_c \Rightarrow W_c P_W = P_c$$

So, max $P_c C$ = min WP_W. Hence it is satisfying the complementary slackness and it is also satisfying the strong duality.

3.3.4 For 2×1 matrices (2 resource and 1 product)

Suppose a shop of sweets makes mishty dahi. Milk and sugar are used as resources to make the mishty dahi. So our primal problem is −

$$\begin{aligned} \text{maximize} \quad & P_dQ = P_dS/d_s = P_dM/d_m \\ \text{subject to} \quad & d_sQ \leq S \\ & d_mQ \leq M \\ \text{and} \quad & Q \geq 0 \end{aligned}$$

where,
d_s = amount of sugar used in per unit of mishty dahi, d_m = milk used in per unit of mishty dahi, P_d = price per unit of mishty dahi, S = total amount of sugar, M = total quantity of milk and Q = min{S/d_s, M/d_m} = maximum no. of times, we can make mishty dahi.

The dual of this problem would be -

$$\begin{aligned} \text{minimize} \quad & O_sS + O_mM \\ \text{subject to} \quad & O_sd_s + O_md_m \geq P_d \\ \text{and} \quad & O_s, O_m \geq 0 \end{aligned}$$

where,
O_s = per unit price of sugar, O_m = per unit price of milk.
Then, using the equation of constraints-

$$O_s \geq (P_d - O_md_m)/d_s \tag{3.6}$$

⇒,

$$min(S(P_d - O_md_m)/d_s + O_mM) = SP_d/d_s + O_md_m(M/d_m - S/d_s) \tag{3.7}$$

If the amount of sugar is less than the milk means that after making maximum units of mishty dahi, if we do not have the sugar in an amount so that we can make one more unit of mishty dahi then the equation (3.7) becomes −

$$min(S(P_d - O_md_m)/d_s + O_mM) = SP_d/d_s \tag{3.8}$$

which is equal to the objective function of our primal problem. Similarly, using the constraint equation we can write-

$$O_m \geq (P_d - O_s d_s)/d_m \tag{3.9}$$

$\Rightarrow$,

$$min(SO_s + M(P_d - O_s d_s)/d_m) = O_s d_s(S/d_s - M/d_m) + MP_d/d_m \tag{3.10}$$

Now, if the quantity of milk is less than the sugar then equation (3.10) becomes -

$$min(SO_s + M(P_d - O_s d_s)/d_m) = MP_d/d_m \tag{3.11}$$

which is equal to the objective function of our primal problem.

Hence, for the optimal solution -
min $(O_s S + O_m M)$ = max $P_d Q$, where Q = min{S/d_s, M/d_m}

3.4 Dual problem derived algebraically

We write the primal problem as

$$\begin{array}{rl} \text{maximize} & c^T x \\ \text{subject to} & Ax \leq b \\ \text{and} & x \geq 0 \end{array}$$

where, $c^T x = c_1 x_1 + c_2 x_2 + \ldots\ldots\ldots\ldots + c_n x_n$, $Ax = \sum_{i,j} a_{ij} x_j$ and $b = b_i$,i,j = 1,2,.....n.

In general, we can write the problem in the form of linear equations. So, the system of linear equations is -

$$\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \ldots\ldots\ldots + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots\ldots\ldots + a_{2n}x_n \leq b_2 \\ \ldots\ldots \quad \ldots\ldots \quad \ldots\ldots \\ \ldots\ldots \quad \ldots\ldots \quad \ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots\ldots\ldots + a_{nn}x_n \leq b_n \end{array}$$

Now multiply all the linear equations by the corresponding y_i and add them. Then we get the equation

$$C_1x_1 + C_2x_2 + ... + C_nx_n \leq \sum_i b_iy_i \tag{3.12}$$

where i = 1,2,....,n and

$$\begin{array}{rcl} C_1 & \leq & \sum_i a_{i1}y_i \\ C_2 & \leq & \sum_i a_{i2}y_i \\ .... & ..... & .... \\ .... & ..... & .... \\ C_n & \leq & \sum_i a_{in}y_i \end{array}$$

