CS 435 : Linear Optimization — Fall 2006

Lecture 1: Introduction

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe:*Anup Kumar*
Indian Institute of Technology, Bombay

1 History of Linear Optimization

This subject first came up in manufacturing and industrial engineering in 1940s where they need to derive maximum profit with less expenditure.

2 Definition of Linear Programming

We illustrate the problem of Linear Programming through the following example.

Example 1 A factory needs to produce different goods from different kinds of raw materials available. Each good yields a certain amount of profit per unit. Each good in the process of its manufacture uses a certain proportion of each available raw material. The amount of each raw material available is limited.

The problem is how much quantity of each good to produce using these raw materials in order to maximize the total profit on the goods.

The problem is mathematically formulated as follows :

- Raw Materials Available: M_1, M_2, M_3, $\cdots$, M_m
- Quantity of each material available: b_1, b_2, b_3, $\cdots$, b_m
- Goods to be produced: G_1, G_2, G_3, $\cdots$, G_n
- Profit on each good: p_1, p_2, p_3, $\cdots$, p_n
- A_{ij}: Proportion of material M_i required to produce one unit quantity of good G_j
- x_j: Quantity of each good G_j produced. $x_j \geq 0$.
- Total Profit: The sum of profit on all goods $= \sum_j x_j\, p_j$

Optimization Problem: How much of each good to be produced, x_j, to maximize the total profit under the given constraints?

The name *Linear Optimization* comes from the fact that the quantity which is to be optimized is a linear function of the unknown quantity x_j and the constraints on x_j are linear inequalities. The above problem is an instance of maximizing the value.

There can be problem instances where we have to minimize some quantity and find out the unknowns which minimize that quantity. For example, the above problem could have been formulated as minimum raw material requirement under certain constraints such as given profit, etc.

Problem Statement:

- Inputs: b_i, p_j, the matrix A
- Outputs: x_j, which denotes the quantity of each G_j to be produced
- To Maximize: $\sum_j x_j p_j$. If profit p_j is denoted by an $n \times 1$ column matrix p, then we have to maximize the value $p^t x$ where x denotes the column matrix of x_js.
- Constraints:
  - for all material i, $\sum_j a_{ij} x_j \leq b_i$ or $Ax \leq b$
  - $x_j \geq 0$

3 Solution Idea

We need to understand what $\{x : Ax \leq b\}$ looks like. Suppose there were only two variables x_1 and x_2. We begin by considering the object $\{x : Ax = b\}$. The linear equations $\sum_j a_{ij} x_j = b_i$ represents a set of straight lines in two dimensional plane with x_1 and x_2 as co-ordinates.

The solution to these equations can be a single point in the plane or a line or empty. This can be extended further when there are 3 or more variables. Our goal is to have a geometric interpretation and an algebraic description for these sets.

In the next class we shall see the solutions to the linear equations and their geometric description.

CS 435 : Linear Optimization | Fall 2006

Lecture 2: Standard LP formulation, Linear algebra : solution to simultaneous linear equations $Ax = b$, Gaussian elimination

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe:*Dinesh Gadge*
Indian Institute of Technology, Bombay

1 Formulation

A linear optimization problem can be formulated as

$$\begin{aligned} \max\ & c^T x \\ & Ax \leq b \end{aligned} \tag{1}$$

where, A is an $m \times n$ matrix, c a $n \times 1$ vector, b a $m \times 1$ vector and x a $n \times 1$ vector. We are given as **Input:** c, A, b and desire as **Output** : x. Among all x that satisfies $Ax \leq b$ find one which maximises $c^T x$. The set $Ax \leq b$ is a set of points which have typical properties. Our first goal is to understand them and be able to describe some of these.

2 Understanding the set of all solutions to $Ax = b$

Before considering the set of inequalities, $Ax \leq b$, we consider the set of equalities $Ax = b$. This is short hand notation for the following set of equalities given below.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

We can solve such a system of equations using Gaussian Elimination. Here is an example.

Example 1

$$2x + 7y = 13 \tag{I}$$
$$x + 3y = 4 \tag{II}$$

Replacing II by -$\frac{1}{2}$·I + II gives

$$\begin{aligned} -\frac{7}{2}y + 3y &= -\frac{13}{2} + 4 \\ y &= 5 \end{aligned} \tag{2}$$

3 Gaussian Elimination

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

Assume that $a_{11} \neq 0$. If this is not so we exchange the first row with some other row which has a non-zero first co-ordinate. Then for each of the equations except the first, multiply the first equation by a suitable constant and subtract from the respective equations to get rid of x_1 in the other equations.

We then do this with the other equations ignoring the first variable.

There are 2 operations used in Gaussian Elimination.

1. Exchange two rows.
2. Replace row_j with $\alpha \cdot row_i + row_j$, where α is some constant.

It is possible that using these two operations the co-efficients of some x_i [for example x_2] in all rows except one are zeroed out.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ 0 + 0 + a_{23}x_3 + \cdots + a_{2n}x_n &= b'_2 \\ 0 + 0 + a_{33}x_3 + \cdots + a_{3n}x_n &= b'_3 \\ &\vdots \\ 0 + 0 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b'_m \end{aligned}$$

In such a case, assume that, some $a_{ij} \neq 0$ [for example $a_{23} \neq 0$], then we repeat the process.

Finally, we get something like,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ 0 + 0 + \cdots + a_{2i_1}x_{i_1} + \cdots + a_{2n}x_n &= b''_2 \\ &\vdots \\ 0 + 0 + \cdots + 0 + a_{ki_{k-1}}x_{i_{k-1}} + \cdots + a_{kn}x_n &= b''_k \\ &\vdots \\ 0 + 0 + \cdots + 0 + 0 + 0 + \cdots + 0 &= b''_l \\ &\vdots \\ 0 + 0 + \cdots + 0 + 0 + 0 + \cdots + 0 &= b''_m \end{aligned}$$

Observations:

1. All zero rows occur after the non-zero rows.

2. If, from the top, the first t rows are nonzero, and the first non-zero entry of the ith row $i = 1, \ldots, t$, is at the k_ith column, then $k_i > k_{i-1}$, $i = 2, \ldots, t$. That is, the first non-zero entries in the rows appear later and later from top to bottom.

3.1 Existence of a solution for $Ax = b$

$Ax = b$ does not have a solution if in a particular row i, all coefficients $a_{ij} = 0$, but $b_i \neq 0$. This is both necessary and sufficient condition for non-existence of a solution for $Ax = b$ as we shall see later.

Once we have the matrix in this form, it is easy to get solutions to the set of equations, if one exists. Except for $x_1, x_{i_1}, \ldots, x_{i_{k-1}}$ set any values to the other variables. Now solve for the variables $x_1, x_{i_1}, \ldots, x_{i_{k-1}}$ in the reverse order.

If it has n non-zero rows then it has only one solution. Otherwise $\{x : Ax = b\}$, in general, has many solutions.

3.2 Why is this procedure correct?

The procedure being correct means that the values of the variables obtained by the procedure indeed satisfy the original set of equations. So to prove the correctness of the procedure, we have to prove that the solution set does not change on applying the operations of Gaussian Elimination. Clearly exchanging two rows does not change the solution set.

THEOREM 1 *Given a set of equations, suppose eq_j is replaced by*

$$\alpha \cdot eq_i + eq_j$$

then the solutions set does not change.

PROOF: Let eq_i be $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i$ and eq_j be $a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jn}x_n = b_j$. Consider the old and the new set of equations. Note that they differ only in the jth equation. This proof consists of 2 parts.

Part 1: *If the set of solutions satisfies original set of equations then it satisfies the new set.* Let $< x'_1, x'_2, \ldots, x'_n >$ be the solution to the original set. Then $< x'_1, x'_2, \ldots, x'_n >$ specifically satisfies eq_i and eq_j. So,

$$\begin{aligned} a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n &= b_i \\ a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n &= b_j \end{aligned}$$

that is,

$$\begin{aligned} a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n - b_i &= 0 \\ a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n - b_j &= 0 \end{aligned}$$

So,

$$\begin{aligned} & \alpha(a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n - b_i) + (a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n - b_j) \\ &= \alpha \cdot 0 + 0 \\ &= 0 \end{aligned}$$

Therefore,

$$\alpha(a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n) + (a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n) = \alpha \cdot b_i + b_j$$

Hence $< x'_1, x'_2, \ldots, x'_n >$ satisfies $\alpha \cdot eq_i + eq_j$. Since rest of the equations in both the sets are identical, $< x'_1, x'_2, \ldots, x'_n >$ satisfies the new set.

Part 2: *If the set of solutions satisfies new set of equations then it satisfies the original set.* Let $< x'_1, x'_2, \ldots, x_n >$ be the solution to the new set. Then $< x'_1, x'_2, \ldots, x'_n >$ specifically satisfies $\alpha \cdot eq_i + eq_j$ and eq_i. So,

$$\begin{aligned} \alpha(a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n) + (a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n) &= \alpha \cdot b_i + b_j \\ a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n &= b_i \end{aligned}$$

that is,

$$\begin{aligned} \alpha(a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n - b_i) + (a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n - b_j) &= 0 \\ a_{i1}x'_1 + a_{i2}x'_2 + \cdots + a_{in}x'_n - b_i &= 0 \end{aligned}$$

So,

$$\begin{aligned} \alpha \cdot 0 + (a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n - b_j) &= 0 \\ \Rightarrow a_{j1}x'_1 + a_{j2}x'_2 + \cdots + a_{jn}x'_n &= b_j \end{aligned}$$

Hence $< x'_1, x'_2, \ldots, x'_n >$ satisfies eq_j. Since rest of the equations in both the sets are identical, $< x'_1, x'_2, \ldots, x'_n >$ satisfies the old set. □

4 Understanding $Ax = b$ geometrically

Another way of looking at $Ax = b$ is through geometry. The operation of adding a constant times another equation to an equation rotates one of the hyperplanes in $\mathbb{R}^n$. Fig. 1 illustrates this as the case of rotation of lines (1-dimensional hyperplane) in a x-y plane (i.e. $\mathbb{R}^2$). Here the solid lines are from example 1 and one of the dotted lines is obtained by rotating the line corresponding to eq. II to the one corresponding to eq. 2. The other dotted line can be obtained by similarly manipulating eq. I

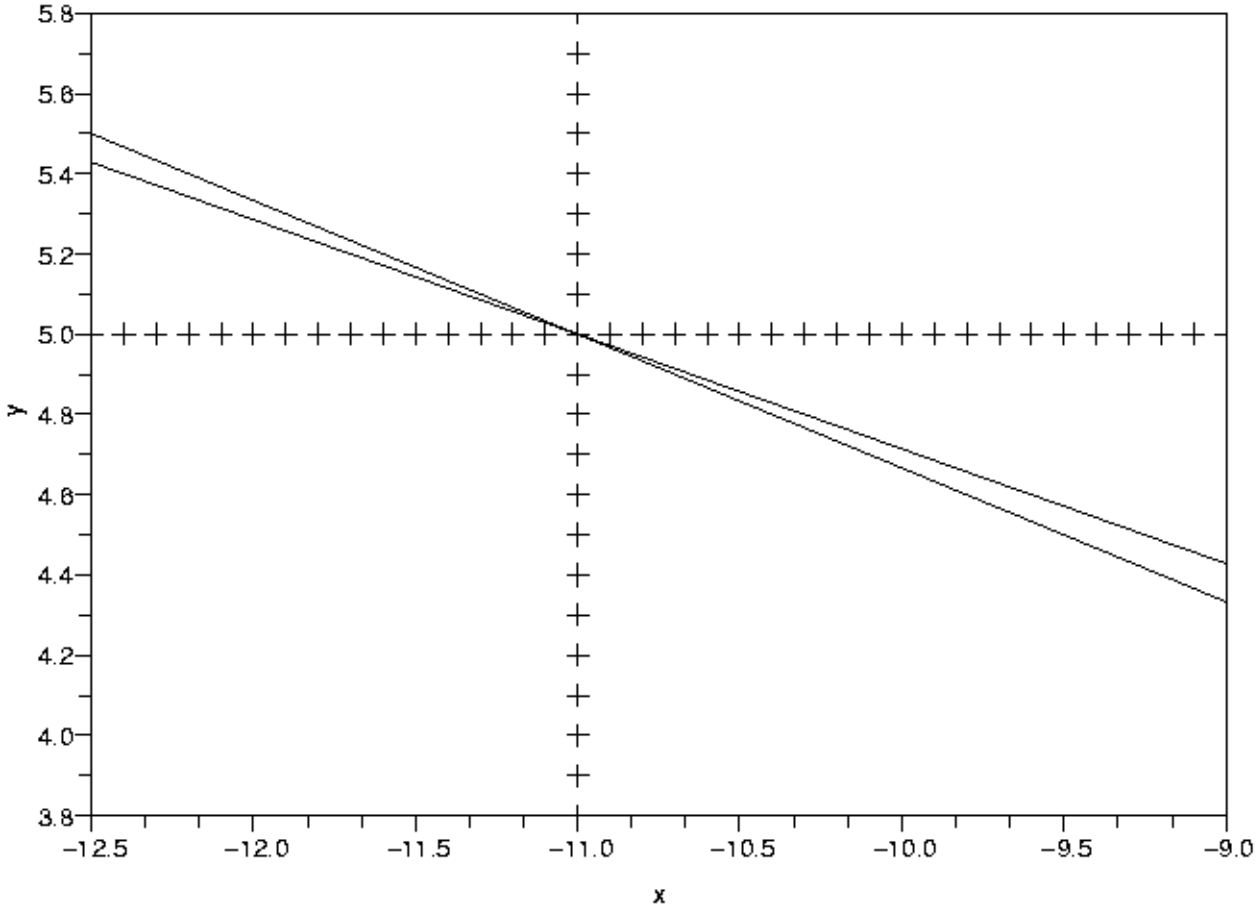


Figure 1: Geometrical way of looking at $Ax = b$

CS 435 : Linear Optimization — Fall 2006

Lecture 3: Linear algebra I

Lecturer: *Sundar Vishwanathan* — Scribe: *Luv Kumar*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Vector Space

A *vector space* is defined as a set of vectors $\mathbf{V}$ and the real numbers $\mathbf{R}$ (called *scalars*) with the following operations defined:

- **Vector Addition:** $\mathbf{V} \times \mathbf{V} \rightarrow \mathbf{V}$, represented as $\mathbf{u} + \mathbf{v}$, where $\mathbf{u}, \mathbf{v} \in \mathbf{V}$.
- **Scalar Multiplication:** $\mathbf{R} \times \mathbf{V} \rightarrow \mathbf{V}$, represented as $a.\mathbf{u}$, where $a \in \mathbf{R}$ and $\mathbf{u} \in \mathbf{V}$.

Following are the properties of a vector space.

- **Abelian Group laws:**
  1. **Associativity:** $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
  2. **Identity:** $\exists$ a zero vector $\overline{\mathbf{0}}$ which is the group identity element, i.e. $\overline{\mathbf{0}} + \mathbf{u} = \mathbf{u}$
  3. **Inverse:** $\forall \mathbf{u} \in \mathbf{V}$, there exists the additive inverse $-\mathbf{u}$ s.t. $\mathbf{u} + (-\mathbf{u}) = \overline{\mathbf{0}}$
  4. **Commutativity:** $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
- **Scalar multiplication laws:**
  1. **Multiplication by** 0**:** $0.\mathbf{u} = \overline{\mathbf{0}}$
  2. **Multiplication by** -1**:** $(-1).\mathbf{u} = -\mathbf{u}$
  3. **Identity multiplication:** $1.\mathbf{u} = \mathbf{u}$
  4. **Distributivity of vector sum:** $a.(\mathbf{u} + \mathbf{v}) = a.\mathbf{u} + a.\mathbf{v}$, where $a \in \mathbf{R}$ and $\mathbf{u}, \mathbf{v} \in \mathbf{V}$
  5. **Distributivity of scalar sum:** $(a + b).\mathbf{u} = a.\mathbf{u} + b.\mathbf{u}$
  6. **Associativity of scalar multiplication:** $a.(b.\mathbf{u}) = (ab).\mathbf{u}$

2 Subspace

$\mathbf{U} \subseteq \mathbf{V}$ is a *subspace* of $\mathbf{V}$ if $\mathbf{U}$ itself is a vector space, i.e. for all $\mathbf{u}_1, \mathbf{u}_2 \in \mathbf{U}$ and $\alpha \in \mathbf{R}$, $\mathbf{u}_1 + \mathbf{u}_2 \in \mathbf{U}$ and $\alpha.\mathbf{u}_1 \in \mathbf{U}$. For example, if $\mathbf{u} \in \mathbf{V}$, then $\mathbf{U} = \{\alpha.\mathbf{u}\ \forall \alpha \in \mathbf{R}\}$ is a subspace. For $\alpha = -1$, $-\mathbf{u}_1 \in \mathbf{U}$ whenever $\mathbf{u}_1 \in \mathbf{U}$. Hence, $-\mathbf{u}_1 + \mathbf{u}_1 \in \mathbf{U}$. Therefore, $\overline{\mathbf{0}}$ is always a member of any subspace.

Example 1 In a 2-dimensional space, any line passing thru the origin is a subspace. If there is any vector in $\mathbf{U}$ that does not lie on this line, then $\mathbf{U}$ has to be the entire plane.

3 Linear Dependence, Independence and basis

DEFINITION 1 *Vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ *are linearly dependent if there exist* $\alpha_1, \ldots, \alpha_n \in \mathbf{R}$*, not all zero, such that*

$$\sum_{i=1}^{n} \alpha_i . \boldsymbol{v}_i = \overline{\boldsymbol{0}}$$

DEFINITION 2 *Vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ *are linearly independent if they are not linearly dependent, i.e. for* $\alpha_1, \ldots, \alpha_n \in \mathbf{R}$

$$\sum_{i=1}^{n} \alpha_i . \boldsymbol{v}_i = \overline{\boldsymbol{0}} \quad \Rightarrow \quad \alpha_i = 0, \ \forall i$$

Basis of a vector space is defined in terms of linear dependence as follows:

DEFINITION 3 *Vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ *form the basis of a vector space* $\mathbf{V}$ *iff:*

1. *They are linearly independent.*
2. *Every other vector* $\boldsymbol{w}$ *which belongs to* $\mathbf{V}$ *can be written as*

$$\boldsymbol{w} = \sum_{i=1}^{n} \beta_i . \boldsymbol{v}_i$$

Alternatively, $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ *form the basis of vector space* $\mathbf{V}$ *if on adding other vector* $\boldsymbol{w} \in \mathbf{V}$ *to this set, the set becomes linearly dependent.*

There can be multiple basis for the same vector space, but all of them will have the same size. Therefore, if $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is a basis, and $\mathbf{u}_1, \ldots, \mathbf{u}_m$ is also a basis, then $m = n$. The number of vectors in the basis is called the *dimension* of the vector space.

CS 435 : Linear Optimization | Fall 2006

Lecture 4: Linear algebra : Basis, Dimension

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe:*Bhaskara Aditya*
Indian Institute of Technology, Bombay

1 Basis (contd.)

We first give the proof for the result stated in the previous lecture – if two sets of vectors S and T form a basis for a vector space V, their sizes are the same (i.e., $|S| = |T|$). We recall what is meant by saying a set X of vectors is a basis for a space V:

1. The vectors in X are linearly independent (i.e., there's no nontrivial combination of them which is zero).
2. Any vector in V can be expressed as a linear combination of vectors in X.

We will prove the result by contradiction, by assuming $|S|$ is strictly smaller than $|T|$, and then obtaining a set $S' \subset T$ which is also a basis, with the property $|S'| = |S|$. This is clearly a contradiction, because $|S'| = |S| < |T|$ and $S' \subset T$, so the vectors in $T \setminus S'$ can be expressed in terms of those in S', contradicting the fact that T is a basis (in particular, (1) above).

We start with a lemma.

Lemma 1 *Suppose $S = \{v_1, \ldots, v_n\}$ and let $x = \sum_{i=1}^{n} \alpha_i v_i$ with $\alpha_1 \neq 0$. Let $S' = \{x, v_2, \ldots, v_n\}$. Then, $span(S) = span(S')$.*

Proof: The proof in both directions is easy. The crucial thing is to observe that since $\alpha_1 \neq 0$, we can write v_1 as a combination of $x, v_2, \ldots, v_n$. Thus, any vector in $span(S)$ is also in $span(S')$. For the other direction, since x is a combination of v_i's, every vector in $span(S')$ is also in $span(S)$. This proves the equality. □

We now proceed with the proof of the theorem.

Theorem 1 *Suppose $S = \{u_1, u_2, \ldots, u_m\}$ and $T = \{v_1, v_2, \ldots, v_n\}$ be two sets of vectors such that each is a basis for the vector space V. Then $|S| = |T|$.*

Proof: The proof will follow the outline above. Suppose without loss of generality, that $m < n$. Starting with the set $S_0 = S$, we do the following: replace one of the vectors of S_0 by a vector from T such that the new set, say S_1, still spans the entire V. Also, if v_i was the vector in T which was added to S_0, we set $T_1 = T \setminus \{v_i\}$.

We now repeat this step m times. Further, we ensure that at each step, the element removed from S_i is one of the u_j's (not the v_j's that have been added). We now show that such an operation is indeed always possible. More precisely, assume that we have two sets S_i and T_i (with $i < m$). We show that it is always possible to obtain a set S_{i+1} such that:

1. S_{i+1} is obtained from S_i by removing one of the u_j's from S_i and adding one of the v_j's (which is from T_i) (call it x) to it.

2. The span of S_{i+1} is the same as the span of S_i.

Further, we set $T_{i+1} = T_i \setminus \{x\}$. We prove this as follows.

Note that since initially we have that $S_0 = S$ is a basis, and at every step so far the span was preserved, we can assume S_i spans the entire V. We may also assume that we have re-numbered the u_i's and v_i's such that $S_i = \{u_{i+1}, \ldots, u_m, v_1, \ldots, v_i\}$ and $T_i = \{v_{i+1}, \ldots, v_n\}$.[1] Since S_i spans the whole of V, we have

$$v_{i+1} = \sum_{j=i+1}^{m} \alpha_j u_j + \sum_{j=1}^{i} \beta_j v_j \tag{1}$$

Now, at least one of α_j's must be non-zero, else we would have a non-trivial combination of v_i's as zero, which is not possible since T is a basis. So assume without loss of generality α_{i+1} is non-zero. Then, by the lemma above, replacing u_{i+1} by v_{i+1}, we still get a basis. This is the required S_{i+1}.

Thus, if we repeat this process m times, the resulting set, S_m will have just v_j's and they span the whole of V. This is a contradiction, as we have seen in the outline above. □

So we have proved that given a vector space V, any basis for it will have the same size (assuming there exists a finite basis). Thus, this number is a property of V alone, and it is called the **dimension** of V. The above result proves that it is well-defined.

2 Solutions to $Ax = \mathbf{0}$

We now want to look at the set of all solutions to the system $Ax = \mathbf{0}$, where A is an $m \times n$ matrix and x is a vector in $\mathbb{R}^n$. From now on, denote $\mathcal{S} = \{x \ : \ Ax = \mathbf{0}\}$. Note that $\mathcal{S}$ is a subspace of $\mathbb{R}^n$. This is because if $x \in \mathcal{S}$ then $A(\alpha x) = \alpha(Ax) = \mathbf{0}$, so $\alpha x \in \mathcal{S}$. Similarly, we can verify that $x_1, x_2 \in \mathcal{S} \Rightarrow x_1 + x_2 \in \mathcal{S}$.

Having proved that $\mathcal{S}$ is a subspace, we ask the natural question – what is it's dimension? We give an answer to this problem in terms of the matrix A. We will, in fact, prove the following in the coming lectures.

THEOREM 2 1 *Suppose k is the number of linearly independent columns in the matrix A. Then, $dim(\mathcal{S}) = n - k$.*

We will also prove a 'row version' of this theorem.

THEOREM 3 1 *Suppose k is the number of linearly independent rows in the matrix A. Then, $dim(\mathcal{S}) = n - k$.*

This gives, as an interesting and non-trivial corollary, that the number of linearly independent rows in a matrix is equal to the number of linearly independent columns. This is interesting, because it is not obvious at all, at first sight. This number is defined to be the **rank** of the matrix A.

We will end with some ideas relating to the proof of Theorem 2. One observation is the fact that $x = (x_1 \ x_2 \ \ldots \ x_n)^T$ is in $\mathcal{S}$ iff $\sum_{i=1}^{n} x_i A^{(i)} = \mathbf{0}$ where $A^{(i)}$ is the ith column of the matrix A. This is easy to see by writing out the above summation.

[1] If $i = 0$, $S_0 = S$, $T_0 = T$, and we interpret summations of the form $(\sum_{j=1}^{i} \ldots)$ as zero.

CS 435 : Linear Optimization — Fall 2006

Lecture 5: Linear algebra : column space of matrix A, solution space of Ax=0, relationship between them

Lecturer: *Sundar Vishwanathan* — Scribe: *Ayush Choure*
Computer Science & Engineering — Indian Institute of Technology, Bombay

Last Lecture : Given vectors v_1,v_2,v_3,...,v_n the space spanned by them is $\sum_{i=1}^{n} \alpha_i v_i$. Thus $\sum_{i=1}^{n} \alpha_i v_i$ is a subspace.

Consider the space of vectors $\{x : Ax = \overline{0}\}$.

Looking by the column perspective

$$Ax = b \Rightarrow A^1x_1 + A^2x_2 + \ldots + A^nx_n = b$$

here, A^i is the i^{th} column of A and x_i is the i^{th} component of vector x

which means b is in the column space of A. Hence, in the given situation

$$A^1x_1 + A^2x_2 + \ldots + A^nx_n = \overline{0}$$

Assume : $A^1, A^2, \ldots, A^k$ are a basis for the space spanned by $A^1, A^2, \ldots, A^n$, $k \leq n$.

which implies,

$$\begin{aligned} A^{k+1} &= \textstyle\sum_{j=1}^{k} \alpha_j^{k+1} A^j \\ A^{k+2} &= \textstyle\sum_{j=1}^{k} \alpha_j^{k+2} A^j \\ &\cdots \\ A^n &= \textstyle\sum_{j=1}^{k} \alpha_j^{n} A^j \end{aligned}$$

hence, dimension of this space is atleast $n - k$

$$dim\{x : Ax = \overline{0}\} \geq n - k$$

now, to prove that it is EXACTLY $n - k$.

Proof: Take an x_0 such that $Ax_0 = \overline{0}$. Also let U_i be such that,

U_{k+1} is

$$\begin{pmatrix} \alpha_1^{k+1} \\ \alpha_2^{k+2} \\ \vdots \\ \alpha_k^{k+1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

U_{k+2} is

$$\begin{pmatrix} \alpha_1^{k+2} \\ \alpha_2^{k+2} \\ \vdots \\ \alpha_k^{k+2} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Now consider the vectors x and $x\prime$ such that,

$$x = [x_1 \dots x_n]^T$$
$$x\prime = x - \{x_{k+1}U_{k+1} + \dots + x_nU_n\}$$

but

$$Ax = \overline{0} \qquad \text{and}$$
$$A\{x_{k+1}U_{k+1} + \dots + x_nU_n\} = \overline{0}$$

because given that, x and U_i are from the null space of A. Therefore

$$Ax\prime = \overline{0}$$

note that because of way in which $x\prime$ is defined, its last $n-k$ components are zero. Hence above equation means that the combination of first k columns is also zero. Since the first k columns are linearly independent, therefore, the linear combination is *trivial*.Hence,

$$x_1\prime = x_2\prime = \dots = x_n\prime$$

therefore, the dimension of $\{x : Ax_0 = \overline{0}\}$ is EXACTLY $n-k$.

In the next lecture, we will analyse the row perspective.

CS 435 : Linear Optimization — Fall 2006

Lecture 6: Linear Algebra

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe:*Alekh Agarwal*
Indian Institute of Technology, Bombay

1 Row Rank of a matrix

The space $\{x \ : \ Ax = \mathbf{0}\}$ is called the null space of the matrix A. Having already seen that the dimension of the null space of A is equal to the n - number of linearly independent columns in the matrix A, we now examine how this relates to the number of linearly independent rows of A. We will first prove that Gaussian Elimination does not change the number of linearly independent rows. Note that we have already proved that Gaussian Elimination does not change the set $\{x \ : \ Ax = \mathbf{0}\}$. Later we will relate the row rank to the dimension of $\{x \ : \ Ax = \mathbf{0}\}$.

We first prove the following lemma:

Lemma 1 *Gaussian elimination does not change the number of linearly independent rows in a matrix.*

Proof: Gaussian elimination consists of two elementary operations. Exchanging two rows and multiplying a row with a scalar and adding it to another row. It is clear that exchanging two rows does not change the row rank of a matrix Below we will show that multiplying a row with a scalar and adding it to another row does not change the row rank of a matrix.

Consider an $n \times m$ matrix A of rank r. Without loss of generality, let the basis of the row space be $A_1, \ldots, A_r$, where A_i denotes the i_{th} row of A.

Now look at the space spanned by the rows $A_1, A_2, \ldots, A_r$. Suppose A_i is replaced by $A_i + cA_j$. Clearly A_i belongs to this new space and hence all the vectors in the old space are also in the new space. Also $A_i + cA_j$ is in the old space. So the vectors in the new space are also in the old space. Since the space has not changed, the dimension remains unchanged. □

Now, consider the following lemma.

Lemma 2 $\dim(\{x \ : \ Ax = \mathbf{0}\}) \geq n - k$

Proof: By the previous lemma, we can assume that we can use Gaussian Elimination to solve the equation without changing either the null space or the row rank. After the Gaussian elimination, our matrix A looks like:

$$\begin{pmatrix} 0 & \ldots & 1 & 0 & \ldots & 0 & a_{k+1} & \ldots & a_m \\ 0 & \ldots & \ldots 1 & 0 & \ldots & 0 & a_{k+1} & \ldots & a_m \\ & & & \vdots & & & & & \\ 0 & \ldots & \ldots & \ldots & 1 & \ldots & a_{k+1} & \ldots & a_m \\ 0 & & & \ldots & & & & 0 & \\ & & & \vdots & & & & & \end{pmatrix}$$

Here A_t contains a 1 in the position i_t for $i \leq t \leq r$ and $1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq k$. For simplicity, we assume that $i_t = t$.

Now x be any solution to the equation. Then it can be easily seen that we can assign any arbitrary values to $x_{k+1} \ldots x_n$, and then solve for $x_1 \ldots x_k$ to find a solution to the equation.In particular then, we look at the vectors of the form $u^{i-k} = (x_1, x_2, \ldots, x_n) \quad i = k+1, \ldots, n$, where $u_i^{i-k} = 1$ and $u_j^{i-k} = 0$ for $j > k, j \neq i$. Then the u^i's are of the form:

$$\begin{pmatrix} u_1^1 \\ \vdots \\ u_k^1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{pmatrix} u_1^2 \\ \vdots \\ u_k^2 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad \begin{pmatrix} u_1^{n-k} \\ \vdots \\ u_k^{n-k} \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

where u_j^i are obtained as outlined above for $1 \leq j \leq k$. Clearly these $n-k$ vectors $u^1, \ldots, u^{n-k}$ are linearly independent, as there last $(n-k)$ rows are linearly independent. Also each u^i is a solution to $Ax = \mathbf{0}$ by construction. Thus we have $(n-k)$ linearly independent solutions to $Ax = \mathbf{0}$. This proves the lemma. □

Now, as before we will prove that the $\dim(\{x \ : \ Ax = \mathbf{0}\}) = n - k$

THEOREM 3 $\dim(\{x \ : \ Ax = \mathbf{0}\}) = n - k$

PROOF:

Consider any x' such that $Ax' = \mathbf{0}$. Now we construct the new vectors:

$$\begin{aligned} \tilde{x} &= x'_{k+1} u^1 + \cdots + x'_n u^{n-k} \\ x'' &= x' - \tilde{x} \end{aligned}$$

Note that by construction,

$$x''_i = 0, \ i = k+1, \ldots, n. \tag{1}$$

Then we have

$$\begin{aligned} Ax'' &= A(x' - \tilde{x}) \\ &= Ax' - \sum_{i=1}^{n-k} x'_{k+i} A u_i \\ &= \mathbf{0} - \sum_{i=1}^{n-k} x'_{k+i} \mathbf{0} \quad \text{because} Au_i = \mathbf{0} \\ &= \mathbf{0} \end{aligned} \tag{2}$$

Now the matrix $Ax^{''}$ looks like

$$\begin{pmatrix} x_1^{''} & 0 & & \dots & & & \\ 0 & x_2^{''} & 0 & \dots & & & \\ & & & \vdots & & & \\ 0 & \dots & \dots & x_k^{''} & 0 & \dots & \\ 0 & & & \dots & & & 0 \\ & & & \vdots & & & \end{pmatrix} \tag{3}$$

because of the structure of A and Equation 1. Thus by Equations 2 and 3, we get that

$$\begin{aligned} x_i^{''} &= 0 \quad \text{for } i = 1, 2, \dots, k \\ \Rightarrow x_i' &= \sum_{j=1}^{n-k} x_i' u_i^j \quad \text{for } i = 1, 2, \dots, k \end{aligned} \tag{4}$$

Also, we have

$$x_i' = \sum_{j=1}^{n-k} x_{k+i}' u_i^j \quad \text{for } i = k+1, \dots, n \tag{5}$$

by definition of u^i's. Hence, by Equations 4 and 5, we get that

$$x' = \sum_{i=1}^{n-k} x_{k+i}' u^i$$

Thus any other solution to $Ax = \mathbf{0}$ lies in the span of $u_1, \dots, u_{n-k}$. Thus we get

$$dim(\{x \ : \ Ax = \mathbf{0}\}) = n - k$$

Hence proved □

By Theorem 3, we get that the dimension of the null space of the matrix A obtained after applying Gaussian elimination to the original matrix A' is equal to the row rank of A. Having already proved that the Gaussian Elimination does not change the row rank of a matrix, this is also equal to the row rank of the origial matrix A'.

This concludes the proof for the fact that the row rank of a matrix is equal to n - the dimension of its null space. As we have already proved that this is also equal to the column rank of a matrix, we get as a corollary that the row rank and column rank of a matrix are equal.

COROLLARY 4 *The row rank of a matrix is equal to its column rank.*

2 Solutions to $Ax = \mathbf{b}$

Having studied the properties of $\{x \ : \ Ax = \mathbf{0}\}$, we looked at the solutions of $Ax = \mathbf{b}$ for any arbitrary column vector $\mathbf{b}$. In this part, the following result was discussed.

THEOREM 5 *Let x_0 be a vector in $\mathbb{R}^n$ such that $Ax_0 = \mathbf{b}$. Then every solution to $Ax = \mathbf{b}$ can be written in the form $x_0 + x'$, where $Ax' = \mathbf{0}$.*

PROOF: Consider any $\tilde{x}$ such that $A\tilde{x} = \mathbf{b}$. Then, we have

$$\begin{aligned} Ax_0 = \mathbf{b} \quad \text{and} \quad A\tilde{x} &= \mathbf{b} \\ \Rightarrow A(\tilde{x} - x_0) &= \mathbf{0} \end{aligned}$$

Also $\tilde{x} = x_0 + (\tilde{x} - x_0)$. Hence proved □

The solution set to $Ax = \mathbf{b}$ looks like a subspace shifted by a vector x_0.

3 Convex Sets

We look at some of the geometric properties of sets of points in this section. Consider any two points $v_1$and v_2. Then the vector $v_1 + k(v_2 - v_1)$ lies on the line segment joining v_1 and v_2 for $k \in [0, 1]$.Rearranging, we can write this as $(1-k)v_1 + kv_2$, or as $\lambda_1 v_1 + \lambda_2 v_2$ where $\lambda_1 + \lambda_2 = 1$ and $0 \leq \lambda_1, \lambda_2 \leq 1$. What is interesting, however, is that this generalizes to larger sets as well. If we consider a set of n points $S = \{v_1, \ldots, v_n\}$, then any point lying in the polygon with $v_1, \ldots, v_n$ as its vertices can be written as $\sum_{i=1}^{n} \lambda_i v_i$, where $\sum_{i=1}^{n} \lambda_i = 1$ and $0 \leq \lambda_i \leq 1$.
We now define the term *convex combination.*

DEFINITION 1 *Given n vectors $v_1, \ldots, v_n$, vector v of the form*

$$v = \sum_{i=1}^{n} \lambda_i v_i, \quad 0 \leq \lambda_i \leq 1, \quad \sum_{i=1}^{n} \lambda_i = 1$$

is called a convex combination *of $v_1, \ldots, v_n$.*

A *convex set* is defined as:

DEFINITION 2 *A set of points S is called* convex *if for any subset S' of S and for any point p which we get by convex combination of points in S', $p \in S$.*

As an example the set $\{x \ : \ Ax \leq \mathbf{b}\}$ is convex. This is because for any $x_1, \ldots, x_n$ satisfying $Ax_i \leq \mathbf{b}$, $A(\sum_i \lambda_i x_i) = \sum_i \lambda_i A x_i \leq \sum_i \lambda_i \mathbf{b} = \mathbf{b}$ as $\sum_i \lambda_i = 1$.

CS 435 : Linear Optimization — Fall 2006

Lecture 7: Linear Algebra

Lecturer: *Sundar Vishwanathan* — Scribe:*Shaunak Godbole*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Convex Sets

Let us start by defining a *convex combination*.

Definition 1 *Given a set of vectors v_i, any vector of the form*

$$v = \sum_{i=1}^{n} \lambda_i v_i, \quad 0 \leq \lambda_i \leq 1, \quad \sum_{i=1}^{n} \lambda_i = 1$$

is called a convex combination of the v_i's.

Now, using the above definition, a *Convex Set* can be defined as:

Definition 2 *A set S of vectors(points) is called convex if all convex combinations of vectors in S are in S.*

Another way of looking at the definition is:

Definition 3 *If for any two points in S, the line segment joining them is in S, then the set S is said to be convex.*

Figure 1: Example of a convex and a non-convex polygon

We can see in the figure that any line segment joining 2 points in the convex region will lie in the region itself. Thus the region is convex. This is not true with the other non-convex figure. We can see in the figure a line segment which joins 2 points in the polygon is not completely situated in the polygon and hence the polygon is not convex.

THEOREM 1 *If S_1 and S_2 are two convex sets, then $S_1 \cap S_2$ is a convex set.*

PROOF: Let $x_1, x_2 \in S_1 \cap S_2$. Now since x_1 and x_2 belong to S_1 (which is convex), any convex combination of them lies in S_1. Similarly we can say that this convex combination of x_1 and x_2 lies in S_2. Thus the convex combination lies in $S_1 \cap S_2$. Thus $S_1 \cap S_2$ is convex. □

THEOREM 2 *$\{x : Ax \leq b\}$ is a convex set.*

PROOF: Let x_1 and x_2 be in the set. Then

$$Ax_1 \leq b \text{ and,}$$

$$Ax_2 \leq b$$

Consider λ_1 and λ_2 (> 0) such that $\lambda_1 + \lambda_2 = 1$
Then,

$$\lambda_1(Ax_1) + \lambda_2(Ax_2) \leq b(\lambda_1 + \lambda_2)$$
$$\Rightarrow \lambda_1(Ax_1) + \lambda_2(Ax_2) \leq b$$
$$\Rightarrow A(\lambda_1 x_1 + \lambda_1 x_2) \leq b$$

Thus, $\{x : Ax \leq b\}$ is convex. □

An alternate proof could be as follows. Let us look at $A_1 x \leq b_1$. All the points satisfying this ineqality lie on one side of the hyper-plane $A_1 x = b$. Thus the set formed by these points is convex (it is easy to check). Similarly the sets of solutions to the inequalities $A_2 x \leq b_2$, $A_3 x \leq b_3 \ldots (A = [A_1, A_2, \ldots])$ are also convex.

Thus, A can be seen as an intersection of Convex regions. And from Theorm 1 we can argue that since set A is an intersection of convex regions, the set A is convex.

2 Maximize $c^T x$

How do we maximize $c^T x$ over the set of all x satisfying $Ax \leq b$?

First, consider maximizing over the region $x^T x \leq 1$. This is a set such that all points are at a distance less than or equal to 1 from the origin. We can see that this is a convex set.

We know that $c^T x$ increases in the direction of c. Thus we start moving in the direction of c. The last point where $c^T x$ touches the sphere is the point of maxima. It can be easily seen that at this point, $c^T x$ is a tangent to the sphere $x^T x$.

Now consider any convex polygon on a 2-d plane. To maximize $c^T x$ in 2-d keep moving along c. The last point where $c^T x$ touches the polygon will the point of maxima. It can be observed that this point will be a boundary point.

Extending this argument to a n-dimensional plane, it seems that $c^T x$ will attain its maximum value at the boundary points of the region $Ax \leq b$.

A local maximum of a function f can be defined as follows.