Therefore,

$$\sum_j C_jx_j \leq \sum_{i,j\leq n}(\sum_{i,j} a_{ij}y_i)x_j \tag{3.13}$$

for j = 1,2,.....,n. We can write it as

$$\sum_j C_jx_j \leq \sum_{i,j}(\sum_{i,j\leq n} a_{ij}x_j)y_i \tag{3.14}$$

and we know that

$$\sum_{i,j\leq n} a_{ij}x_j \leq b_i$$

Hence,

$$\sum_j C_jx_j \leq \sum_i b_iy_i \tag{3.15}$$

3.5 Weak Duality theorem

Theorem 3.5.1. *The objective function value of the dual at any feasible solution is always greater than or equal to the objective function value of the primal at any feasible solution.*

Proof. : Let $x_j = \{x_1, x_2, ..., x_n\}$ be the primal feasible solutions and $y_i = \{y_1, y_2, ..., y_m\}$ be the dual feasible solutions. Then we have to prove that[11]

$$\sum_j c_jx_j \leq \sum_i b_iy_i$$

$\Rightarrow$

$$\begin{aligned}\sum_j c_j x_j &\leq \sum_j (\sum_i a_{ij} y_i) x_j \\ &= \sum_{ij} a_{ij} y_i x_j \\ &= \sum_i (\sum_j a_{ij} x_j) y_i \\ &\leq \sum_i b_i y_i\end{aligned}$$

□

3.6 Complementary Slackness

In general, given a problem

$$\begin{array}{rl} \text{maximize} & \sum_{j=1}^{n} c_j xj \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \leq bi \quad (i = 1, 2, ..., m) \\ \text{and} & x_j \geq 0 \quad (j = 1, 2, ..., n) \end{array}$$

We define the slack variables $x_{n+1}, x_{n+2}, ..., x_{n+m}$ by

$$x_{n+1} = bi - \sum_{j=1}^{n} a_{ij} x_j \quad (i = 1, 2, ..., m)$$

and the objective function, z by

$$z = \sum_{j=1}^{n} c_j xj$$

From the weak duality theorem , let us denote

$$\triangle(x, y) = y^T(b - Ax) + (y^T A - c^T)x = 0$$

then either

$$y^T(b - Ax) = 0$$

or

$$(y^T A - c^T)x = 0$$

Let $y^T(b - Ax) = 0$. If $y^T = 0$ then $(b - Ax) > 0$. If $(b - Ax) = 0$ then $y^T > 0$ and hence, $b = Ax$. Similarly Let $(y^T A - c^T)x = 0$. If x = $0 \implies (y^T A - c^T) > 0$ and if $x > 0 \implies (y^T A - c^T) = 0$ and hence $y^T A = c^T$.

So, If the i^{th} slack variable of the primal is not zero, then the i^{th} variable of the dual is equal to zero. Similarly if the j^{th} slack variable of the dual is not zero then the j^{th} variable of the primal is equal to zero.

3.6.1 Complementary Slackness Theorem

Theorem 3.6.1. *A primal feasible solution* $\{x_1^*, x_2^*, x_3^*, ..., x_n^*\}$ *and a dual feasible solution* $\{y_1^*, y_2^*, y_3^*, ..., y_m^*\}$ *are both optimal if and only if [12]*

$$x_j^* > 0 \quad \Rightarrow \quad \sum_{i=1} a_{ij} y_i^* = c_j \qquad \forall j \in [n]$$

and

$$y_i^* > 0 \quad \Rightarrow \quad \sum_{i=1} a_{ij} x_j^* = b_i \qquad \forall i \in [m]$$

Proof of this follows from the above definition of Complementary Slackness.