DEFINITION 4 *If there exists a small neighbourhood N of x_0 where $f(x_0) \geq f(x) \forall x \in N$, then x_0 is said to be a point of local maxima.*

THEOREM 3 *If f is linear and f convex, then a local maximum is a global maximum.*

PROOF: Let x_0 be a local maximum and y be a global maximum.
Consider a point $P = (1 - \epsilon)x_0 + \epsilon y$. If $\epsilon \to 0$, then P lies in any small neighbourhood of x_0 (in particular, in the neighbourhood where x_0 is the point of local maximum).
Now, consider $f((1 - \epsilon)x_0 + \epsilon y)$. Since f is linear, this can be written as

$$(1 - \epsilon)f(x_0) + \epsilon f(y)$$

$$= f(x_0) + \epsilon(f(y) - f(x_0))$$

We can now observe that at point P, which is in the neighbourhood of x_0, the value of the function $f((1 - \epsilon)x_0 + \epsilon y) \geq f(x_0)$. Thus $f(x_0)$ can be maximum only if $f(x_0) = f(y)$.
Thus, we can see that a local maximum is the same as a global maximum. □

CS 435 : Linear Optimization — Fall 2006

Lecture 8: Extreme points

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Aditya G Parameswaran*
Indian Institute of Technology, Bombay

In this lecture, we define *extreme points* in geometry and try to give an linear algebra perspective for the same.

Definition 1 *Given points $p_1, p_2, p_3, \ldots, p_n$, the convex hull is the smallest convex set containing these points.*

To simplify the discussion, we make the following assumptions on the nature of $Ax \leq \mathbf{b}$ which hold for the rest of the course unless otherwise stated.

1 Assumptions on the nature of the convex set $Ax \leq \mathbf{b}$

- *Assumption 1:* $Ax \leq \mathbf{b}$ is bounded. No single coordinate of x satisfying $Ax \leq \mathbf{b}$ can increase or decrease without bound.
- *Assumption 2:* $Ax \leq \mathbf{b}$ has no degeneracies. This is equivalent to saying that not more than n hyperplanes pass through a point in an n-dimensional space. This assumption makes sense because small perturbations of the hyperplanes can remove the degeneracies.
- *Assumption 3:* $Ax \leq \mathbf{b}$ should be *full dimensional*. This is equivalent to saying that one should be able to place a n-dimensional sphere, however small, in the region defined by $Ax \leq \mathbf{b}$. For two dimensions, the convex set should have area, and for three dimensions, the convex set should have volume. In n dimensions, the convex set should have an n-dimensional volume.

2 Extreme Points and its Algebraic Interpretation

Definition 2 *An extreme point is a point in a convex set that cannot be represented as a convex combination of any two other distinct points in the convex set.*

Thus, an extreme point does not lie on the segment between two other distinct points of the convex set. Intuitively, all segments with the extreme point in the interior "stick out" of the convex set.

Theorem 1 *Given a convex set $Ax \leq \mathbf{b}$ in n-dimensional space satisfying the assumptions of Sec. 1, the extreme points of a convex set are precisely those points in $Ax \leq \mathbf{b}$ which can be expressed as an intersection of n linearly independent hyperplanes out of the set of hyperplanes that define the convex set $Ax \leq \mathbf{b}$.*

PROOF:
Part 1: ($\Rightarrow$) Any point which can be expressed as an intersection of n linearly independent hyperplanes out of the set of hyperplanes that define the convex set $Ax \leq \mathbf{b}$ is an extreme point.

Let x_0 be any such point. We split $Ax_0 \leq \mathbf{b}$ into two parts

$$A^{'}x_0 = \mathbf{b}^{'} \quad (1)$$
$$A^{''}x_0 < \mathbf{b}^{''} \quad (2)$$

(We effectively move the inequalities in $Ax_0 \leq \mathbf{b}$ that are equalities into the first set $A^{'}x_0 = \mathbf{b}^{'}$)

We notice that since x_0 is a point on the intersection of n linearly independent hyperplanes in the defining set of hyperplanes, $A^{'}$ has n linearly independent rows.

Let there be x_1 and x_2 in the given convex set such that $x_0 = \lambda x_1 + (1-\lambda)x_2$, $0 < \lambda < 1$. We then have

$$\lambda A^{'}x_1 + (1-\lambda)A^{'}x_2 = \mathbf{b}^{'} \quad (3)$$

But since x_1 and x_2 are part of the convex set, $A^{'}x_1 \leq \mathbf{b}^{'}$ and $A^{'}x_2 \leq \mathbf{b}^{'}$.

Neither of these inequalities can be strict (or else Eq. 1 will not hold) and therefore $A^{'}x_1 = \mathbf{b}^{'}$ and $A^{'}x_2 = \mathbf{b}^{'}$.

Now, we are given that x_0 is a solution of $A^{'}x = \mathbf{b}^{'}$, which is nothing but the intersection of n linearly independent hyperplanes. Thus, we have $x_0 = x_1 = x_2$.

Thus x_0 cannot be expressed as a convex combination of two other distinct points in the convex set and is therefore an extreme point.

Part 2: ($\Leftarrow$) Let x_0 be an extreme point. If we split $Ax_0 \leq \mathbf{b}$ into two parts

$$A^{'}x_0 = \mathbf{b}^{'} \quad (4)$$
$$A^{''}x_0 < \mathbf{b}^{''} \quad (5)$$

then A' has n linearly independent rows.

See next lecture for a proof.

□

CS 435 : Linear Optimization — Fall 2006

Lecture 9: Convex hull of extreme points

Lecturer: *Sundar Vishwanathan* — Scribe: *Ankur Taly*
Computer Science & Engineering — Indian Institute of Technology, Bombay

In this lecture, we complete the proof of the thoerem on extreme points mentioned in the previous lecture and then begin the proof of the thoerem on convex hull of the extreme points of $\{x : Ax \leq \mathbf{b}\}$.

Proof(contd): Here we prove the converse relation that is every extreme point of $\{x : Ax \leq \mathbf{b}\}$ can be expressed as an intersection of n linearly independent hyperplanes
We prove this by showing that if a point cannot be expressed as an intersection of n linearly independent hyperplanes then we can express it as a convex combination of two points in the set.
Let x_0 be a point. We split $Ax_0 \leq \mathbf{b}$ into two parts

$$A^{'}x_0 = \mathbf{b}^{'} \quad (1)$$
$$A^{''}x_0 < \mathbf{b}^{''} \quad (2)$$

Since $A^{''}x_0$ is strictly less than $\mathbf{b}^{''}$,we can draw a small enough sphere around x_0 such that every point x within the sphere satifies $A^{''}x_0 < \mathbf{b}^{''}$. This means that $\exists$ a (= radius of this sphere) such that for which all vectors $\bar{\epsilon}$ which have magnitude($|\bar{\epsilon}|$) $\leq a$ we have

$$A^{''}(x_0 + \bar{\epsilon}) < \mathbf{b}^{''} \quad (3)$$

Now assume that $A^{'}$ does not have n linearly independent vectors. Then $A^{'}x = 0$ will have a non zero solution. Let x_1 be a non zero solution of $A^{'}x = 0$. Then it is easy to observe $\forall \lambda \in R$

$$A^{'}(x_0 + \lambda x_1) = \mathbf{b}^{'} \quad (4)$$
$$A^{'}(x_0 - \lambda x_1) = \mathbf{b}^{'} \quad (5)$$

By making λ small we can make magnitude of λx_1 to become less than a and so by Eq. 1

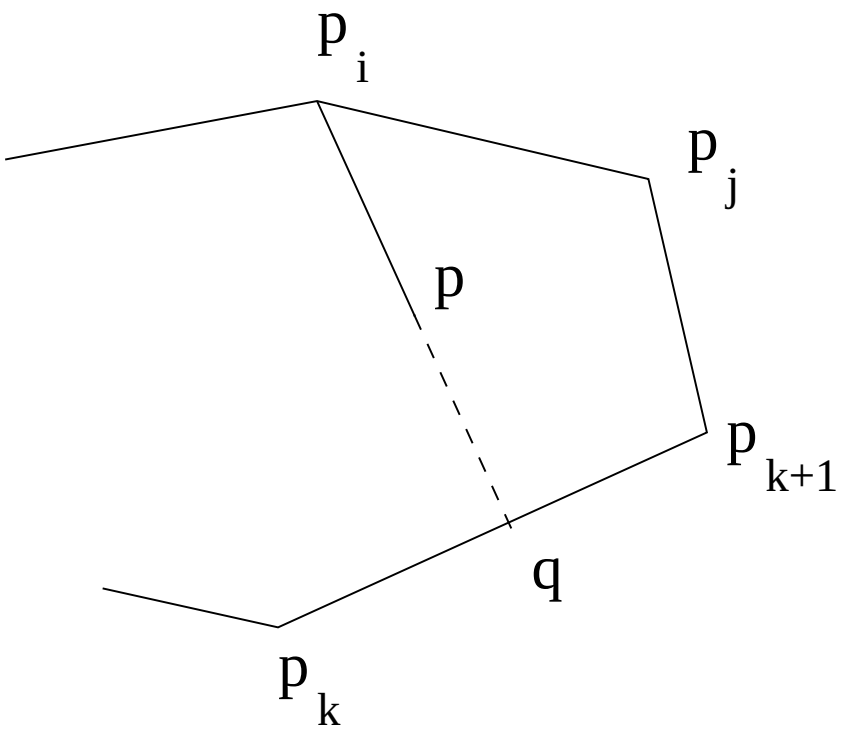
$$A^{''}(x_0 + \lambda x_1) < \mathbf{b}^{''} \quad (6)$$
$$A^{''}(x_0 - \lambda x_1) < \mathbf{b}^{''} \quad (7)$$

So $x_0 + \lambda x_1$ and $x_0 - \lambda x_1$ are points in $\{x : Ax \leq \mathbf{b}\}$ and also $x_0 = (1/2)(x_0 + \lambda x_1) + (1/2)(x_0 - \lambda x_1)$. Thus we have expressed x_0 as a convex combination of two points in the set. This is a contradiction !. Hence we are proving the theorem.

1 Convex hull of the extreme points

Definition 1 *A convex hull of finite points $p_1, p_2, \ldots p_n$ is the set of all points p which can be written as convex combination of $p_1, p_2, \ldots p_n$.*

We have the following theorem



THEOREM 1 *Let $p_1, p_2, \ldots p_n$ be extreme points of $x : Ax \leq \mathbf{b}$. Then $x : Ax \leq \mathbf{b}$ is the convex hull of the points $p_1, p_2, \ldots p_n$*

PROOF:
It is easy to observe that any convex combination of $p_1, \ldots p_n$ belongs to the set since $p_1, \ldots p_n$ belongs to $x : Ax \leq \mathbf{b}$. So

$$\text{Convex hull of } p_1, \ldots p_n \subset \{x : Ax \leq \mathbf{b}\} \tag{8}$$

What remains to show is that any x_0 such that $Ax_0 \leq \mathbf{b}$ can be written as

$$x_0 = \Sigma \lambda_i p_i \text{ where } \Sigma \lambda_i = 1 \ , \ 0 \leq \lambda_i \leq 1 \tag{9}$$

We show this by induction on dimensions.
Base case : Consider the 2-D space and a convex set $\{x : Ax \leq \mathbf{b}\}$ in it. Let p be any point inside the set. Now we take the extreme point p_i and join it to p. Next we extend this line untill it touches one of the bounding segments ($p_k p_{k+1}$ in this case) at some point (say q)
Since q lies on the segment joining p_k and $p_k + 1$ it can be expressed as a convex combination of p_k and $p_k + 1$. Therefore,

$$q = \lambda_1 p_k + \lambda_2 p_k + 1 \text{ where } \lambda_1 + \lambda_2 = 1. \tag{10}$$

Also p lies on the segment joining q and p_i. So p can expressed as a convex combination of q and p_i. Therefore,

$$p = \lambda_3 p_i + \lambda_4 q \text{ where } \lambda_3 + \lambda_4 = 1. \tag{11}$$

Combining the above equations we get

$$p = \lambda_3 p_i + \lambda_4 \lambda_1 p_k + \lambda_4 \lambda_2 p_k + 1 \tag{12}$$

Now $\lambda_3 + \lambda_4\lambda_1 + \lambda_4\lambda_2 = 1$. Thus we have expressed p as a convex combination of the extreme points. Hence the result is true in 2-D.

In next class.

□

CS 435 : Linear Optimization — Fall 2006

Lecture 10: $Ax \leq b$ as a convex combination of its extreme points

Lecturer: *Sundar Vishwanathan* — Scribe: *Hidayath Ansari*
Computer Science & Engineering — Indian Institute of Technology, Bombay

In this lecture, we complete the proof of a theorem stating that all points in the set $Ax \leq \mathbf{b}$ can be expressed as a convex combination of its extreme points. We then prove that a linear function on such a set is maximized at an extreme point, and show how that is used to construct the Simplex algorithm.

Theorem 1 *Let $p_1, p_2, p_3, \ldots, p_t$ be the extreme points of the convex set $S = \{x : Ax \leq \mathbf{b}\}$ Then every point in S can be represented as $\sum_{i=1}^{t} \lambda_i p_i$, where $\sum_{i=1}^{t} \lambda_i = 1$ and $0 \leq \lambda_i \leq 1$*

Proof: Proof is by induction on the dimension.
Consider $p \in S$. Join p_1 to p and extend to meet q on the boundary. For the point q, we must then have

$$A_1 q = b_1 \tag{1}$$

$$A^{''} q < \mathbf{b}^{''} \tag{2}$$

(where $A^{''}$ is the rest of A), because q is on the boundary, and $A_1 q > b_1$ outside the feasible region (having crossed the hyperplane). From the first equality, we solve for one variable, say x_n and replace it throughout in $A^{''}$. This allows us to construct a new convex set $S^{'} = \{x : Cx \leq \mathbf{d}\}$, in *one less dimension.*

By the induction hypothesis, q can be written as a convex combination of extreme points in this object, $S^{'}$. Hence,

$$p = \beta p_1 + (1 - \beta) q \tag{3}$$

$$= \beta p_1 + (1 - \beta) \sum_{i=1}^{t^{'}} \gamma_i q_i \tag{4}$$

This however is in terms of the extreme points $q_1, q_2, q_3, \ldots, q_{t'}$ of $S^{'}$. We show that the extreme points of $S^{'}$ are also extreme points of S. Suppose they were not. Let $p^{'}$ be an extreme point of $S^{'}$ but not of S. Then $\exists\ p_1^{'}, p_2^{'} \in S$ such that $p^{'} = \lambda p_1^{'} + (1 - \lambda) p_2^{'}$. By construction of points in $S^{'}$,

$$b_1 = A_1 p^{'} \tag{5}$$

$$= \lambda A_1 p_1^{'} + (1 - \lambda) A_1 p_2^{'} \tag{6}$$

But since we have $A_1 p_1^{'} \leq b_1$ and $A_1 p_2^{'} \leq b_1$ (both p_1 and p_2 are in S), we must have the equality holding in both for the above equality (6) to be true. Therefore $p_1^{'}$ and $p_2^{'}$ must also be in $S^{'}$. $p^{'}$ can not then be extreme in $S^{'}$, as it is the convex combination of two points in the same set.

For the base case, take the dimension to be 0. This completes the proof. □

The following theorem will put the last step in place to construct an algorithm for solving LP problems.

Theorem 2 *A linear function on $S = \{x : Ax \leq \mathbf{b}\}$ is maximized at an extreme point.*

Proof: Let a linear function f attain its maximum at point p, where $p = \sum_{i=1}^{t} \lambda_i p_i$ (This is a valid assumption by the previous theorem). Then $f(p) = \sum_{i=1}^{t} \lambda_i f(p_i)$. If all of the $f(p_i)$'s were lesser than $f(p)$, their convex combination cannot sum to $f(p)$. Therefore for at least one i, $f(p_i) = f(p)$. □

After having proved this, we have a finite algorithm at our disposal now. An extreme point is an intersection of n linearly independent hyperplanes. We just need to pick all combinations of n rows from A ($\binom{m}{n}$ in number), solve for x_0 in $A^{'} x_0 = \mathbf{b'}$ using Gaussian Elimination, **verify** that the solution indeed satisfies all other inequalities, and then calculate $c^T x$.

The verification part is important, as the n hyperplanes we choose may end up defining an infeasible point. An example is 2-D is shown in Figure 1.

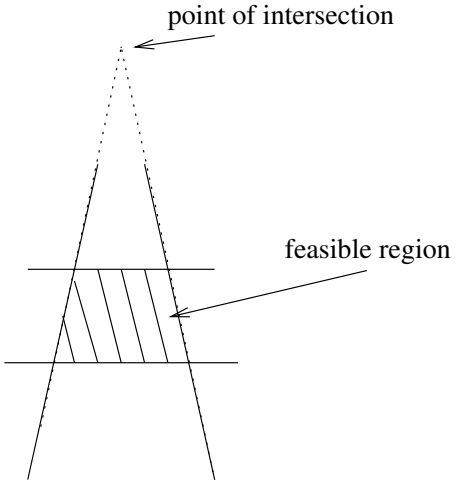


Figure 1: Why we need to verify

A rather simple formulation of the algorithm could then be:

Start at an extreme point.
While a neighbour of higher cost exists, move to it.

Intuitively, this would work, as by a previous result a local maximum in such a problem is also a global maximum. A more formal description of the Simplex algorithm and proof of its correctness is done in subsequent lectures.
Questions raised at this juncture are:

1. How do we start the process? It is pointless to obtain all extreme points and then pick one from among them.
2. How do we move to a neighbour?
3. Why are we guaranteed that the optimal is attained when we stop?

Lecture 10: Extreme Points

Lecturer: *Sundar Vishwanathan* Scribe:*Swati Goyal*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Extreme Points

Definition 1 *An extreme point in a convex set is a point which cannot be represented as a convex combination of two other points of the set.*

In the last lecture, we proved that in a space of dimension n, an extreme point is the intersection point of n linearly independent hyperplanes i.e
Take an x_0 such that $Ax_0 \leq b$ given by

$$A^{'}x_0 = b^{'} \tag{1}$$

$$A^{''}x_0 < b^{''} \tag{2}$$

then x_0 is an extreme point iff $A^{'}$ consists of n linearly independent rows(hyperplanes). Note that we have assumed $Ax_0 \leq b$ to be non-degenerate.

2 Convex Combination of Extreme Points

Consider a region $Ax \leq b$
Let $p_1, p_2..p_n$ be the extreme points of this region. Then this region can also be represented as the convex hull of these extreme points.
We will prove this statement in the following section.

Theorem 1

Any point x, such that $Ax \leq b$, can be written as convex combination of the extreme points of this region.

Proof: The proof will use induction on dimension of the region.
Base Case: Dimension of the region is 0.Then there is just one point and hence the theorem is vacuously true. We illustrate the main technique by looking at an example in 2 dimension. Consider the region in Fig1.

Figure 1: Convex polygon in dimension 2

Let p be any point in the given region and say $p_1, p_2 \ldots$ be the extreme points of the region. Now join p_1 and p and extend it to intersect p_3p_4 at q. Since q lies on the line joining p_3p_4 it can be written as a convex combination of p_3 and p_4.

$$q = \Sigma\alpha_i p_i, 0 \leq \alpha_i \leq 1, \Sigma\alpha_i = 1, i = 3, 4 \tag{3}$$

$$p = \beta_1 p_1 + \beta_2 q, 0 \leq \beta_i \leq 1, \Sigma\beta_i = 1 \tag{4}$$

and hence

$$p = \Sigma\gamma_i p_i, 0 \leq \gamma_i \leq 1, \Sigma\gamma_i = 1 \tag{5}$$

Hypothesis: Assume that any x, such that $Ax \leq b$ in space of dimension $n-1$ can be written as convex combination of the extreme points of the region.

Induction: Consider a region $S : Ax \leq b$ in space of dimension n with $p_1, p_2 \ldots$ as the extreme points
Consider a point $p \in S$. Join p_1 and p and extend the line segment to meet a point q on the boundary of S.At this boundary, at least one of the inequalities in $Ax \leq b$ has to be equality. For simplicity assume that the first is equality.
Thus at q,

$$A_1 q = b_1 \tag{6}$$

$$A'' q < b'' \tag{7}$$

where $q = [q_1, q_2, \ldots]$
Solve (6) for some q_i and substitute its value in (7).
Thus (7) now reduces to another region S' of dimension $n-1$ of the form $Cx \leq D$.
By assumed hypothesis, q can be written as convex combination of extreme points of S'.

$$q = \Sigma\alpha_i (p')_i \texttt{ where } \Sigma\alpha_\texttt{i} = \texttt{1 and } \texttt{0} \leq \alpha_\texttt{i} \leq \texttt{1} \tag{8}$$

Now p can be written as a convex combination of q and p_1 i.e. as a convex combination of $p_1' \ldots$ and p.But this still does't prove that p is a convex combination of $p_i \ldots$ (extreme points of S)
For this we would have to prove that the extreme points of S' are a subset of the extreme points of S.
Assume this is not true,that is, p' is an extreme point of S' but not of S.