3.7 Strong Duality theorem

Theorem 3.7.1. *If the primal has an optimal solution* $(x_1{}^*, x_2{}^*, ..., x_n{}^*)$*, then the dual also has an optimal solution,* $(y_1{}^*, y_2{}^*, ..., y_m{}^*)$ *such that[12]*

$$c^T x^* = b^T y^*$$

or

$$\sum_{j=1}^{n} c_j xj^* = \sum_{j=1}^{n} b_i yi^*$$

The proof of this theorem is very tedious and not very insightful. So, we are not going to prove this theorem here.
Strong duality is a very powerful tool to solve many optimization problems. It is very hard to solve some problems in their primal form in compare to their dual form, so solving the dual can be far quicker than solving the primal. If we know that dual solution is optimal, then we can say that the primal solution is also optimal and both optimal solutions would be equal and unique by strong duality.

Chapter 4

Strong Duality and Max-Flow Min-Cut Problem

In this chapter we discuss about the max-flow and the min-cut problem. Min-cut problem is the dual of the max-flow problem. So, this chapter is all about to explain how to get the dual from the primal for the network flow problem. At the end of the chapter we prove the max-flow min-cut theorem using the strong duality.

4.1 The Primal-Dual Method Applied to Max-Flow Min-Cut Problem

The first recorded application of max-flow min-cut Problems came into existence in the mid-1950s. A report on the rail networks that linked the soviet union to its satellite countries in eastern europe, was published by the air-force researchers T. E. Harris and F. S. Ross. The network was modeled as a graph of 44 vertices and 105 edges. Vertices represents the geographic regions and edges represents links between those regions in the rail network. The rate at which material could be shipped from one region to the next was represented by the weight given to each edge. They determined both the maximum amount of the stuff that could be moved from Russia into Europe, essentially by trial and error, as well as the cheapest way to disrupt the network by removing the links or in other words by blowing up the train tracks.[13]
For both problems, the input is a directed graph G = (V,E) along with a source s and target t. $u \to v$ denote the directed edge from vertex u to vertex v. The maximum flow problem is to find the maximum amount of material that can be transported from one vertex to another; the minimum cut problem asks for the minimum damage

needed to separate two vertices.

4.1.1 Max-Flow Problem

A flow network is a directed graph G with two special nodes denoted by s,t, where s is a source node out which flow leaves and t is a sink node in which flow arrives. E is the set of edges of the network which represent pipes that carry flow and V is the set of vertices. c(u,v) represents the maximum capacity of each edge (u,v).

To understand the maximum flow problem we use a graph to model material that flows through pipes. Each edge represents a pipe and has a capacity which is an upper-bound on the flow rate in units/time. The pipes can be of different sizes. So, our problem is to compute maximum rate that we can ship material from a designated source to a designated sink.

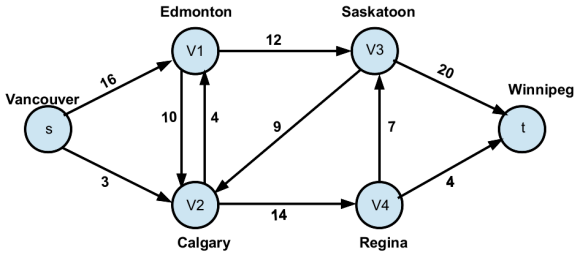

Figure 4.1:

In a flow network each edge (u,v) has a non-negative capacity c(u,v). If (u,v) is not in E, we assume that c(u,v) = 0. We assume that every vertex v in V is on some path from s to t.
e.g. $c(s, v_1) = 16$, $c(v_1, s) = 0$, $c(v_2, v_3) = 0$
For each edge (u,v), the flow f(u,v) is a real valued function that must satisfy three conditions :

1. Capacity constraint : $\forall$ u,v $\in$ V, f(u,v) $\leq$ c(u,v)
2. Skew symmetry : $\forall$ u,v $\in$ V, f(u,v) $\leq -$f(v,u)
3. Flow conservation : $\forall u \in V - \{s, t\}$, $\sum_{v \in V}$ f(u,v) = 0

Note : - The skew symmetry condition implies that f(u,u) = 0.
-We show only the positive flows in the network.