Since, $p^{'} \in S^{'}$ and as $S^{'}$ is a subset of S, thus $p^{'} \in S$.
Also , $\exists p_1, p_2 \in S$ such that $p^{'}$ is the convex combination of p_1, p_2(as $p^{'}$ is not an extreme point in S).

$$p^{'} = \lambda p_1 + (1-\lambda)p_2 \ 0 \leq \lambda \leq 1 \tag{9}$$

Now, for $p^{'}$ to be an extreme point in S,

$$A_1 p^{'} = b_1 \tag{10}$$

must be true, from (6), as (7) is already satisfied for all points in region $S^{'}$.
i.e.

$$A_1 \lambda p_1 + A_1(1-\lambda)p_2 = b_1 \ \texttt{from(9)and(10)} \tag{11}$$

Since $A_1 p_1 \leq b_1$ and $A_1 p_2 \leq b_1$, both of them have to be equal to b_1. Thus, p_1 and p_2 satisfy (6) and thus belong to $S^{'}$.
This leads to a contradiction since $p^{'}$(extreme point) cannot be expressed as convex combination of two or more points of the region.
Hence, the extreme points of $S^{'}$ are a subset of the extreme points of S.
Thus,

$$q = \Sigma \alpha_i p_i^{'} = \Sigma \alpha_i p_i \tag{12}$$

$$p = \beta_1 p_1 + \beta_2 q \tag{13}$$

$$p = \Sigma \gamma_i p_i \ \texttt{from(12)and(13)} \tag{14}$$

Thus, every point in S can be expressed as the convex combination of the extreme points of S. □

3 Maximize $c^T x$

Given a linear function $c^t x$ and a region $Ax \leq b$, such that the dimension of the space is n and the number of rows in A is m,then the number of the extreme points of the region is C(m,n). Each extreme point is an intersection of n linearly independent planes, and thus the number of extreme points is at most the number of possible ways of choosing n planes from m choices.
But before finding the maximum value for the function in the given region, we will have to prove a theorem.

THEOREM 2

The maximum value of a linear function f(x) occurs at an extreme point of the region $Ax \leq b$.

PROOF: Let f be maximum at point p of the region $Ax \leq b$.
From the previous theorem we have,

$$p = \sum \alpha_i p_i \ 0 \leq \alpha_i \leq 1 \ \Sigma \alpha_i = 1 \tag{15}$$

where $p_1 \ldots$ are the extreme points of S.
Thus,

$$f(p) = f(\sum \alpha_i p_i) \tag{16}$$

$$f(p) = \sum \alpha_i f(p_i) \tag{17}$$

since f is a linear function.

$$\sum \alpha_i f(p_(i)) \leq \theta \sum \alpha_i \tag{18}$$

where θ is the maximum value among f(p_i).Thus

$$\theta = f(p) = \sum \alpha_i f(p_i) \leq \theta \sum \alpha_i \leq \theta \tag{19}$$

Thus there must exist some extreme pointp_i such that $f(p) = f(p_i)$, otherwise the equality cannot be satisfied in (19).
Hence, the maximum value of f occurs at an extreme point. □

In next lecture, we will look at the simplex algorithm for finding the extreme points of the region $Ax \leq b$.

CS 435 : Linear Optimization | Fall 2006

Lecture 11: Simplex Algorithm: Finding a neighbour of larger cost

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Sanchit Garg*
Indian Institute of Technology, Bombay

1 Overview

1. Basic Approach for algorithm
   (a) Start with an extreme point.
   (b) Move to a neighbour of larger cost if one exists.
2. Issues with this approach
   (a) To find starting point
   (b) How to move from one extreme point to one of its neighbour (`Peberting`)
   (c) When algorithm stops, and to show that we have the point which maximizes $c^T A$
3. We have already accomplished following results in previous lectures
   (a) A linear function on $Ax \leq b$ is maximized at an extreme point.
   (b) The intersection of n linearly independent hyperplanes for equality (and hence corresponding n linearly independent rows of A) gives an extreme point if it satisfies the other inequalities.

2 Some statements/results

1. The intersection of n linearly independent rows/hyperplanes yield an extreme point. Also the intersection of $n-1$ linearly independent hyperplanes will give a line.
2. For each extreme point, there will be n neighbouring extreme points. One corresponding to each line given by different sets of $n-1$ hyperplanes excluding one from the n hyperplanes that define the extreme point in consideration.
3. The direction of the vectors from an extreme point to its neighbours are given by the columns of $-A'^{-1}$. A' is a $n \times n$ matrix such that $A'x_0 = b$.

   Proof: A' consists of n linearly independent rows which solve to give a unique solution x_0. Consider points on the line in between x_0 and one of its neighbour y_i. Since they all lie on line segment joining x_0 and y_i, they satisfy $n-1$ rows of A' with equality and remaining ones with inequality. Lets say i^{th} row of A' will be an inequality. Therefore points in between x_0 and y_i satisfy the

following equations.

$$\begin{aligned} A_i' y_i &< b_i \\ A_1' y_i &= b_1 \\ A_2' y_i &= b_2 \\ &\vdots \\ A_{i-1}' y_i &= b_{i-1} \\ A_{i+1}' y_i &= b_{i+1} \\ &\vdots \\ A_n' y_i &= b_n \end{aligned}$$

Clearly, direction of vectors from x_0 to its neighbours will be given by

$$y_1 - x_0, y_2 - x_0, y_3 - x_0, \ldots, y_n - x_0 \tag{1}$$

where y_i is a neighbouring extreme point of x_0.
Consider a matrix P whose columns are given by $y_i - x_0$.

$$P = (y_1 - x_0, y_2 - x_0, y_3 - x_0, \ldots y_n - x_0) \tag{2}$$

$$\begin{aligned} &Since\ A'_j y_i = b_j = A'_j x_0, \forall j \neq i\ (i, j = 1 \ldots n) \\ &\quad \Rightarrow A'_j (y_i - x_0) = 0, \forall j \neq i \end{aligned} \tag{3}$$

$$\begin{aligned} &Also\ A'_i y_i \leq b_i \\ &\quad \&\ A'_i x_0 = b_i \\ &\quad \Rightarrow A'_i (y_i - x_0) \leq 0 \end{aligned} \tag{4}$$

This gives $A'P$ as

$$\begin{bmatrix} (-)ve & 0 & 0 & \ldots & 0 \\ 0 & (-)ve & 0 & \ldots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \ldots & (-)ve \end{bmatrix}$$

Now $A'(-A'^{-1})$ would be

$$\begin{bmatrix} -1 & 0 & 0 & \ldots & 0 \\ 0 & -1 & 0 & \ldots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \ldots & -1 \end{bmatrix}$$

This means that $y_i - x_0$ is a scaled version of the i^{th} column of $-A'^{-1}$. Hence the direction vectors of the neighbours of x_0 are given by the columns of $-A'^{-1}$.

3 Simplex Algorithm

Recall that we started with an extreme point and if there exists a neighbour of larger cost we moved to that neighbour. To check if x_0 has a neighbour y_i with larger cost we need to determine if the cost

increases in the direction of $y_i - x_0$ (which is given by i^{th} column of $-A'^{-1}$). If for any y_i this cost increases then y_i is our next extreme point and we repeat the procedure. The Algorithm stops when we have no such neighbour, and the output is the final extreme point.
If we have found that the cost increases in the direction of $y_i - x_0$, we need to find y_i to proceed further. Since x_0 & y_i lie on same line, they have $n-1$ hyperplanes in common. To find y_i we need to get the n^{th} hyperplane. Consider

$$A(x_0 + \epsilon v_i) \leq b \tag{5}$$

where v_i is the direction vector $(y_i - x_0)$. With $\epsilon = 0$, we have $A'x_0 = b$. As we gradually increase ϵ, at some value one of the inequalities will become an equality. This yields the point y_i.
Note: There would be many hyperplanes that would solve with these $n-1$ hyperplanes to give a point. But they wont be extreme point as they might not satisfy other inequalities. Only, point closest to x_0 on line will satisfy all conditions and it would be an extreme point.

What remains is to prove that if all the neighbours of x_0 have cost at most the cost of x_0, then x_0 will be a point of maximum cost. This will be taken in the next lecture.

CS 435 : Linear Optimization — Fall 2006

Lecture 11: Simplex Algorithm: Determining a neighbour of greater cost

Lecturer: *Sundar Vishwanathan* — Scribe: *Rajhans Samdani*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Simplex Algorithm

First have a relook at the simplex algorithm :

1. Start at an extreme point.
2. Move to a "neighbour" of greater cost if one exists and repeat the same procedure with it. If no such neighbour exists then exit with this point as the optimum point.

2 The notion of directions

Lets consider an example

Example 1 Consider two half-planes in 2-d space given by

$$3x + 4y = 12$$
$$7x + 3y = 6$$

We have to find out their point of intersection and the direction of the vectors v_1 and v_2 "along" the lines. It turns out in this case that if the coordinates of the point of intersection are (x, y) then we can find out x and y by solving the equation

$$\begin{bmatrix} -7 & 3 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 12 \end{bmatrix}$$

Also the direction of potential neighbours can be determined to be the vectors $\begin{bmatrix} 4 \\ -3 \end{bmatrix}$ and $\begin{bmatrix} -3 \\ -7 \end{bmatrix}$, respectively. If we write v_1 and v_2 as columns of a matrix B, $\begin{bmatrix} -3 & -7 \\ 4 & -3 \end{bmatrix}$ then we observe that AB is a diagonal matrix with negative entries on the diagonal. □

We generalize this to n dimensions. Consider the usual linear optimization problem with the constraint $Ax \leq b$ and the cost function $c^\intercal x$. Consider any of its extreme points x_o. The point x_o satisfies the equation

$$A' x_o = b' \tag{1}$$

where A' is the matrix formed by certain n linearly independent rows of the matrix A and b' is a vector having corresponding coefficients from b. Now, from x_o there are n lines or directions coming out which "connect" it to its neighbours. We wish to find out the vectors corresponding to these directions.
Let the matrix A' be written as

$$\begin{bmatrix} A'_1 \\ A'_2 \\ \vdots \\ A'_n \end{bmatrix}$$

where A'_i is the i^{th} row. Now consider a point x_i on any line l_i coming out of x_o. Clearly this line is determined by the intersection of certain $n-1$ rows out of the n linearly independent rows of A'. Let x_i be such that

$$A'_j x_j = b'_j \quad \forall j, 1 \leq j \leq n, j \neq i \tag{2}$$
$$A'_j x_j < b'_j \qquad j = i \tag{3}$$

The direction of the vector along the line l_i from the point x_o towards x_i is $x_i - x_o$. Now we construct a matrix, having as columns all such vectors for $i = 1, 2, ..n$,

$$Z = \begin{bmatrix} x_1 - x_o, & x_2 - x_o, & \ldots, & x_n - x_o \end{bmatrix}$$

Now consider

$$A'Z = \begin{bmatrix} A'x_1 - A'x_o, & A'x_2 - A'x_o, & \ldots, & A'x_n - A'x_o \end{bmatrix} \tag{4}$$

Given the properties of x_i, we know that

$$(A'x_i)^j = (A'x_o)^j \quad j \neq i \tag{5}$$
$$(A'x_1)^j < (A'x_o)^j \quad j = i \tag{6}$$

So it is clear that the i^{th} row of the matrix $A'Z$ has a negative entry at the i^{th} position and zeroes at all other positions which means that matrix $A'Z$ is a diagonal matrix with negative entries on the diagonal. Clearly enough we can multiply each column of the matrix Z with a suitable **negative** constant such that

$$A'Z = I, \tag{7}$$

where I is an identity matrix. And now since we only care about the sign of a direction vector and not its magnitude(which basically means we can ignore the magnitude of the constants multiplied with each column of Z to get $A'Z = I$) so we can say the following about the directions coming out of the point x_o

THEOREM 1 *The direction vectors are the columns of the negative of the inverse of matrix A'.*

3 Finding neighbouring points with higher cost

One of the steps in the simplex algorithm says "move to the neighbour with larger cost". Once we know the direction towards each neighbour, it is very easy to find out which of the neighbours have higher cost. This is because of the fact that if one of the neighbours x_i has higher cost than the current extreme point x_o, then every point on the line segment joining x_i and x_o has cost higher than x_o. Lets prove this.

PROOF: Let x' be any arbitrary point on the line segment joining x_o and x_i such that $x' \neq x_o$. Now, we know that any point on the line segment joining x_o and x_i can be written as a convex combination of these two. So,

$$x' = \lambda x_i + (1 - \lambda) x_o \tag{8}$$

, where $\lambda > 0$. So we get,

$$c^\intercal x' = \lambda c^\intercal x_i + (1 - \lambda) c^\intercal x_o \tag{9}$$

or,

$$c^\intercal x' = \lambda (c^\intercal x_i - c^\intercal x_o) + c^\intercal x_o \tag{10}$$

And since $c^\intercal x_i > c^\intercal x_o$ so clearly $c^\intercal x' > c^\intercal x_o$

□

So choose any point on the line segment joining x_i and x_o given by a column of $(-A')^{-1}$ and see if it has higher cost than x_o. If it has, then the neighbour x_i will also have higher cost than x_o otherwise not.

After having found out whether a particular direction fetches higher cost or not, now we move on to actually finding that neighbouring point. Given the direction vector v_i from x_o to x_i, any point x' on the line segment joining these two can be written as

$$x' = x_o + \epsilon v_i \quad \epsilon \geq 0 \tag{11}$$

where ϵ is suitably chosen so that the point x' doesn't shoot past x_i. This can be expressed in the constraint

$$A(x_o + \epsilon v_i) \leq b \tag{12}$$

We know that the point $x' = x_o + \epsilon v_i$ satisfies

$$A'_j x' = b'_j \quad \forall j \neq i, \quad 1 \leq j \leq n \tag{13}$$

So we keep increasing the value of ϵ until x' starts satisfying

$$A_k x' = b_k \tag{14}$$

where A_k is linearly independent of all rows of A'. This x' will be the required neighbour.
Hence,

$$\epsilon = min_t \frac{b_t - A_t x_o}{A_t v_i}, \tag{15}$$

where t ranges over all those rows of A which do not belong to A'.

The neighbouring point can also be found in another way.

1. We take $n-1$ linearly independent rows of A' which decide one particular direction.
2. Then we add to them any row from the matrix A such that this row is linearly independent of all the n rows of the matrix A' and then we find out the point determined by this new set of rows.
3. If this point is feasible, that is it satisfies $Ax \leq b$ then this is the required neighbouring point, else we repeat this process with other rows of the matrix A until we find the neighbouring point.

CS 435 : Linear Optimization — Fall 2006

Lecture 12: The Simplex Algorithm III

Lecturer: *Sundar Vishwanathan* — Scribe: *B. Aditya Prakash*
Computer Science & Engineering — Indian Institute of Technology, Bombay

In this lecture we will give the stopping condition of the Simplex algorithm and also prove that the algorithm is correct.

To recap, at a given extreme point x_0:

1. There are matrices $A^{'}$, $b^{'}$, $A^{''}$ and $b^{''}$ (constructed from A and b) s.t. $A^{'}x_0 = b^{'}$ and $A^{''}x_0 \leq b^{''}$.
2. The directions of the neighbouring extreme points are the columns of the matrix $-A^{'-1}$.

Stopping Condition *The algorithm stops at an extreme point x_0 and returns it as optimal when the cost at all the neighbouring extreme points of x_0 is less than that at x_0.*

1 Proof of Correctness

We now try to prove that the Simplex algorithm is correct i.e. when it terminates, we indeed have found the globally optimal point.

First of all note that this is *not* the same as saying that x_0 is local maximum and hence by a previously proved theorem, it is also a global maximum. This is so because we just know that the cost at x_0 is maximum as compared to its *neighbours* - not compared to a small enough *neighbourhood* around it.

1.1 First Approach

One approach would be to consider a small enough neighbourhood N around x_0. Now suppose we somehow prove that any point $p \in N$ can be written as a convex combination of all the neighbours of x_0, i.e.:

$$p = \sum_{i=0}^{n} \lambda_i x_i \tag{1}$$

$$\sum \lambda_i = 1 \tag{2}$$

Then we are clearly done because we know that $\forall i \; c^T x_i \leqslant c^T x_0$. Hence,

$$c^T p = \sum \lambda_i c^T x_i \leqslant c^T x_0 (\sum \lambda_i) = c^T x_0 \tag{3}$$

Thus, x_0 is a local maximum and consequently a global maximum.

1.2 Second Approach

We adopt a different approach than the above for the proof. Assume that x_0, where the algorithm terminates, is not an optimal point. Also, suppose that there is some other optimal point x_{opt}. Therefore, $c^T x_{opt} > c^T x_0$.

Now as x_0 is an extreme point, $A^{'}$ has full rank. So, even $-A^{'-1}$ has full rank. Or in other words, the n columns form a basis of the space. Hence, the vector $x_{opt} - x_0$ can be written as a linear combination of these columns, i.e.:

$$x_{opt} - x_0 = \sum_i \beta_i (-A^{'-1})^{(i)} \tag{4}$$

where $B^{(i)}$ represents the i^{th} column of a matrix B. Pre-multiplying with $A^{'}$ in the above equation, we get

$$A^{'}x_{opt} - A^{'}x_0 = \sum_i \beta_i A^{'}(-A^{'-1})^{(i)} \tag{5}$$

Note the following in the above equation:

1. As x_{opt} is a feasible point, $A^{'}x_{opt} \leqslant b^{'}$ whereas $A^{'}x_0 = b^{'}$. Hence $A^{'}x_{opt} - A^{'}x_0$ will be a vector with each component ≤ 0.
2. $A^{'}(-A^{'-1})^{(i)} \leq \mathbf{0}$ or to be more specific it has a zero at all positions except at the i^{th} row where it is -1.

These two observations imply that $\forall j \ \beta_j \geqslant 0$.

Now pre-multiply with c^T in Equation 4. We get

$$c^T x_{opt} - c^T x_0 = \sum_i \beta_i c^T (-A^{'-1})^{(i)} \tag{6}$$

We know that as $(-A^{'-1})^{(i)}$ are directions of the neighbours, $(-A^{'-1})^{(i)} = \alpha_i(x_i - x_0)$ where x_i's are the neighbours of x_0 and $\alpha_i \geq 0$. As we have stopped at x_0, $\forall x_i \ c^T x_i - c^T x_0 \leq 0$. Coupled with the fact that $\beta_j \geqslant 0$, the R.H.S. of the above equation is ≤ 0. Hence, we get

$$c^T x_{opt} - c^T x_0 \leq 0 \tag{7}$$

Comparing this with our assumption, we find that there is a contradiction. So, our assumption is wrong and x_0 is indeed an optimal point.

Lecture 12: The Simplex Algorithm III

Lecturer: *Sundar Vishwanathan* Scribe: *Prekshu Ajmera*
Computer Science & Engineering Indian Institute of Technology, Bombay

Let x_0 be the current extreme point. From previous lectures we already know that:

- The directions of the neighboring extreme points of x_0 are given by the columns of the matrix $-A'^{-1}$
- We can write $Ax_0 \leq b$ as

$$A'x_0 = b' \tag{1}$$

$$A''x_0 < b'' \tag{2}$$

  where A' consists of n linearly independent rows(hyperplanes).

In this lecture we will give the stopping condition of the Simplex algorithm and prove the correctness of the algorithm.

1 Stopping Condition

If all the neighboring extreme points of x_0 have a cost $\leq$ the cost of x_0 then x_0 is optimal and the algorithm stops at x_0.

Proof First of all note that this is *not* the same as saying that x_0 is local maximum and hence by a previously proved theorem, it is also a global maximum. This is so because we just know that the cost at x_0 is maximum as compared to its *neighbours* - not compared to a small enough *neighbourhood* around it.

First Approach

Let us consider a small neighborhood N of x_0. Also, assume that we can write any point $p \in N$ as a convex combination of all the neighboring extreme points of x_0 i.e.

$$p = \sum_{i=0}^{n} \lambda_i x_i \quad ; \quad 0 \leq \lambda_i \leq 1, \Sigma\lambda_i = 1 \tag{3}$$

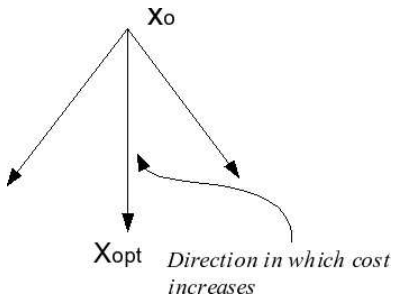
This is similar to taking a weighted average with $\lambda_i s$ be the probabilities.
Now we know that $\forall i \; c^T x_i \leq c^T x_0$. Hence,

$$c^T p = c^T(\sum_{i=0}^{n} \lambda_i x_i) = \sum_{i=0}^{n} \lambda_i c^T x_i \leq c^T x_0(\sum_{i=0}^{n} \lambda_i) = c^T x_0 \tag{4}$$

Thus, x_0 is a local maximum and consequently a global maximum.
To be shown : (i) $p = \sum \lambda_i x_i$ (ii) Starting criteria

Second Approach

Assume that x_0 is not an optimal point and there is some other point x_{opt} which is optimal. Thus, the cost increases along $x_{opt} - x_0$.



Now, x_0 is an extreme point

$\Rightarrow A^{'}$ has full rank
$\Rightarrow -A^{'-1}$ has full rank
$\Rightarrow n$ columns form a basis of the space
$\Rightarrow$ vector $x_{opt} - x_0$ can be written as a linear combination of these columns. Hence,

$$x_{opt} - x_0 = \sum_i \beta_i((-A^{'-1})^i) \tag{5}$$

Pre-multiplying with $A^{'}$ in the above equation, we get

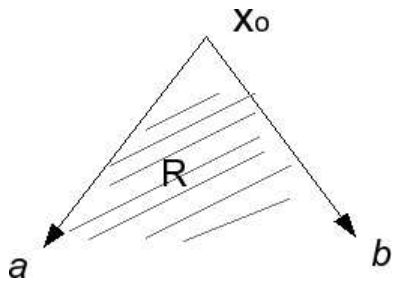
$$A^{'}x_{opt} - A^{'}x_0 = \sum_i \beta_i A^{'}((-A^{'-1})^i) \tag{6}$$

What can we say about β_i ?

Lets look at the following figure:

Any point in the region R can be written as $a\beta_1 + b\beta_2$. For feasibility, β_1 and β_2 have to be positive, otherwise we will be out of the region R ($Ax \leq b$). Thus, intutively it should be clear that $\beta_i \geqslant 0$.

More formal proof for $\beta_i \geqslant 0$ will be given in the next lecture.



Now pre-multiplying with c^T in (5), we get

$$c^T x_{opt} - c^T x_0 = \sum_i \beta_i c^T((-A'^{-1})^i) \tag{7}$$

Now, $(-A'^{-1})^i = x_i - x_0$ where x_i's are the neighboring extreme points of x_0. Since we have stopped at x_0, $\forall x_i \; c^T x_i - c^T x_0 \leq 0$. Also, $\beta_j \geqslant 0$. This implies that the R.H.S. of the above equation is $\leqslant 0$. Hence,

$$c^T x_{opt} - c^T x_0 \leqslant 0 \tag{8}$$

Thus, we find that cost decreases along $x_{opt} - x_0$ which is a contradiction. So, our assumption was wrong and x_0 is indeed an optimal point. Hence proved !

CS 435 : Linear Optimization Fall 2006

Lecture 13: Proof of correctness of Simplex Algorithm(contd) and Introduction to Duality Theorem

Lecturer: *Sundar Vishwanathan* Scribe: *Akhil Lodha*
Computer Science & Engineering Indian Institute of Technology, Bombay

Let x_0 be an extreme point. Suppose it is given by :

$$A^{'}x_0 = b^{'}, A^{''}x_0 < b^{''} \tag{1}$$

From lecture 11 we know that the neighbours of x_0 are along the columns of $-A^{'-1}$

1 Proof of correctness of Simplex algorithm

Theorem 1 *If the cost decreases along the columns of $-A^{'-1}$ then x_0 is optimal.*

Proof: The columns of $-A^{'-1}$ span R^n. Let x_{opt} be an optimal point i.e. $c^T x_{opt} \geq c^T x_0$ then we need to show that $c^T x_{opt} \leq c^T x_0$ to establish $c^T x_{opt} = c^T x_0$ and hence $x_{opt} = x_0$. Since the columns of $-A^{'-1}$ are a basis the vector $x_{opt} - x_0$ can be represented as a linear combination of them.

$$x_{opt} - x_0 = \sum \beta_j (-A^{'-1})^j \tag{2}$$

Now consider $A'(x_{opt} - x_0)$

$$A'x_{opt} - A'x_0 = \sum \beta_j A'(-A^{'-1})^j \tag{3}$$

We know that $A'x_{opt} \leq b' and A'x_0 = b'$ hence $A'(x_{opt} - x_0) \leq 0$. Also note that $A'(-A^{'-1})^j$ is an $n \times 1$ vector whose j^{th} element is -1 and remaining elements are 0. Hence

$$A'x_{opt} - A'x_0 = \begin{pmatrix} -\beta_1 \\ -\beta 2 \\ . \\ . \\ . \\ -\beta n \end{pmatrix} \tag{4}$$

$\Rightarrow \forall j \beta_j >= 0$

From the discussion above we infer that $\beta_j \geq 0$ for each j.

Now consider

$$c^T x_{opt} - c^T x_0 = \sum \beta_j c^T (-A^{'-1})^j \tag{5}$$

Since the cost decreases along the columns of $-A^{'-1}$ we have $c^T(-A^{'-1})^j \leq 0$ and since $\beta_j \geq 0$ we conclude that $\sum \beta_j c^T (-A^{'-1})^j \leq 0$

Hence $c^T x_{opt} \leq c^T x_0$ but we know that $c^T x_{opt} \geq c^T x_0$ and $c^T x_{opt} = c^T x_0$. □

Note: Using the above theorem we can now state that when the Simplex Algorithm terminates it gives us an optimal solution.

2 Introduction to Duality theorem

Let x_0 be an optimal point. Using the termination condition of Simplex Algorithm we know that cost decreases along the columns of $-A^{'-1}$.In other words,

$$c^T(-A^{'-1}) = (\gamma_1, \gamma_2, ..., \gamma_n), \gamma_i <= 0 \tag{6}$$

or

$$c^T(A^{'-1}) = (y_1, y_2, ..., y_n), y_i >= 0 \tag{7}$$

$$c^T(A^{'-1}) = y^T, y_i >= 0 \tag{8}$$

$$c^T(A^{'-1}A^{'}) = y^T A^{'} \tag{9}$$

$$c^T = y^T A^{'} \tag{10}$$

We observe that at the optimal point the cost vector can be written as a *non-negative* linear combination of the rows of A'. This means that x_0 is optimal iff x_0 is feasible and the cost can be written as a non-negative linear combination of the rows of A'.

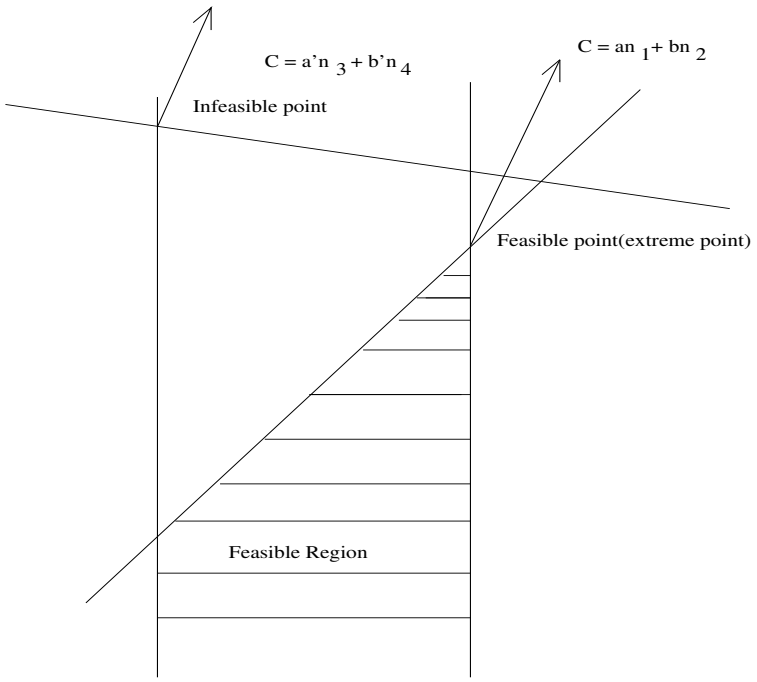


Figure 1: $a, a', b, b' \geq 0$. Cost can be written as a positive linear combination of normals to the hyperplanes.

The rows of A' are also the direction normals to the respective hyperplanes. So a restatement of the above is as follows. Suppose x_0 is an extreme point given by the intersection of n linearly independent hyperplanes then the cost vector can be written as a non-negative linear combination of the normals to these hyperplanes.

Now consider all the points (not necessarily feasible) given by the intersection of n linearly independent hyperplanes where the cost vector can be written as a positive linear combination of the normals. We will show that among such points only the feasible point will have the lowest cost.

Consider the feasible point x_0 and any other point say x satisfying the above requirements, then $x - x_0$ can be written as a positive linear combination of the columns of A' where

$$A'x = b', A^{''}x < b^{''} \tag{11}$$

Note that the cost decreases along the columns of $-A^{'-1}$. Following the steps of the proof of the previous theorem one can show that $x - x_0$ can be written as a non negative linear combination of the columns of $-A^{'-1}$. Since the cost decreases along the columns of $-A^{'-1}$, the cost at x is at least the cost at x_0.

We also note that at such points the cost is $c^T x = y^T A'x = y^T b'$.

This motivates the definition of the following LP called the dual:

$$minimize : y^T b \tag{12}$$

$$A^T y = c \tag{13}$$

$$y >= 0 \tag{14}$$

CS 435 : Linear Optimization — Fall 2006

Lecture 13: Proof of correctness of the Simplex algorithm and introduction to duality

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Shashvat Rai*
Indian Institute of Technology, Bombay

From the previous lectures, we know the following fact that, if x_0 is an extreme point given by

$$A^{'}x_0 = b^{'} \tag{1}$$

$$A^{''}x_0 < b^{''} \tag{2}$$

then the neighbours of x_0 are along the columns of $-A^{'-1}$.

Theorem 1 *If the cost decreases along the columns of $-A^{'-1}$, then x_0 is optimal.*

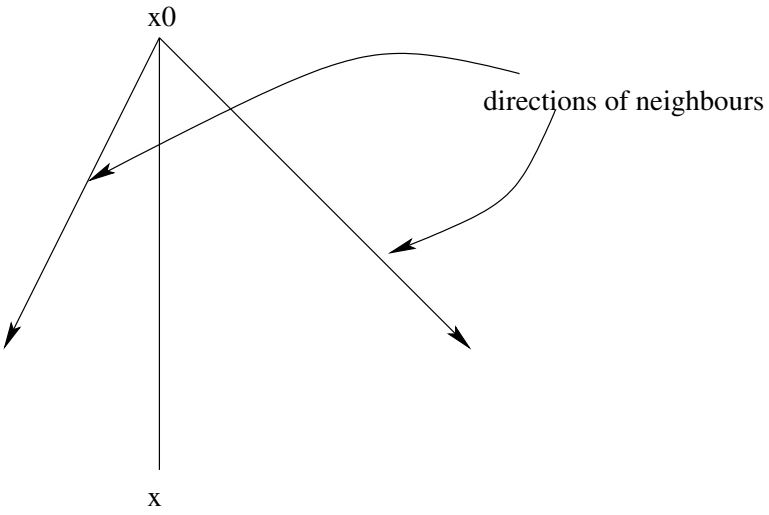


Figure 1: Representation of (x-x0)

Proof: As $A^{'}$ has full rank, $-A^{'-1}$ also has full rank. Thus the columns of $-A^{'-1}$ form a basis of $\Re^n$. Hence any vector can be written as a linear combinations of these n columns.

$$x - x_0 = \sum(\beta_j)(-A^{'-1})^j \tag{3}$$

Premultiplication with $A^{'}$

$$A^{'}x - A^{'}x_0 = \sum \beta_j A^{'}(-A^{'-1})^j \tag{4}$$

Since x is feasible $A^{'}x <= b^{'}$. Also $A^{'}x_0 = b^{'}$. Thus the L.H.S. of equation 4 is a vector each of whose components is at most zero. The R.H.S. is the vector $(\beta_1, \beta_2, ..., \beta_n)^T$. This implies that $\beta_j >= 0$ for all j. Hence $x - x_0 = \sum \beta_j(-A^{'-1})^j$ where $\beta_j >= 0$. Premultiplication with c^T gives

$$c^Tx - c^Tx_0 = \sum \beta_j c^T(-A^{'-1})^j \tag{5}$$

Since $\beta_j >= 0$ and $c^T(-A^{'-1})^j <= 0$ for all j, therefore $c^Tx <= c^Tx_0$. Hence x_0 is optimal. □

Discussion:
Let x_0 be the optimal point and

$$A^{'}x_0 = b^{'} \tag{6}$$

and

$$A^{''}x_0 < b^{''} \tag{7}$$

The cost decreases along the columns of $-A^{'-1}$. This can be written as

$$c^T A^{'-1} = (y_1, y_2, ..., y_n) \text{ where } y_i >= 0, \forall i \tag{8}$$

This means that the cost vector is a positive linear combination of the normals to the hyperplanes.
Consider all points (not necessarily feasible) given by n linearly independent hyperplanes, where the

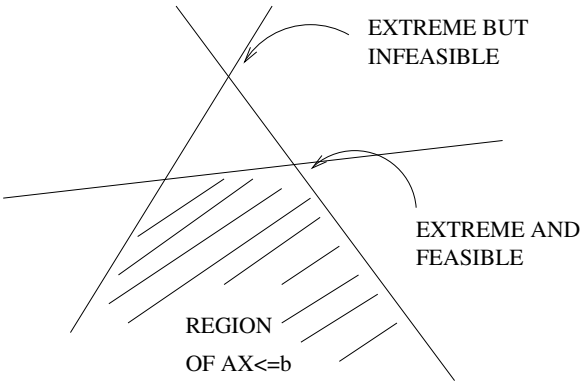


Figure 2: 2-D example of extreme but infeasible points

cost vector can be written as a positive linear combination of the normals. These points are the candidate maximums.

Take any such point x say
Then $c^T x >= c^T x_0$, where x_0 is the optimal point. Clearly this can be written as another linear program in the following manner - As $c^T x = y^T A^{'} x = y^T b^{'}$, thus dual linear program is

$$\begin{gathered} \min\ y^T b^{'} \\ A^T y = c \\ y >= 0 \end{gathered}$$

CS 435 : Linear Optimization Fall 2006

Lecture 9: Duality theorem, how to solve dual using solver for primal

Lecturer: *Sundar Vishwanathan* Scribe: *Lapsy Garg*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Duality

Theoram(Duality): If both primal and dual are feasible and bounded, then their optimums must be equal.
Proof:
If x_0 and y_o are optimum then $c^T x_o = y_o^T b$
For any feasible x and y,

$$c^T x = y^T A x \leq y^T b$$
$$\mathrm{c}^T x \leq y^T b$$

Let x_o be optimum, using the previos discussion

$$c^T x_0 = y^T A x_0 = y^T b'$$

Extend y^T by putting all other y's as 0(zero) to get $c^T x_0 = y^T b$

2 Rephrasing

$$min\ y^T b$$
$$\mathrm{A}^T y = c$$
$$\mathrm{y}_i \geq 0$$

can be rephrased as

$$max\ -b^T y$$
$$\mathrm{A}^T y \leq c$$
$$\mathrm{A}^T y \geq c$$
$$\mathrm{y} \geq 0$$

For reverse part

$$max\ c^T x$$
$$\mathrm{Ax} \leq b$$

can be rephrased as

$$min\ -x^T c$$
$$\mathrm{Ax+x'} = \mathrm{b}$$
$$\mathrm{x'} \geq 0$$

CS 435 : Linear Optimization — Fall 2006

Lecture 15: Complementary slackness, infeasible primal ⇔ unbounded dual, infeasible dual ⇔ unbounded primal

Lecturer: *Sundar Vishwanathan* — Scribe: *Aman Parnami*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Revision

Following were done in previous lecture

- Primal and Dual

Primal	Dual
$\max c^T x$	$\min y^T b$
$Ax \leq b$	$A^T y = c$
	$y \geq 0$

- Proof of existence of a dual for a feasible and bounded primal.
- Theorem 1 (Duality Theorem) *If both the primal and the dual of an LP are feasible, then their optimum value coincide i.e*

$$\boxed{c^T x_0 = y_0^T b \text{ , where } x_0, y_0 \text{ are optimals for Primal and Dual resp.}}$$

2 Optimality Condition

Consider an x_0 such that $\{A\}_{m\times n}\{x_0\}_{n\times 1} \leq \{b\}_{m\times 1}$ and a $\{y_0\}_{m\times 1}$ such that $(y_0)_i \geq 0$ and $A^T y_0 = c$

Then the following statements are equivalent.

1. x_0 and y_0 are optimal $\Leftrightarrow \boxed{c^T x_0 = y_0^T b}$
2. x_0 and y_0 are optimal $\Leftrightarrow \boxed{(y_0)_i > 0 \Rightarrow A_i x_0 = b_i}$

Statement 2 is called **Complementary Slackness**. Geometrically, it means hyperplanes corresponding to (+)ve $(y_0)_i$'s intersect at x_0. In the other case, $(y_0)_i = 0 \Rightarrow A_i x_0 < b_i$ i.e A_i corresponds to a hyperplane which does not passes through x_0.

PROOF: We use 1 as definition of Optimality.

- **2 ⇒ 1**

$$\begin{aligned}
y_0^T b &= \sum_{j=1}^{m} y_{0j} b_j \\
&= \sum_{j=1}^{m} y_{0j}(A_j x_0) \qquad (\text{using } A_i x_0 = b_i \ ) \\
&= \sum_{j=1}^{m} y_{0j} \left(\sum_{i=1}^{n} A_{ji} x_{0i} \right) \\
&= \sum_{i=1}^{n} x_{0i} \left(\sum_{j=1}^{m} A_{ji} y_{0j} \right) \qquad (\text{using } A^T y_0 = c \ ). \\
&= x_0^T c \\
&= c^T x_0
\end{aligned}$$

- **1 ⇒ 2**

$$\begin{aligned}
y_0^T b &= x_0^T c \\
&= \sum_{i=1}^{n} x_{0i} \left(\sum_{j=1}^{m} A_{ji} y_{0j} \right) \qquad (\text{using } A^T y_0 = c \ ) \\
&= \sum_{j=1}^{m} y_{0j} \left(\sum_{i=1}^{n} A_{ji} x_{0i} \right) \\
&= \sum_{j=1}^{\vdots} y_{0j}(A_j x_0) \\
&\leq \sum_{j=1}^{m} y_{0j} b_j \qquad (\text{using } y_{0j} \geq 0 \text{ and } A_j x_0 \leq b_j \ ) \\
&= y_0^T b
\end{aligned}$$

⇒ for non-zero (+)ve $(y_0)_i$'s, $A_i x_0 = b_i$ otherwise inequality remains leading to contradiction.

□

3 Next Class

Following questions were raised at the end of class and will be fully discussed in next lecture :

- What if any of the Primal or Dual is **infeasible** ?
  Infeasibility *means no point satisfies all the constraints.*
- What if cost is **unbounded** ?
  Unbounded *here means no optimum value exists or that there are feasible points with arbitrary large cost.*
  Ex. max -x,$x \leq 10$
  Here optimum value for cost = -x can't be found with given constraints.

CS 435 : Linear Optimization — Fall 2006

Lecture 15: Complementary slackness, Duality Theeorem

Lecturer: *Sundar Vishwanathan* — Scribe: *Jay Prakash*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 The Duality Theorem

Primal	Dual
$\max c^T x$	$\min y^T b$
$Ax \leq b$	$A^T y = c$
	$y \geq 0$

Theorem 1 *If both the primal and the dual of an LP are feasible, then their optimum values coincide*

If primal LP is feasible and bounded, there is some optimal point x_o. As discussed in the last class we can construct a $y \geq 0$ such that $A^T y = c$ and $y^T b = c^T x$. This means that the dual is feasible and the optimum values coincide.