Example of a Flow

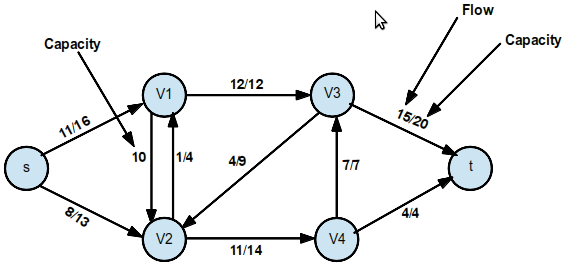

Figure 4.2:

$f(v_2, v_1) = 1$, $c(v_2, v_1) = 4$
$f(v_1, v_2) = -1$, $c(v_1, v_2) = 10$ and
$f(v_3, s) + f(v_3, v_1) + f(v_3, v_2) + f(v_3, v_4) + f(v_3, t) = 0 + (-12) + 4 + (-7) + 15 = 0.$

The value of a flow is given by

$$|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$$

This is the total flow leaving s = The total flow arriving in t.
Example
$|f| = f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t) = 11 + 8 + 0 + 0 + 0 = 19$
$|f| = f(s, t) + f(v_1, t) + f(v_2, t) + f(v_3, t) + f(v_4, t) = 0 + 0 + 0 + 15 + 4 = 19$

We assume that there is only flow in one direction at a time. If we have several sources and several targets and we want to maximize the total flow from all sources to all targets, we reduce to max-flow by creating a supersource and a supersink.

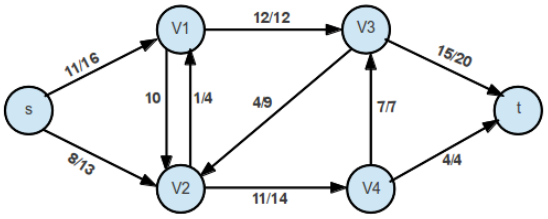

Figure 4.3:

4.1.2 Min-Cut Problem

Dual of maximum flow problem known as minimum cut problem. A cut may be defined as a set of directed arcs such that if we remove the arcs of the cut, no directed path from the source to the sink will be left. A cut (S,T) of a flow network is a partition of V into S and T = V - S, such that $s \in S$ and $t \in T$.

The net flow through a cut (S,T) -

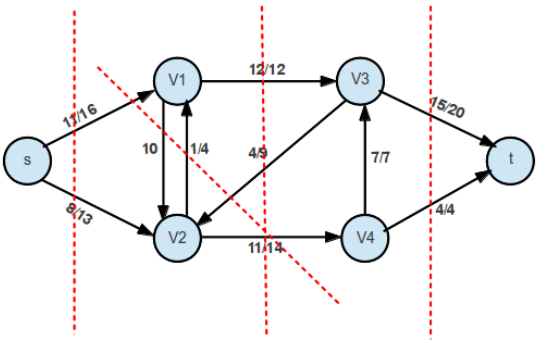

Figure 4.4:

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v)$$

f(S,T) = 12 - 4 + 11 = 19

The capacity of a cut (S,T) -

$$c(S,T) = \sum_{u \in S, v \in T} c(u,v)$$

c(S,T) = 12 + 0 + 14 = 26
The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

Generalizing the Concept of Cut

S is called the S-side of the cut and V - S is the T side of the cut.

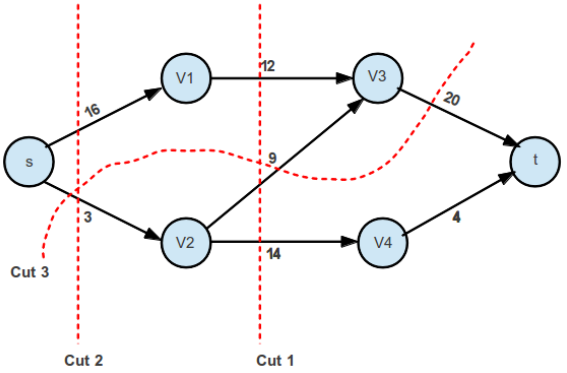

Figure 4.5:

Cut-1 = $\{V_1 \rightarrow V_3, V_2 \rightarrow V_3, V_2 \rightarrow V_4\}$ with S-side = $\{s, V_1, V_2\}$ and T-side = $\{V_3, V_4, t\}$.
Cut-2 = $\{s \rightarrow V_1, s \rightarrow V_2\}$ with S-side = $\{s\}$ and T-side = $\{V_1, V_2, V_3, V_4, t\}$.
The arcs going from T-side to S-side don't count in the cut.
Cap(Cut-1) = 12 + 9 + 14 = 35
Cap(Cut-2) = 16 + 3 = 19
The cut defined by S-side = $\{s, V_1, V_3\}$ doesn't include arc $V_2 \rightarrow V_3$
Cut-3 = $\{s \rightarrow V_2, V_3 \rightarrow t\}$ with Cap(Cut-3) = 3 + 20 = 23

To understand the concept of an st-flow, we define some notations :-

$$\sigma^+(u) = \{e \in E : e = (u, v) for\ some\ v\}\text{: arcs leaving u.}$$
$$\sigma^-(x) = \{e \in E : e = (v, u) for\ some\ v\}\text{: arcs entering u.}$$

where 'e' represents an edge from u to v. A function $f : E \rightarrow \mathbb{R}^+$ is called an st-flow if the following holds :

$$\sum_{\sigma^+(u)} f_e - \sum_{\sigma^-(u)} f_e = 0$$

for all $u \in V - \{s, t\}$.

$$0 \leq f_e \leq c_e \qquad \forall\ e \in E.$$

In maximum flow problem we assume that no arc enters in 's' and no arcs leaves from 't'. We define the value **val f** of an st-flow f as the amount of flow leaving s, i.e.,

$$\mathbf{val\ f} = \sum_{\sigma^+(\mathbf{s})} \mathbf{f_e}$$

The maximum flow problem is to find "an st-flow f with maximum value val(f)".

The primal for the network flow, called the max-flow, is given in an LP as follows :-

$$Max \sum_{(s \to v) \in E} f_{s \to v}$$

subject to

$$\begin{aligned}
f_{u \to v} &\leq c(u \to v) && \forall (u \to v) \in E \\
\sum_{(u \to v) \in E} f_{u \to v} - \sum_{(v \to w) \in E} f_{v \to w} &\leq 0 && \forall v \in V - \{s, t\} \\
- \sum_{(u \to v) \in E} f_{u \to v} + \sum_{(v \to w) \in E} f_{v \to w} &\leq 0 && \forall v \in V - \{s, t\} \\
f_{u \to v} &\geq 0 && \forall (u \to v) \in E
\end{aligned}$$

In the primal, there are three constraint equations. The third constraint equation is same as the second constraint equation, just interchanging the minus sign between the flows. The first constraint equation is for the edges, which is called a capacity constraint and the next two constraint equations are based on the conservation of the flow for the vertices.

To construct the dual of this primal max-flow problem, we generate as many variables as there are constraint equations in the primal problem, that means we define a dual variable for each of the constraint equation or inequality.

So, to construct the dual, we multiply each inequality by a defined dual variable.[14]

$$Max \sum_{(s \to v) \in E} f_{s \to v}$$

subject to

$$\begin{aligned}
f_{u \to v} &\leq c(u \to v) && * y_{u \to v} && \forall (u \to v) \in E \\
\sum_{(u \to v) \in E} f_{u \to v} - \sum_{(v \to w) \in E} f_{v \to w} &\leq 0 && * y_v && \forall v \in V - \{s, t\} \\
- \sum_{(u \to v) \in E} f_{u \to v} + \sum_{(v \to w) \in E} f_{v \to w} &\leq 0 && * y_v^{'} && \forall v \in V - \{s, t\} \\
f_{u \to v} &\geq 0 && && \forall (u \to v) \in E
\end{aligned}$$