2 Optimality Condition

Consider an x_0 such that $Ax_0 \leq b$ and a y_0 such that $y_0 \geq 0$ and $A^T y_0 = c$. Then the statements

1. x_0 and y_0 are optimal respectively for the primal and dual if and only if $c^T x_0 = y_0^T b$.
2. x_0 and y_0 are optimal respectively for the primal and dual if and only if $(y_0)_i > 0 \Rightarrow A_i x_0 = b_i$.

are equivalent.
Proof: We prove 1 using 2

$$\begin{aligned}
y_0^T b &= \sum_{j=1}^{m} y_{0j} b_j \\
&= \sum_{j=1}^{m} y_{0j} (A_j x_0) \qquad \text{(using 2)} \\
&= \sum_{j=1}^{m} y_{0j} \left(\sum_{i=1}^{n} A_{ji} x_{0i} \right) \\
&= \sum_{i=1}^{n} x_{0i} \left(\sum_{j=1}^{m} A_{ji} y_{0j} \right) \\
&= x_0^T c \qquad \text{(using } A^T y_0 = c \text{)} \\
&= c^T x_0
\end{aligned}$$

Proving 2 using 1

$$\begin{aligned} c^T x_0 &= x_0^T c \\ &= \sum_{i=1}^{n} x_{0i} \left(\sum_{j=1}^{m} A_{ji} y_{0j} \right) \\ &= \sum_{j=1}^{m} y_{0j} \left(\sum_{i=1}^{n} A_{ji} x_{0i} \right) \\ &= \sum_{j=1}^{m} y_{0j} (A_j x_0) \\ &\leq \sum_{j=1}^{m} y_{0j} b_j \qquad \text{(using } y_{0j} \geq 0 \text{ and } A_j x_0 \leq b \text{)} \\ &= y_0^T b \end{aligned}$$

But we know that $c^T x_0 = y_0^T b$, hence, $y_0 > 0 \Rightarrow A_i x_0 = b_i$ This condition is called complementary slackness. □

3 Infeasibility and Unboundedness

Every LP problem is either feasible or infeasible. Now, the feasible problems either have a solution or are unbounded.
Infeasibility implies that there is no solution to Ax ≤ b. Example of an unbounded LP is

$$\begin{aligned} &\max\ (-x) \\ &\quad x \leq 10. \end{aligned}$$

Lecture 16: Another way of looking at duality

Lecturer: *Sundar Vishwanathan* Scribe: *TAs*
Computer Science & Engineering Indian Institute of Technology, Bombay

In this lecture we will see another way of interpreting duality. However, before that, let us clarify some doubts on duality that few students seemed to have in the last class.

Let us consider the following primal-dual pair.

$$\begin{aligned} \text{P}: \quad \max\ & c^T x & \qquad \text{D}: \quad \min\ & y^T b \\ \text{s.t. } & Ax \leq b & \text{s.t. } & A^T y = c \\ & & & y \geq 0 \end{aligned}$$

All that the duality theorem states is the following.

- "P is feasible and has a finite maximum" $\Rightarrow$ "D is feasible and the two optimum values coincide".
- "P is infeasible and D is feasible" $\Rightarrow$ "D is unbounded".
- "P feasible and unbounded" $\Rightarrow$ "D is infeasible".

Solve the following exercises.

Exercise 1 Construct a P-D pair where both are infeasible.

Exercise 2 Show that dual of D is actually P.

Exercise 3 Comment about the feasible regions of P & D.

1 Another way of looking at duality

Let us consider the following LP problem.

$$\begin{aligned} \max\ & 14x_1 + 7x_2 + 22x_3 + 10x_4 & & \\ \text{s.t. } & 10x_1 + 3x_2 + 10x_3 + 7x_4 \leq 20 & & (1) \\ & 3x_1 - 12x_2 - 13x_3 + 14x_4 \leq 35 & & (2) \\ & 4x_1 + 4x_2 + 12x_3 + 3x_4 \leq 4 & & (3) \end{aligned}$$

If we look at the above equations closely we can see that an upper bound of the objective is 24. This is because if we simply add (1) and (3), we get $14x_1 + 7x_2 + 22x_3 + 10x_4 \leq 24$. We could have multiplied the above equations by any non-negative factors (if we multiply by negative factors, the direction of the inequalities changes which we do not want) and then added to get an upper bound of the objective. This observation gives us another way of solving the LP.

Let us have the following general LP. Say it the primal (P).

$$\begin{aligned} \max\ & c_1x_1 + c_2x_2 + \cdots + c_nx_n & & \\ \text{s.t. } & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 & \} \times y_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 & \} \times y_2 \\ & \vdots & \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m & \} \times y_m \end{aligned}$$

Suppose we are able to find out non-negative multipliers $y_1, y_2, \cdots, y_m$ such that, when the equations are multiplied by the respective multipliers and added together, we get the objective functions on the left hand side. That is,

$$\begin{aligned} a_{11}y_1 + a_{21}y_2 + \cdots + a_{m1}y_m &= c_1 \\ a_{12}y_1 + a_{22}y_2 + \cdots + a_{m2}y_m &= c_2 \\ &\vdots \\ a_{1n}y_1 + a_{2n}y_2 + \cdots + a_{mn}y_m &= c_n \\ y_i &\geq 0 \qquad \forall i = 1..m \end{aligned}$$

Then the sum of the right hand sides i.e. $b_1y_1 + b_2y_2 + \cdots + b_my_m$ gives an upper bound of the primal. Now consider all possible sets of multipliers satisfying the above requirement. Consider the upper bound given by each such set. The least among the upper bounds gives the optimum of the primal. Hence, the optimum objective of the primal is the same as the optimum objective of the following LP. Say it the dual (D).

$$\begin{aligned} \min\ & b_1y_1 + b_2y_2 + \cdots + b_my_m \\ \text{s.t. } & a_{11}y_1 + a_{21}y_2 + \cdots + a_{m1}y_m = c_1 \\ & a_{12}y_1 + a_{22}y_2 + \cdots + a_{m2}y_m = c_2 \\ & \vdots \\ & a_{1n}y_1 + a_{2n}y_2 + \cdots + a_{mn}y_m = c_n \\ & y_i \geq 0 \qquad \forall i = 1..m \end{aligned}$$

Let us explore all cases of the duality theorem from this new perspective.

1. "P is feasible and has a finite maximum". It easily follows from the above discussion that "D is feasible and the two optimum values coincide".

2. "P feasible and unbounded". In this case, we wont be able to find a set of multipliers with the requirement stated in the above discussion. The reason is as follows. Suppose we are able to find some set of multipliers $y_1^*, y_2^*, \cdots, y_m^*$. Then the objective of P should be bounded from above by $b_1y_1^* + b_2y_2^* + \cdots + b_my_m^* = U$, some finite quantity. But, that contradicts the fact that P is unbounded. Hence, "D is infeasible".

3. "P is infeasible and D is feasible". Then D is unbounded. The reason is as follows. Suppose, D is bounded. Then D is both feasible and bounded. Again, we have seen that the dual of D is P. Now, taking D as the primal, by the statement already proved in part 1, P is feasible. But that contradicts the fact that P is infeasible. Hence "D is unbounded".

CS 435 : Linear Optimization | Fall 2006

Lecture 17: Matching in Bipartite Graphs

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Hardeep Singh Guru*
Indian Institute of Technology, Bombay

1 Matching

Definition 1 *A* Matching in a graph is a set of edges, no two of which share an end-point.

Let us look at an example. In the following graph the marked edges form a matching as no two share a common vertex.

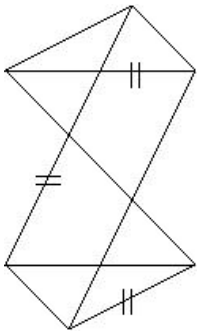


Figure 1: Matching

2 Algorithm

We now look at an algorithm which takes as input a bipartite graph and outputs a matching of maximum size. The basic idea of the algorithm lies in iteratively increasing the size of the matching by one at every stage until a matching of maximum size is obtained.

Let M be the current set of edges in the matching and let M_0 be any maximum matching.

Now, consider the graph induced by $M \oplus M_0$. We can clearly see that this graph would consist of connected components with the degree of each vertex in these being either zero, one or two. The degree would be zero for the end points of the edges which are matched in both M and M_0 or in neither. The degree will be one in the case when there is a matching edge incident on that vertex in only one of the matching and two when two edges incident on the vertex are matched, one in M and the other in M_0.

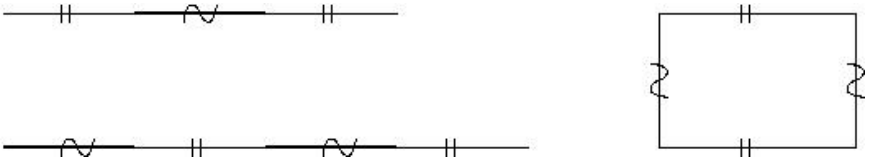


Figure 2: Connected Components

It can also be seen that the connected components of $M \oplus M_0$ are either paths or cycles.

What can we say about the cycles? We can see that the cycles would necessarily need to be of even length.

Hence, we see that the connected components are either paths or cycles of even length.

Now, let us consider a matching M which has size exactly one less the size of the maximum matching M_0.

CLAIM 1 *$M \oplus M_0$ will contain exactly one path of odd length.*

Suppose that there are two odd length paths. They can be either one of the following types.

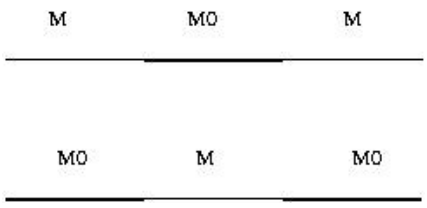


Figure 3: Odd Length Paths

Consider the path of first type where the first edge in the path belongs to M. This cannot be the case since we can otherwise switch the matched edges in M and M_0 which belong to this path and hence increase the size of M_0 by one.

Also, the paths of type two starting with M_0 cannot be more than one in number because then we can switch the matched edges belonging to M and M_0 in all these paths and hence get a matching of size greater then the size of the matching in M_0 which is a contradiction since M_0 has been assumed to be the maximum matching.

Similarly one can prove the following claim.

CLAIM 2 *If M is not a maximum there is a path in the graph which starts and ends at vertices that are unmatched (i.e., do not have any edge of M incident on them) and alternate edges in the path are from M. That is the first, third, ..., last edges are not in M while the second, fourth, ... are in M.*

For a proof note that such a path exists in the graph induced by $M \oplus M_0$. All other connected components in $M \oplus M_0$ have at least as many edges from M as from M_0. Such an path is called an augmenting path. Suppose we find an augmenting path, then we can increase the size of M by "switching" the matched and the unmatched edges. That is remove the matched edges in the path from M and introduce in the unmatched edges on the path in M_0.

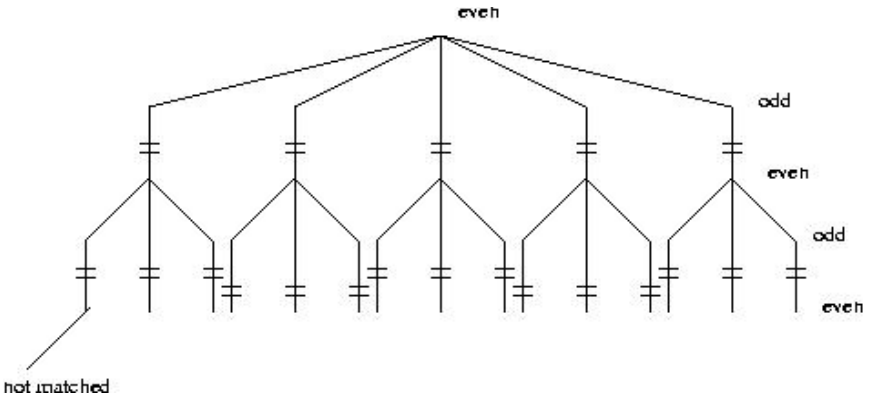


Figure 4: Augmenting Path

The broad outline of our algorithm is now clear. In each iteration, find an augmenting path and increase the size of the matching. We next see how to find such an augmenting path.

We start with a vertex which has no matched edges and label it as “even”. The other end of the edges from this vertex are labeled as “odd” as shown in the figure. At the “odd” level we go through only the matched edges and at “even” level we go through all the edges. Suppose we find an odd vertex that is unmatched, we have found the path that we are looking for which consists of alternating matched and unmatched edges and starts and ends with an unmatched vertex. Hence we can switch the matched and unmatched edges along this path and increase the size of the matching by one. We iteratively repeat this procedure until no augmenting path can be found in the graph. The matching thus obtained will be of maximum size.

Also, we can ignore back edges since edges from an “even” vertex will be only to an “odd” vertex (the graph is bipartite).

Next time we will prove that the algorithm is indeed correct.

CS 435 : Linear Optimization — Fall 2006

Lecture 17: Matching in Bipartite Graphs

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Ramesh Nidadavolu*
Indian Institute of Technology, Bombay

A set of disjoint edges in G is called a *matching*. *Maximum matching* in a graph G is a *Matching* of maximum size.

First we discuss a few details. If M is *matching* and M_0 is a *maximum matching* then $M\Delta M_0$[1] has connected components which are either even cycles or simple paths. Also in a odd path the first edge in the path cannot be an edge from the matching M, else we could interchange the M_0 edges and the M edges in this path and increase the size of the matching M_0 by 1.

Another interesting thing to note is that if $A\Delta B = C$ then $A\Delta C = B$. (Proof is simple. Draw a venn diagram to see that it should hold)

We also state that $M\Delta M_0$ has atleast one component which is a path of odd length.(Proof is easy. Solve the exercises below to figure out the proof)

Exercise 1 If M is a matching of size $|M_0| - 1$ then there exists an odd length path in $M\Delta M_0$.

Exercise 2 For any matching M there is atleast one odd length path in $M\Delta M_0$. (Hint: If all components would be even then $|M| = |M_0|$?)

Also as we pointed out earlier any odd length path has to start with an edge from M_0

Now given a graph G and a matching M we define a augumenting path as a path in G whose start and end edges are not in M and alternate edges are in M along the path.(so an augumenting path would have odd number of edges with more unmatched edges than matched edges.)

Now we prove that if a matching M is not a maximum matching there must exist an augumenting path wrt to M. (Actually a stronger result is proven below)

1 Main result

We prove the following result: *A matching M is maximum iff there does not exist any augumenting path in G.*

If the matching is maximum and there exists an augumenting path P then $M\Delta P$ would be a matching of greater size.Contradicton. Also If the matching is not maximum then there must exist an odd length path given by $M\Delta M_0$ (as proven earlier).

Thus proven.

2 Why this result?

The idea is that in searching for a matching that is maximum we can start of with some arbitary matching M and then look for augementing paths and then construct a new matching $M\Delta P$ which would be of a size greater than the matching we started of with. Thus repeatedly looking for augumenting paths and incrementing our matching size we hope to end up with a maximum matching. More on this in the next lecture.

[1] $A\Delta B = (A - B) \cup (B - A)$

CS 435 : Linear Optimization — Fall 2006

Lecture 18: Review of graph algorithms and extending it for general graphs

Lecturer: *Sundar Vishwanathan* — Scribe: *Girish Sahani*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Review of Previous lecture:

Our objective is to find a matching M of maximum size in a bipartite graph G.
Suppose edge set of the graph is E. Then a path is called an *alternating path* if its edges are alternately in M and $E - M$.
An *augmenting path* is an alternating path whose start and end vertices are free (unmatched).
To find an augmenting path in G wrt a matching M, we will use a Modified BFS algorithm. This algorithm finds an augmenting path and increments the size of current matching M by inverting M. This gives us a matching M' with $|M'| = |M| + 1$. This algorithm terminates when no augmenting path can be found.

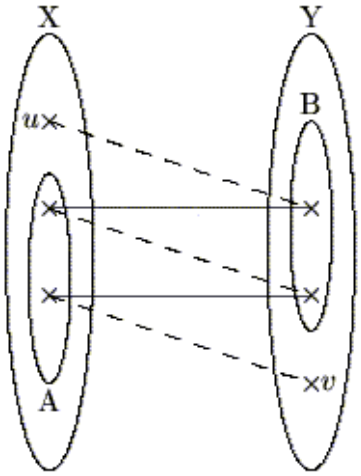


Figure 1: Alternating path in a Bipartite Graph (broken lines indicate non-matched edges)

2 Proof of correctness of modified BFS algorithm

THEOREM 1 *M is maximum if and only if there exists no augmenting path in G.*

PROOF: Proved in previous lecture. □

Hence if there exists an augmenting path P from u, then applying modified BFS with u as root node must find an augmenting path. In fact, the alogrithm finds the shortest augmenting path P from u. Let $\{u_0, u_1, ...u_{k-1}\}$ be the order of vertices in P, $u = u_0$.

LEMMA 1 *All alternating paths starting from u are traversed in the modified BFS (assuming we do not stop even if we obtain an augmenting path in the middle of BFS) in order.*

PROOF: By induction on length of path. Let k represent length of path.
Base Case: $k = 0, k = 1$
This is trivially true. In the first iteration, algorithm marks all non-matching edges incident on u, and in next iteration one matching edge from each of the vertices in level 1 is marked. Now as there is atmost one matching edge per vertex, we have covered all alternating paths starting from u and of length 1 and 2.

Inductive Step: Assuming the induction hypothesis holds for $k = l$, consider $k = l + 1$.
if $l = 2 * m$:
Modified BFS algorithm will mark one matching edge (if any) incident on each vertex in level $2 * (m - 1)$, and as all alternating paths of length $2 * m - 1$ have been marked (induction hypothesis), and as there is atmost one matching edge per vertex ⇒ all alternating paths starting from u and of length $l = 2 * m$ are marked.
if $l = 2 * m + 1$:
Modified BFS Algorithm will mark all unmatched edges from each vertex in level $2 * m$, and as all alternating paths of length 2*m have been marked ⇒ all alternating paths starting from length $l = 2 * m + 1$ are marked.
Thus we have proved the lemma by induction.

□

We use this lemma to prove correctness of modified BFS algorithm:
Since augmenting path is also an alternating path, from above lemma the algorithm guarantees to find an augmenting path (if any). As the algorithm finds alternating paths with increasing length, shortest augmenting path will be found first. Thus the modified BFS algorithm is correct.
Time complexity of this algorithm = $O(m * n^2)$, where $|V| = n, |E| = m$.

3 Extending the modified BFS algorithm to general graphs

Non-bipartite graphs can also contain even-even edges (as shown in the figure above). This will result in odd length cycles.
Hence if we apply modified BFS on non-bipartite graphs, the algorithm might not find an augmenting path. Such an example is shown in the Fig. 2. The augmenting path is shown in dark.
This is a non trivial polynomial time algorithm and it was solved by Jack Edmonds. A solution to this problem is to shrink the odd-cycles (blossoms) to a single vertex.
This will be discussed in detail in the next lecture.

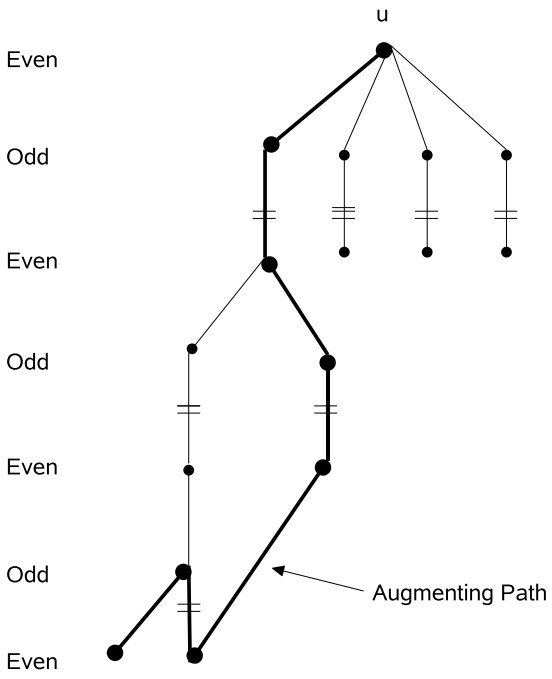


Figure 2: Modified BFS on General graph

CS 435 : Linear Optimization — Fall 2006

Lecture 18: Review of graph algorithms (contd.)

Lecturer: *Sundar Vishwanathan* — Scribe: *Vipul Shingde*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Recap of previous lecture

In previous lectures we studied following definitions and theorem:

Definition 1 *A path is alternating w.r.t. a matching M if its edges alternate between those in M and those not in M.*

Definition 2 *An augmenting path is an alternating path that starts from and ends on free (unmatched) vertices.*

Theorem 1 *A matching M is maximum if and only if it does not contain any augmenting path.*

Algorithm for maximum matching: We start with a single edge in the matching M. In each iteration we first find an augmenting path P. Then we exchange the matched and unmatched edges in P to get a matching of one greater size. We keep doing this till a matching with no augmenting path is found. Theorem 1 guarantees that this matching is maximum.