Doing the duality transformation carefully, we get the following :-[14]

$$Min \sum_{(u \to v) \in E} c(u \to v) y_{u \to v}$$

subject to

$$\begin{aligned}
y_{s \to v} + y_v - y_v^{'} &\geq 1 && \forall (s \to v) \in E \\
y_{u \to v} + y_v - y_v^{'} - y_u + y_u^{'} &\geq 0 && \forall (u \to v) \in E(V - \{s, t\}) \\
y_{v \to t} - y_v + y_v^{'} &\geq 0 && \forall (v \to t) \in E \\
y_{u \to v} &\geq 0 && \forall (u \to v) \in E \\
y_v &\geq 0 && \forall v \in V \\
y_v^{'} &\geq 0 && \forall v \in V
\end{aligned}$$

In the second constraint equation of the dual problem, we do not include the source and sink node. Since the total flow through a vertex other than s and t, is zero, so we need to introduce more variables (y_u and $y_u^{'}$)to convert the equality($= 0$) into the inequality (≥ 0).

When we convert the primal LP into the standard form of the dual, then we get this expression of the dual. We know that in the standard form we write the primal as

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ \text{and} \quad & x \geq 0 \end{aligned}$$

and the corresponding symmetric dual can be expressed as -

$$\begin{aligned} \text{minimize} \quad & b^T y \\ \text{subject to} \quad & y^T A \geq c \\ \text{and} \quad & y \geq 0 \end{aligned}$$

So, for the dual, matrix A would be an $(E \cup V_1 \cup V_1{}' \times E)$ matrix, where V_1, denotes the vertices according to the second constraint equation and $V_1{}'$ are the vertices according to the third constraint equation of the primal. We know that $V_1 = V_1{}'$, we are just denoting them like this only for our convenience.
Matrix A contains three submatrices - the first submatrix is an $E \times E$ matrix for the edges of the network which contains 1 on its diagonal positions and 0 at the other entries.
The second submatrix is a $(V_1 \times E)$ matrix for the vertices which satisfies the second constraint equation of the primal. The elements for this submatrix is 0,1 and -1. For a vertex $v \in V$, if the flow comes inside v, we put 1, if the flow leaves out from v, we put -1 and 0 otherwise.
The third submatrix is just the opposite of the second submatrix. According to the third constraint equation, if the flow comes inside v, we put -1, if the flow is leaving from v, we put 1 and 0 otherwise.
The matrix y would be an $(E \cup V_1 \cup V_1{}' \times 1)$ matrix, which also contains three submatrices, one for the edges and next two for the vertices V_1 and $V_1{}'$.
matrix c would be a $(1 \times E)$ matrix, which contains 0 and 1 only.

Now, consider a column of the matrix A, for an edge $(u \to v)$, it contains 1 in the first submatrix at the diagonal position for that particular edge and second submatrix contains -1, at the position of vertex u, 1 at the position of vertex v and 0 for the other entries, third submatrix of A contains 1, at the position of vertex u, -1 at the position of vertex v and 0 for the other entries. So when we multiply the y^T matrix to the matrix A, we get the expression shown in the second constraint equation of the dual.

Now, we simplify this by introducing the variables $d_v = y_v - y_v^{'}$ for each $v \in V - \{s, t\}$. We get the following modified dual LP :

$$Min \sum_{(u \to v) \in E} c(u \to v) y_{u \to v}$$

subject to

$$\begin{aligned} y_{s \to v} + d_v &\geq 1 && \forall (s \to v) \in E \\ y_{u \to v} + d_v - d_u &\geq 0 && \forall (u \to v) \in E(V - \{s, t\}) \\ y_{v \to t} - d_v &\geq 0 && \forall (v \to t) \in E \\ y_{u \to v} &\geq 0 && \forall (u \to v) \in E \end{aligned}$$

Adding the variables for t and s, their values as follows, $d_t = 0$, $d_s = 1$, we get the more homogeneous and symmetric dual LP, so we don't have to treat s and t separately.

$$Min \sum_{(u \to v) \in E} c(u \to v) y_{u \to v}$$

subject to

$$\begin{aligned} y_{s \to v} + d_v - d_s &\geq 0 && \forall (s \to v) \in E \\ y_{u \to v} + d_v - d_u &\geq 0 && \forall (u \to v) \in E(V - \{s, t\}) \\ y_{v \to t} + d_t - d_v &\geq 0 && \forall (v \to t) \in E \\ y_{u \to v} &\geq 0 && \forall (u \to v) \in E \end{aligned}$$

$d_s = 1$, $d_t = 0$

Which simplifies to the following LP:

$$Min \sum_{(u \to v) \in E} c(u \to v) y_{u \to v}$$

subject to

$$\begin{aligned} d_u - d_v &\leq y_{u \to v} && \forall (u \to v) \in E \\ y_{u \to v} &\geq 0 && \forall (u \to v) \in E \end{aligned}$$

$d_s = 1$, $d_t = 0$

Thus we have obtained the dual of the max-flow problem. Now our problem is reduced to check whether the optimal solution to this dual LP represents a min-cut. Let us first start with the other direction. Given a cut (S,T), with $s \in S$ and $t \in T$, we get a feasible solution of this dual LP by setting -

$d_u = 1$ $\quad \forall u \in S$

$d_u = 0$ $\quad \forall u \in T$

$y_{u \to v} = 1$ $\quad \forall (u \to v) \in (S, T)$

$y_{u \to v} = 0$ $\quad \forall (u \to v) \in E - \{S, T\}$

As for the other direction, we consider the optimal solution for the dual LP and let its target function value be

$$\alpha^* = \sum_{(u \to v) \in E} c(u \to v) y^*_{u \to v}$$

The (*) represents the values of the variables in the optimal LP solution. Using the optimal solution, we generate cuts for each $z \in [0, 1]$ as follows -
for any $z \in [0, 1]$, we define $S_z = \{u | d^*_u \geq z\}$ and $T_z = \{u | d^*_u < z\}$. This is a valid cut as $s \in S$ (as $d^*_s = 1$) and $t \in T$ (as $d^*_t = 0$). An edge $(u \to v)$ is in the cut only if $d^*_u > d^*_v$.

Let us consider the following experiment :- we pick $z \in [0, 1]$ at random and consider the cut (S_z, T_z). Notice further, that actually we have defined a probability measure on the set of cuts where, P(C) = length of the interval of $z \in [0, 1]$, such that $(S_z, T_z) = $ C. Now, for this experiment the probability of $u \in S$ and $v \in T$ is exactly $d^*_u - d^*_v$. In other words, we can say that P(C) is the probability that z falls inside the interval $[d^*_v, d^*_u]$. So, for this experiment, the edge $(u \to v)$ is in the cut with the probability $d^*_u - d^*_v$ (only if $d^*_u > d^*_v$) which is bounded by $y^*_{u \to v}$ (by the inequality $d_u - d_v \leq y_{u \to v}$ in LP).

Now we define an indicator variable $X_{u \to v}$ which is one if, the edge is in the generated cut and zero otherwise[14]. Observe that

1. X = $\sum X_{u \to v} c(u \to v)$ is the random variable that to a cut associates its cost.
2. $E[X_{u \to v}] = P[X_{u \to v} = 1] \leq y^*_{u \to v}$

$\Rightarrow$

$$\begin{aligned} E[\sum X_{u\to v} c(u \to v)] &= \sum_{(u\to v)\in E} c(u \to v) E[X_{u\to v}] \\ &\leq \sum_{(u\to v)\in E} c(u \to v) y^*_{u\to v} \\ &= \alpha^* \end{aligned}$$

Since expectation of a random variable (Z) is the weighted average of the values that the random variable can have. So there must be a value z that is assigned to the random variable (Z) such that $z \leq \alpha^*$. But earlier we said that α^* is the optimum solution for the dual, means that there is no cut which has the capacity less than the α^*. Hence by both of the arguments we can say that α^* is the minimum cut. Hence by the strong duality max-flow is equal to the optimal solution of the dual (i.e. α^*).

4.2 Strong Duality

Theorem 4.2.1. *If the primal LP problem (P) has an optimal solution x^* and the dual (D) has an optimal solution y^* then, $c^T x^* = b^T y^*$.*

The proof of this theorem is somewhat tedious and not very insightful, so we are not going to prove this theorem.