Modified BFS Algorithm
An augmenting path in G w.r.t. a matching M, is found using the following algorithm:

Let L be set of unmatched vertices in G w.r.t. M.
For each $u \in L$
1. Start BFS with root node u.
2. Till BFS is complete
   Let BFS be currently at vertex v
   if v is at even depth
      The children of v in the BFS tree will be unvisited vertices along unmatched edges incident on v.
   else
      if v is unmatched
         Augmenting path $u \rightarrow v$ found.
      else
         The child of v in the BFS tree will be an unvisited vertex along the matched edge incident on v.

2 Proof of correctness of modified BFS algorithm

If there exists an augmenting path P from u, then applying modified BFS with u as root node must find an augmenting path for the algorithm to be correct. In fact, the given alogrithm finds the shortest augmenting path P from u.
Let $\{u_0, u_1, ...u_{k-1}\}$ be the order of vertices in a shortest augmenting path P, with $u = u_0$ as one end-point.

Consider the following lemma

Lemma 2 *The vertex u_i will appear in the i^{th} level of the BFS tree.*

PROOF: By induction on i.

Base Case: $i = 0$
Trivially true from the algorithm.

Inductive Step: Assuming the induction hypothesis holds for $i = l$, consider $i = l + 1$.
if $l = 2 * m + 1$
Modified BFS algorithm will mark one matching edge(if any) incident on each vertex in even levels. Since (u_l, u_{l+1}) is a matched edge, u_{l+1} will appear in level $l + 1$ unless it has already appeared. If it has already appeared, then we can easily show that P will not be the shortest augmenting path.
else if $l = 2 * m$
Modified BFS Algorithm will mark all unmatched edges from each vertex in level $2 * m$, and as all alternating paths of length 2*m have been marked $\Rightarrow$ all alternating paths starting from length $l = 2 * m + 1$ are marked.

□

3 Time Complexity

Consider graph $G(V, E)$ and let $|V| = n$ and $|E| = m$.
Time complexity = Number of iterations times the time to find augmenting path in a matching.
Since the matching increases in size in every iteration, the number of iteration is bounded by n. In each iteration we may have to perform $O(n)$ searches (from unmatched vertices) before finding an augmenting path. A single run of modified BFS takes $O(m)$ time.

Hence time complexity = $O(m * n^2)$.

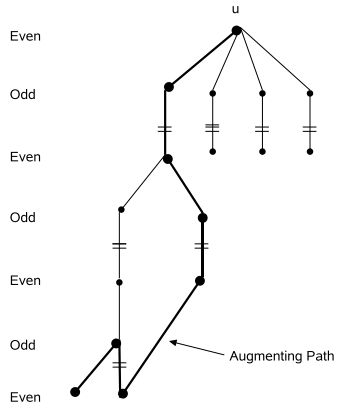


Figure 1: Modified BFS on General graph

[1]With some modifications a better algorithm with run-time of $O(n^{3/2} * m)$ can be obtained

4 Extending the algorithm for general graphs

The above modified BFS algorithm works correctly for bipartite graphs. For general graphs(which can contain odd cycles), the algorithm needs to be modified. Non-bipartite graphs can contain even-even edges(as shown in the figure above). Hence if we apply modified BFS on non-bipartite graphs, it might be possible that the alogrithm might not find an augmenting path. Such an example is shown in the Fig. 1. The augmenting path is shown in dark.
A solution to this problem is to shrink the odd-cycles to a vertex. This approach will be discussed in detail in next lecture.

CS 435 : Linear Optimization — Fall 2006

Lecture 19: Nonbipartite Matching

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Mayank Singhal*
Indian Institute of Technology, Bombay

1 Recap

Our goal is to find a matching M of maximum size given a graph G = (V,E). We know that M is maximum iff there is no augmenting path w.r.t. M in G. And in last lecture we saw how to find a matching of max size for a bipartite graph.Remember there may exist more than one such matching but our goal is to find any one of them. To find an augmenting path we used a modified BFS starting from unmatched vertices. And whenever we found one we saw we can increase the size of M by one.

2 Nonbipartite Matching

The problem in case of non-bipartite graphs is the existence of blossoms.

Definition 1 *A blossom is a cycle of length 2k+1 with k edges matched . The base of the blossom is the vertex with both incident edges unmatched.See figure 1.*

There are no blossoms in bipartite graphs because blossoms have odd no of edges and there does not exist a cycle with odd no of edges in a bipartite graph.In case of a nonbipartite graph whenever we come across a blossom we reduce it to a single vertex. And any edge that does not belong to blossom but is incident on one of its vertices is now incident on this new vertex formed by shrinking the blossom.Now to prove the validity of this operation we need to prove the following lemma.

Lemma 1 *Let M be a matching. let B be a blossom such that the base is unmatched in a graph G. Let $G^{'}$ be the graph obtained by shrinking B. Let $M^{'}$ be the edges of M outside B. There is an augumenting path in G w.r.t M iff there exists an augumenting path in $G^{'}$ w.r.t $M^{'}$.*

Proof:
Part 1: Given an augumenting path in G w.r.t M to show that there exist an augumenting path in $G^{'}$ w.r.t $M^{'}$.

If the augmenting path in G and the blossom had no edge in common then the same augmenting path exists in $G^{'}$ w.r.t $M^{'}$. Now lets look at the case where the augmenting path in G and the blossom have common edges. Note that this path has one end-point outside the blossom. Consider the first time the path enters the blossom. This will be through an unmatched edge. This part yields an augmenting path in $G^{'}$.

Part 2: Given an augmenting path in $G^{'}$ w.r.t $M^{'}$ to show that there exist an augmenting path in G w.r.t M.

Consider an augmenting path in $G^{'}$ w.r.t. $M^{'}$. If this path does not intersect the shrunk vertex (say v_b) then the same path is an augmenting path in G. Otherwise this path has to end at v_b. Extend this to an augmenting path in G by moving in the "appropriate" direction along the blossom till the base.

□

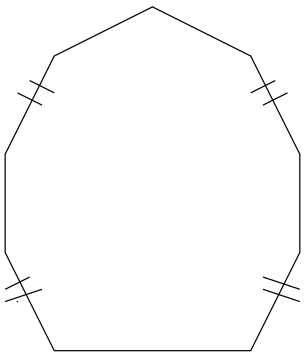


Figure 1: Blossom

CS 435 : Linear Optimization — Fall 2006

Lecture 19: Nonbipartite Matching

Lecturer: *Sundar Vishwanathan* — Scribe: *Neha Singh*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Recap

In last class we had seen how to find the matching of maximum size in a bipartite graph. This is done by starting with any single edge as a matching and then iteratively finding augmenting paths and increasing the size of the matching by one at each step until there is no augmenting path in the graph w.r.t that matching. We had proved the following theorems last time:

Theorem 1 *The matching M of the graph G is maximum iff there is no augmenting path in G w.r.t M.*

Theorem 2 *For bipartite graphs, if there exists an augmenting path in the graph G then the we can find it by using the modified bfs algorithm.*

Now let us proceed to finding maximum matching in a non-bipartite graph.

2 Nonbipartite Matching

Till now we have seen how to find the maximum matching for a bipartite graph. The same algorithm for finding an augmenting path in a bipartite graph will be extended for non-bipartite graphs. However modifications need to be made to the algorithm.
As before, we start with an unmatched even vertex, find all its neighboring odd vertices. If any of these odd vertices were unmatched then we have found an augmenting path else we continue from these odd vertices by following the matched edges incident on them. This yields a new set of even vertices and so on. In case of the a bipartite graph edges can be only between even and odd vertices. Thus while following the path, we keep alternating between even and odd vertices. However in the case of a non-bipartite graph, we can have odd-odd and even-even edges also which will give rise to odd circuits in the graph. We consider the problems arising due to even-even edges only, as in the modified bfs algorithm, after reaching an odd vertex, we follow the matched edge and there can be only one matched edge incident on a vertex.

For example consider the figure below. Here there is an even-even edge between v_1 and v_2. We observe that v_1 and v_2 have a common ancestor which has to be necessarily even. This is because in our modified bfs algorithm, we branch off at only the even vertices. At odd vertices, we only follow the matched edge. If we come across an even-even edge between two vertices while searching for augmenting path, the graph must have an odd cycle where the even-even edge is a part of the odd cycle. In the modified algorithm for finding augmenting paths in non-bipartite graphs, whenever we come across such an odd cycle, we shrink it to a single vertex (say v_o) such that all the vertices adjacent to any of the vertex in the cycle are now adjacent to v_o. Such an odd length cycle having a matched and unmatched edges incident on all but one vertex (the base where both incident edges are unmatched) is called a blossom.

Definition 1 *A blossom is a cycle of length 2k+1 with k edges matched . The base of the blossom is the one and the only vertex in the cycle with both incident edges unmatched.*

Naturally for shrinking to represent a valid operation while searching for augmenting path, we must show that by shrinking a blossom does not add or omit augumenting paths in the graph. The exact method and the proof of correctness will be discussed later. First we prove the following lemma.

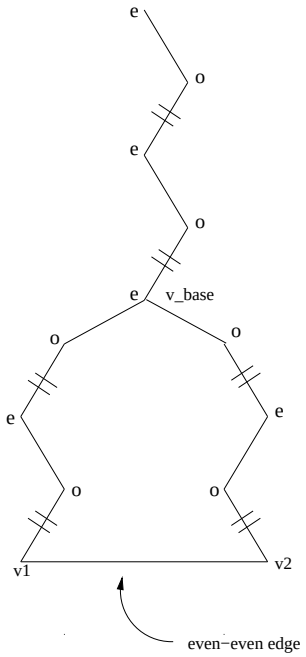


Figure 1: Blossom

LEMMA 3 *Let M be a matching. Let B be a blossom such that the base is unmatched in G. Let $G^{'}$ be the graph obtained by shrinking B. Let $M^{'}$ be the edges of M outside B. There is an augumenting path in G w.r.t M iff there is an augumenting path in $G^{'}$ w.r.t $M^{'}$.*

PROOF: *Part 1:* ($\Rightarrow$) Given an augmenting path in G w.r.t M to show that there exist an augmenting path in $G^{'}$ w.r.t $M^{'}$.

1. Case (i): The augmenting path in G does not intersect the blossom. So the same augmenting path exists in $G^{'}$ w.r.t $M^{'}$.
2. Case (ii): The augmenting path intersects the cycle in G.
   Claim: We can find an augumenting path in $G^{'}$ if we stop as soon as we hit the shrunk vertex(say v_o) while travelling along the augmenting path of G.
   Proof: This is because of the fact that v_o in unmatched in $G^{'}$. Note that all the vertices in the blossom except the base already have an incident matching edge inside the blossom and the base is assumed to be unmatched.

Part 2:($\Leftarrow$) Given an augmenting path in $G^{'}$ w.r.t $M^{'}$ to show that there exist an augmenting path in G w.r.t M.

1. Case (i): The augmenting path in $G^{'}$ does not contain v_o. Then the same path is an augmenting path in G also.
2. Case (ii): The augmenting path contain v_o. Since v_o is an unmatched vertex in $G^{'}$, the augmenting path can either start of end at v_o.

   On expanding the blossom, this path will intersect the blossom in the graph G. Thus the path will contain some edge incident on some vertex of the blossom. The vertex through which the path enters the blossom can be either the base or any other vertex of the blossom.

(a) If the path contains the edge incident on the base of the blossom then we have found an augmenting path in G as the base in unmatched.

(b) If the path contains edge incident on any other vertex of the cycle, i.e., other than the base, then extend this path by moving along the cycle to reach the unmatched base and we will have an augmenting path in G.

□

We note that in the figure above the base is unmatched only when the common ancestor of v_1 and v_2 is the vertex from which we started searching for an augmenting path. To use this lemma, we need to do something more once we identify a blossom. This we will see in the next lecture.

CS 435 : Linear Optimization — Fall 2006

Lecture 20: Algorithm for Non-bipartite Matching

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Lakulish Antani*
Indian Institute of Technology, Bombay

1 Recap

We have seen an algorithm for finding a maximal matching in a bipartite graph. The algorithm is based on a modified BFS, which tries to find an augmenting path with respect to the current matching, if one exists; if no such augmenting path exists, then we conclude that the current matching is maximal. However, the algorithm cannot be used (without modifications) for general non-bipartite graphs.

We have seen that if we encounter an even-even edge during the search, we have an odd-length cycle in the graph. We handle blossoms by "shrinking" them to a single vertex; the only modification required to the previous algorithm is basically to detect and shrink blossoms. This works because of the following theorem (which we have proved):

Theorem 1 *Let $G = (V, E)$ be a graph, and M be a matching in G. Suppose B is a blossom such that its base is unmatched. Consider the graph $G' = (V', E')$ obtained by shrinking B, and the matching M' composed of the matched edges of M which are still present in E'. Then there is an augmenting path in G w.r.t. M if and only if there is an augmenting path in G' w.r.t. M'.*

2 Finding an Augmenting Path

To find an augmenting path for general graphs, we start the (modified) BFS from an unmatched vertex, as before. However, we may encounter an even-even edge between two even vertices v_1 and v_2 (say). These are part of a blossom in the graph, and we first find the base of the blossom. This is easily seen to be the common ancestor v_0 of v_1 and v_2.

Now we need to make sure that v_0 is unmatched for the above theorem to apply. This is clearly true if v_0 is the starting vertex (u). If it is not the starting vertex, then since it has been reached from u, it is matched to its parent in the search graph. Note that there will be an even-length path from v_0 to u. If, on the path p from u to v_0, we change all matched edges to unmatched and vice-versa, we are still left with a valid matching M''. Now it can be shown that there is an augmenting path in G w.r.t M if and only if there is an augmenting path in G w.r.t. M''.

Hence the algorithm is roughly as follows:

```
perform the modified BFS on G as before
u := unmatched vertex from which search begins
if when expanding an even vertex v1, we find an edge to another even vertex v2
    v0 := common ancestor of v1 and v2 in the search graph
    if  v0 = u
        shrink v0
    else
        p := path from u to v0
        change all matched edges in p to unmatched, and vice-versa
        shrink v0
```

In the next lecture, we will see a detailed proof of why the **else** clause above works, and a proof of correctness for the algorithm.

3 Time Complexity

We shall show a generous bound on the time complexity of the algorithm. Suppose the graph is $G = (V, E)$, and that $|V| = n$ and $|E| = m$. Observe that:

- The number of shrinkages performed can clearly be no more than the number of vertices in the graph, i.e. n.
- In the worst case, BFS is performed n times.
- Each BFS takes $O(m)$ time.

So we get a generous bound of $O(n^2m)$ on the complexity.

CS 435 : Linear Optimization — Fall 2006

Lecture 20: A BFS based algorithm for matching in general graph (contd.)

Lecturer: *Sundar Vishwanathan* — Scribe: *Rakesh Venkat*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Overview

In the previous lecture, we looked at obtaining a new graph G' from the given graph G by shrinking one or more *Blossoms* to a single 'macrovertex' in G. In this lecture, we look at relations between augmenting paths in G' and G. We then develop the algorithm for matching in general graphs based on the modified BFS algorithm used in the bipartite matching.

2 Augmenting paths in G' and G

Let M be a matching in the graph G. Suppose B is a blossom such that the *base is unmatched*(Fig: 1 (a)). Shrink vertices of B to a single vertex to get a new graph G'. Note that the base may also be right at the 'top' of the BFS search tree, as shown in Fig: 1 (b). The matching M' in G' consists of those edges in M that remain in G'.

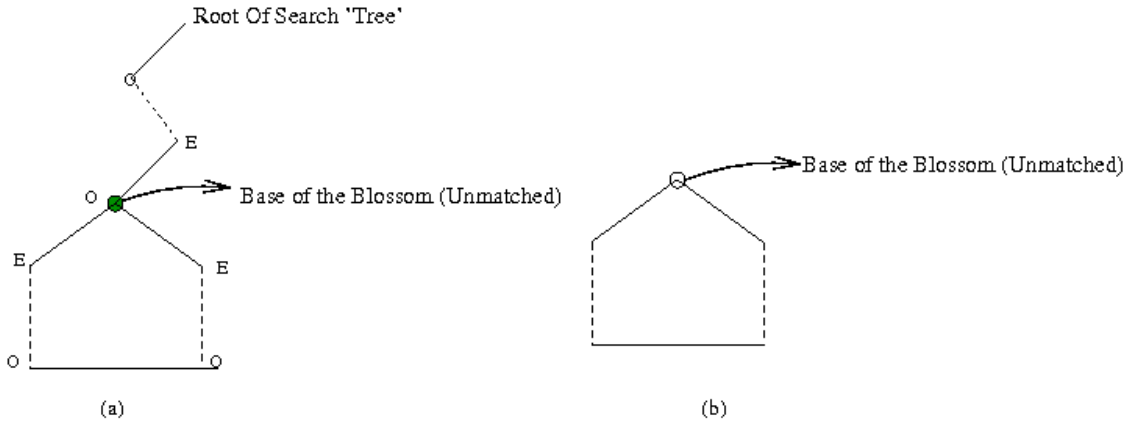


Figure 1: (a) A blossom with an unmatched base (b) A blossom with an unmatched base at the root of the search tree in BFS-modified algo

We start with the following theorem:

Theorem 1 *M' is maximal in G'* iff *M is maximal in G. This is equivalent to proving that* $\exists$*an augmenting path in G* iff $\exists$*an augmenting path in G'.*

Proof: *(This theorem was proved in the previous lecture and was repeated in the class for clarity.)*

Observation 1: The new (shrunk) vertex (say v_0) in the graph G' is unmatched (Fig: 2). This is because all vertices in the blossom B except the base are already matched, and hence can connect to the outer vertices only through unmatched edges. By our assumption, the base is also unmatched, hence v_0 is unmatched.

Part 1: Assume there is an augmenting path P' in G'

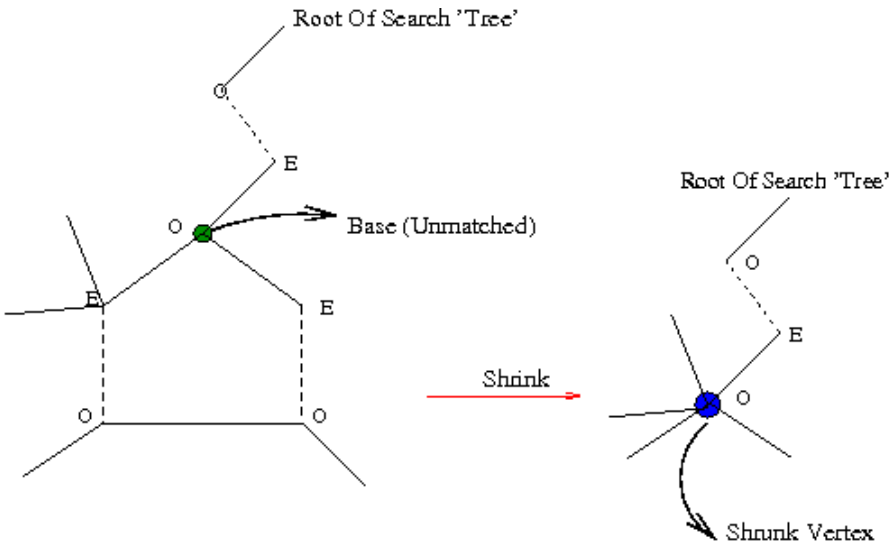


Figure 2: Shrinking a Blossom. The corresponding vertex in the new graph is unmatched.

1. Case (i): If it avoids v_0, then the same augmenting path is present in G too.
2. Case (ii): If it goes through v_0, then as v_0 is unmatched, P' either starts or ends at v_0.
   We can then trace a corresponding augmenting path in P by expanding the blossom, and noting the vertex next to v_0 in P' as shown in Fig: 3. One end of the augmenting path in P is always the base of the blossom. Also, the portion of the path P in the blossom is always of even length.