We are now ready to deal with the max-flow min-cut theorem, that is, the optimal value of the max-flow is equal, by using the strong duality, to the optimal value of the dual.

Lemma 4.2.2. *Given a network G(V,E), then for any flow f and cut C on the network, val f $\leq$cap C.[15]*

Proof. Let C = (S,T). As S contains sources and intermediates, so clearly

$$val f = f_{out}(X) - f_{in}(X) = f_{out}(S) - f_{in}(S)$$

since the contribution of the intermediates to the flow value is zero by conservation law. For an edge with both end-points in S, its flow is counted in both $f_{out}(S)$ and $f_{in}(S)$ and thus doesn't affect the flow value. Therefore, the only edges which positively affect the *val f* are the ones which originating in S and terminating in T, which are precisely the flows over the cut C. Hence, we conclude that

$$val f \leq \sum_{x \in S, y \in T} f(x,y) \leq \sum_{x \in S, y \in T} c(x,y) = capC$$

□

Corollary 4.2.3. *Given a network, let f^* be the maximum flow and C^* be the minimum cut on the network. Then val $f^* \leq$ cap C^*.[15]*

The corollary is a obvious consequence of the above lemma, so we are not giving the proof of this corollary.

4.3 Max-flow Min-cut Theorem

Theorem 4.3.1. *For any directed graph with arc capacity function and distinct vertices s and t the value of a maximum st-flow equals the minimum (S,T)-cut capacity.*

Proof. Let f^* be the maximum flow and C^* be the minimum cut on a given network. By the lemma (4.2.2) there exists some flow f and cut C such that cap C = val f $\leq$ cap C^*, but we know that every cut has more capacity than the minimum cut, so in fact val f = cap C^*. And by corollary (4.2.3) we can say that val $f^* \leq$ cap C^* = val f, but no flow can have the value greater than the maximum flow, so val f^* = cap C^*.[15] □

Bibliography

[1] V. Strassen. *Gaussian Elimination is Not Optimal.* Numerische Mathe-matik 13, 354-356, 1969.

[2] Predrag S. Stanimirovi and Marko D. Petkovi. *Generalized matrix inversion is not harder than matrix multiplication.* University of Niˇ, Department of Mathematics, Faculty of Science, Visegradska 33, 18000 Niˇ, Serbia.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975. Second printing, Addison-Wesley Series in Computer Science and Information Processing.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms.* MIT Press, Cambridge, MA, third edition, 2009.

[5] DONALD GOLDFARB Robert G. BLAND and MICHAEL J. TODD. *Feature Article, The Ellipsoid Method: A Survey.* Cornell University, Ithaca, New York.

[6] Lecturer : Michel X. Goemans. *Combinatorial Optimization.* Massachusetts Institute of Technology, Handout 19, May 4th, 2009.

[7] Lecturer : Sanjeev Arora. *The Ellipsoid Algorithm for Linear Programming.* Princeton University, COS 521, Fall 2005.

[8] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* University of California - Berkeley, Princeton University.

[9] Adejo B. O. and Okutachi A. M. *On increasing the rate of convergence of the ellipsoid algorithm for linear programming.* Department of Mathematical Sciences, Kogi State University, Anyigba, Nigeria.

[10] Vasek Chvatal. *Linear Programming.* W. H. Freeman and Company, McGill University, 1983.

[11] Lecturer : Robert J. Vanderbei. *Linear Programming: Chapter 5, Duality.* Operations Research and Financial Engineering, Princeton University, October 17, 2007.

[12] Lecture : Adrian Vetta. *The Strong Duality Theorem.*

[13] Lecturer : Jeff Erickson. *Maximum Flows and Minimum Cuts.* Fall 2013.

[14] Sariel Har-Peled. *CS 573 Algorithms.* sarielhp.org/teach/05/b, May 29, 2013.

[15] Lecturer : Shaun Joseph. *The Max-Flow Min-Cut Theorem.* December 6, 2007.