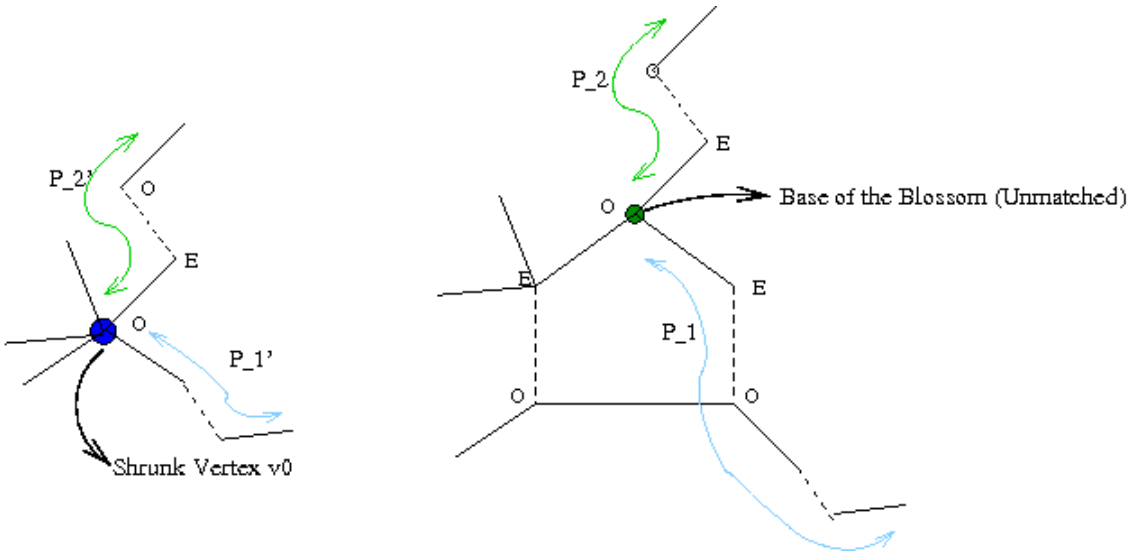


Figure 3: Mapping augmenting paths in G to G' and vice versa. The curved lines of the same colour in the two graphs indicate corresponding augmenting paths.

Part 2: Assume there is an augmenting path P in G

1. Case (i): If it doesnt hit a blossom (B) , we are done, the same augmenting path works in G'.
2. Case (ii): If P goes through the Blossom, the last edge in the path before it hits the blossom must be unmatched.

Thus, no path in G' can go past v_0 in accordance with *Observation 1.*. The part of the path P from the start till it hits the blossom is thus an augmenting path P' in G'. (Again, see Fig: 3).

□

3 Adapting the BFS-based algorithm to the general graph

We run the modified BFS algorithm as in the bipartite case, but with a small modification. In this algorithm, at any point of time, we have a *tree* starting from the current root vertex, and a matching on this.
The only place where we encounter a problem is when we hit a blossom. We can identify a blossom when we follow an edge to an already explored even vertex from an even vertex we are currently at. (See Fig:4). Also note that in such a case, the first explored vertex in the blossom is the base (the lowest common ancestor of the mentioned two even vertices in the tree). The base has to be matched if we are to *enter* the blossom, as in the figure. The algorithm would continue normally as in the bipartite case if we entered the blossom through any other point but the base.

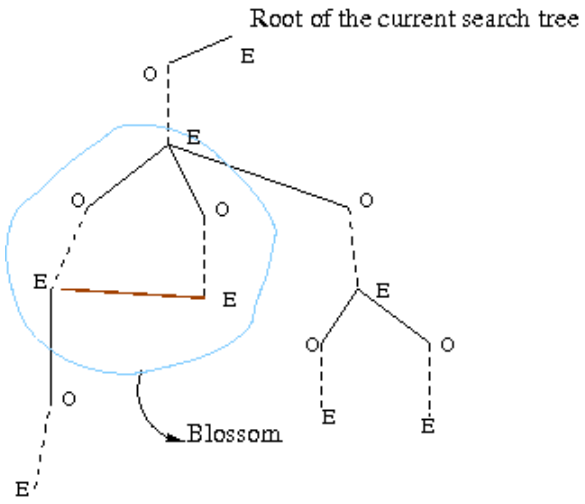


Figure 4: Discovering a blossom in the BFS-based algo in the general graph. On encountering the Brown edge shown, we infer a blossom, and proceed to shrink it, switching the matched-unmatched edges from the root to the base.

On encountering the above case, we do the following:

1. Switch the state of edges on the path till the base from matched-unmatched and vice versa. This preserves the matching size and augmenting paths. (in the next lecture this step is justified in more detail). Call the new graph G_1.

2. Now, the blossom satisfies the conditions of the Theorem above. Shrink the blossom B in G_1 to a single vertex. Call the new graph G'.

   By the previous theorem, augmenting paths in G_1 are preserved in G' and vice versa, and we can also map the paths(and hence matching) appropriately.

Now we continue as usual. We see that in the tree maintained in the general case (as opposed to the bipartite case), the vertices at each level may be either (a) normal vertices or (b) shrunk blossoms.

On finding an augmenting path in the new graph, find the corresponding augmenting path in G_1 (and hence G) by unshrinking the shrunk blossoms one by one in reverse order of shrinking as explained in the theorem above (Part 1), and update the matching.

The algorithm will terminate when there are no remaining augmenting paths in the graph.

Runtime

We shrink blossoms atmost $|V|$ times. Till we hit a blossom, the length of the path is atmost $|E|$. To discover all augmenting paths, we have to run the algorithm from each vertex in turn. So a loose upper bound is $O(|V|^2|E|)$. The algorithm can be implemented in $O(|E|\sqrt{|V|})$.

CS 435 : Linear Optimization — Fall 2006

Lecture 21: Review of graph algorithms (contd.)

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Chaitanya Gokhale*
Indian Institute of Technology, Bombay

1 Recap of previous lecture

Previous lecture(s) dealt with the BFS based algorithm for matching in general graphs. The following lemma was stated and proved earlier :

Lemma 1 *Let M be a matching. let B be a blossom such that the base is unmatched. Let G' be the graph obtained by shrinking B. Let M' be the edges of M outside B. There is an augumenting path in G w.r.t M iff there is an augumenting path in G' w.r.t M'.*

In this lecture, we prove a complimentary lemma which completes the proof of correctness of the algorithm to find a maximum matching in a general graph.

2 Complimentary Lemma and its proof

In order to find an augmenting path in a general graph G w.r.t. matching M, we proceed in the same way as in case of bipartite graphs :

1. Start from an unmatched vertex and do a BFS search. However, in this case we might encouter a blossom.
2. If we reach a blossom B during BFS search, then switch the matched and unmatched edges of the path till the base of the bottom, as a result of which the base becomes unmatched (as shown in fig 1). Let us call the new matching as M'.
3. Now, shrink the blossom B to a single vertex forming graph G' and matching M''. Look for an augmenting path in the new graph.
   We have an augmenting path in G' w.r.t. M'' iff there exists an augmenting path in G w.r.t. matching M.

There exists an augmenting path in G w.r.t. matching M' iff there exists an augmenting path in G' w.r.t. matching M'' (by Lemma 1 above). Hence, in order to prove the correctness of algorithm, we need to prove the first part of the algorithm regarding switching, which can be stated as :

Lemma 2 *Given a graph G and matching M and a path Q starting from an unmatched node v, to the base u of a blossom B (base u being matched). Let M' be the matching obtained on switching the matched and unmatched edges along path Q. Then an augmenting path exists in G w.r.t matching M iff there exists one in G w.r.t. matching M'.*

Proof:

Suppose that there exists an augmenting path P in the Graph G, after switching i.e. w.r.t. matching M'. We have two cases to consider :

(a) The augmenting path P has no edge common with path Q :
Then the same path exists in G w.r.t. matching M, as well.

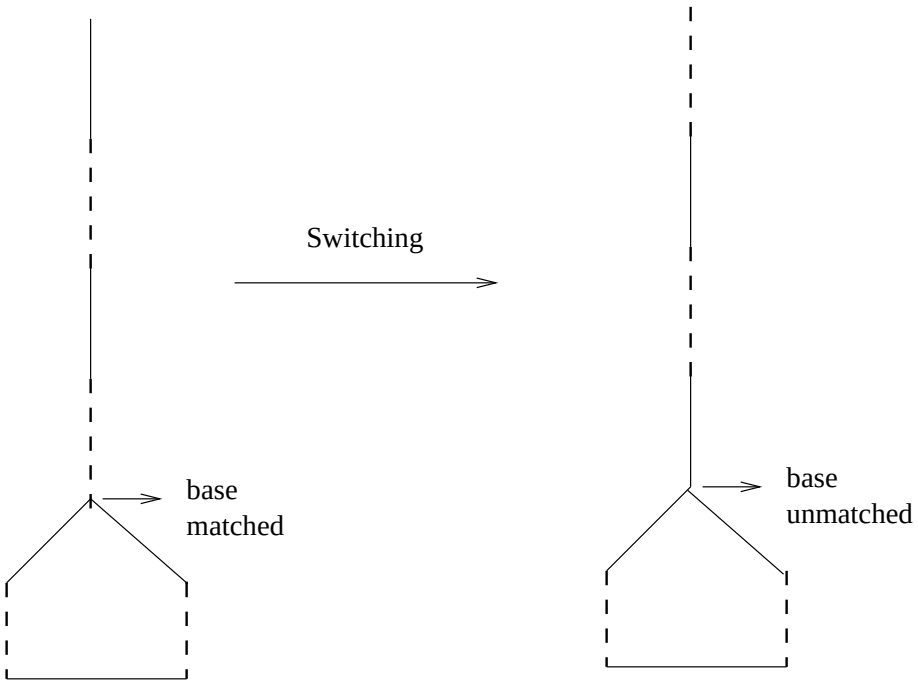


Figure 1: Switching matched and unmatched edges

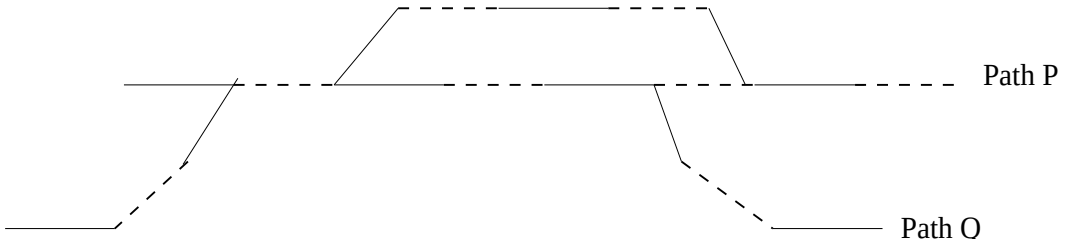


Figure 2: P & Q having Common edges

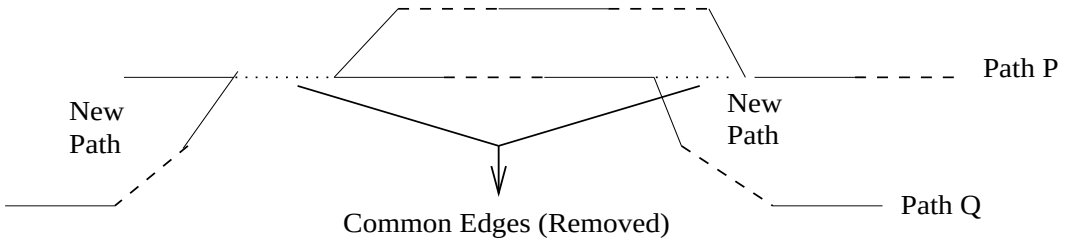


Figure 3: New paths connectign end-points after ignoring common edges

(b) The augmenting path P has some edges common with the path Q (see figure 2):

Then, we mark out the common edges to both the paths. We wish to construct an augmenting path in G w.r.t. M. For that, ignore the common edges in P and Q. We have four end points of the two paths P and Q that orginally existed. Since P is an augmenting path both its end-points are free(unmatched), while one of the end-points of Q is free. Thus, 3 of the 4 end-points are unmatched. On removing the common edges, the 2 pairs of end-points will still be connected by two different paths (refer figure 3. Since 3 of the 4 end-points are unmatched, there must be one of the two new paths having unmatched end-points. This path is alternating and has unmatched end-points. $\Rightarrow$ This is the augmenting path we are looking for.

From (a) & (b) above, if G contains an augmenting path w.r.t. $M^{`}$ then there is an augmenting path w.r.t. M as well. The above argument (cases (a) & (b)) holds for the converse as well and hence, the converse can be proved in the exact same manner.

Hence, the lemma stated above is proved.

□

CS 435 : Linear Optimization Fall 2006

Lecture 22: Using LP techniques to design algorithms for combinatorial problems

Lecturer: *Sundar Vishwanathan* Scribe:*Aditya Varma*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Integer Linear Programs

An *Integer Linear Program* is a linear program where the variables are constrained to take integer values only. An ILP is formulated as

$$\begin{gathered}\max c^T x \\ Ax \leq b \\ x_i \text{ is integral } \forall i.\end{gathered}$$

2 Recap of primal,dual and complementary slackness

Recapitulate that for an LP

$$\begin{gathered}\max c^T x \\ Ax \leq b\end{gathered}$$

the dual is

$$\begin{gathered}\min y^T b \\ A^T y = c\end{gathered}$$

Complementary slackness implies that the optimums of the primal and dual coincide i.e. $c^T x_o = (y_o)^T b$ and where $y_{oi} > 0$ in the dual $A_i x_0 = b_i$ in the primal and vice versa.

3 Formulation Of Minimum Spanning Tree as an ILP

A *spanning tree* is a subgraph of a connected,undirectional graph that connects all vertices and is a tree. The minimum spanning tree is a spanning tree with minimum cost or sum of weight of edges part of the tree. The corresponding ILP may be formulated taking as variables x_e ,one for each edge where a value of 1 for x_e indicates that the edge is part of the tree and value of zero indiacates that the edge is not part of the spanning tree. The cost then is $\sum x_e c_e$ where c_e is the weight of the edge. The ILP formulation then is

$$\min c^T x = \sum x_e c_e$$

Constraints :

$$\begin{gathered} \forall \text{ partitions } \pi \\ \sum_{e crosses \pi} x_e \geq \#(\pi) - 1 \\ 0 \leq x_e \leq 1 \\ x_e \text{ is integral} \end{gathered}$$

where $\#(\pi)$ is the number of parts in the partition π. The first constraint basically represents that the graph is connected.

Now we drop the integral constraint and write the dual of the resulting LP. The dual is

$$\max(\sum_{\pi} y_\pi(\#(\pi) - 1))$$

$$\begin{gathered} \forall e : \sum_{e crosses \pi} y_\pi \leq c_e \\ y_\pi \geq 0 \end{gathered}$$

.

CS 435 : Linear Optimization — Fall 2006

Lecture 22.: Using LP techniques to design algorithms for combinatorial problems : Deriving Kruskal's MST algorithm using LP duality

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Manjinder Singh Chauhan*
Indian Institute of Technology, Bombay

1 Integer Linear Program(ILP)

An Integer linear program is a linear program with an extra constraint that all x_i are integral.
Formally

$$\max\ c^T x$$
$$\mathrm{A}x \leq b$$

where x_i is intergral $\forall i$

Now we allready know that that for

Primal	Dual
$\max\ c^T x$	$\min\ y^T b$
$Ax \leq b$	$A^T y = c$
	$y \geq 0$

if x, y are feasiable and
$y_i > 0 \Longrightarrow \mathrm{A}_i\mathrm{x}{=}b_i$
then the solution is optimal.

The second question in quiz 2 was an attempt to make the above process iterative.

$\max\ c^T x$	$\max\ c^T y$
$Ax \leq b$	$A_1 y \leq 0$
for an x_o	
$A_1 x_o = b_1$	This can be solved much more easily
$A_2 x_o < b_2$	

Now $x = x_o + \epsilon y$
where ϵ depends on A.

2 Formulation of Minimum Spanning Tree as an ILP

Variables: X_e , one per edge
Cost: min $\mathrm{C}^T X$

where C_e is cost of edge e
Constraint: $0 \leq X_e \leq 1$ and X_e is an integer

$$\forall \text{ partition } \pi$$
$$\sum_{ecrosses\pi} X_e \geq \#(\pi) - 1$$
where $\#(\pi)$is the number of parts in a partition.

Next we drop the constraint that X_e is an integer and also that $X_e \leq 1$ Now writing the dual of the MST linear program we get

$$max \sum_{\pi} y_\pi(\#(\pi) - 1)$$
$$\forall \text{ e } \sum_{ecrosses\pi} y_\pi \leq C_e$$
$$y_\pi \geq 0$$

CS 435 : Linear Optimization Fall 2006

Lecture 23: An example illustrating the LP duality based MST algorithm

Lecturer: *Sundar Vishwanathan* Scribe: *TAs*
Computer Science & Engineering Indian Institute of Technology, Bombay

In this note we illustrate the MST algorithm derived in the class using LP duality.

1 Relaxed LP for the MST of an undirected graph and its Dual

The *relaxed* LP formulation (i.e. after removing the constraints x_e *integer*, $x_e \leq 1$) for the MST of a graph $G(V, E)$ is the following:

$$\begin{aligned} \min \quad & \sum_e c_e x_e & \\ \text{s.t.} \quad & \sum_{e \text{ crosses } \pi} x_e \geq \#(\pi) - 1 & \forall \pi \\ & x_e \geq 0 & \forall e \in E \end{aligned}$$

where

- $c_e \geq 0$ is the weight of the edge $e \in E$
- π denotes a partition of the vertex set V and can be represented as the set of disjoint subsets of the vertex set V, i.e. $\pi \equiv \{p_1, p_2, \ldots, p_k\}, p_i \subset V, \cup p_i = V, p_i \cap p_j = \phi$. $\#$ is the number of 'parts' in π i.e. $= k$
- e crosses π means one end vertex of e belongs to p_i and other end belongs to $p_j, i \neq j$.

The dual LP is given by

$$\begin{aligned} \max \quad & \sum_e y_\pi (\#(\pi) - 1) & \\ \text{s.t.} \quad & \sum_{e \text{ crosses } \pi} y_\pi \leq c_e & \forall e \in E \\ & y_\pi \geq 0 & \forall \pi \end{aligned}$$

2 Formulations for an example graph

Let us see the formulation for an example graph in figure 1(i).

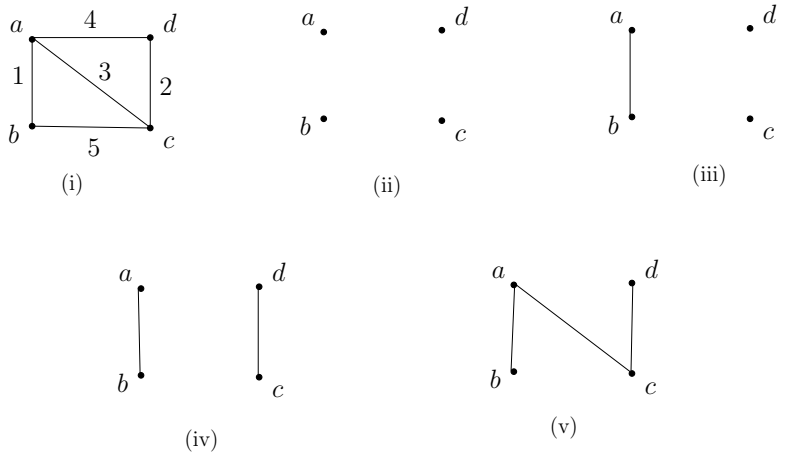


Figure 1: An example graph and intermediate steps of the algorithm

The relaxed primal LP is:

$$
\begin{array}{lrll}
\min & x_{ab} + 2x_{cd} + 3x_{ac} + 4x_{ad} + 5x_{bc} & & \\
\text{s.t.} & x_{ab} + x_{cd} + x_{ac} + x_{ad} + x_{bc} \geq 4 - 1 & \text{for } \pi = \{a, b, c, d\} \\
& x_{cd} + x_{ac} + x_{ad} + x_{bc} \geq 3 - 1 & \text{for } \pi = \{ab, c, d\} \\
& x_{ab} + x_{cd} + x_{ac} + x_{bc} \geq 3 - 1 & \text{for } \pi = \{ac, b, d\} \\
& x_{ab} + x_{cd} + x_{ad} + x_{bc} \geq 3 - 1 & \text{for } \pi = \{ad, b, c\} \\
& x_{ab} + x_{cd} + x_{ac} + x_{ad} + x_{bc} \geq 3 - 1 & \text{for } \pi = \{bc, a, d\} \\
& x_{ab} + x_{cd} + x_{ac} + x_{ad} \geq 3 - 1 & \text{for } \pi = \{bd, a, c\} \\
& x_{ab} + x_{ac} + x_{ad} + x_{bc} \geq 3 - 1 & \text{for } \pi = \{cd, a, b\} \\
& x_{ac} + x_{ad} + x_{bc} \geq 2 - 1 & \text{for } \pi = \{ab, cd\} \\
& x_{ab} + x_{cd} + x_{ac} \geq 2 - 1 & \text{for } \pi = \{ac, bd\} \\
& x_{ab} + x_{cd} + x_{ad} + x_{bc} \geq 2 - 1 & \text{for } \pi = \{ad, bc\} \\
& x_{cd} + x_{ac} + x_{bc} \geq 2 - 1 & \text{for } \pi = \{abc, d\} \\
& x_{cd} + x_{ad} \geq 2 - 1 & \text{for } \pi = \{abd, c\} \\
& x_{ab} + x_{bc} \geq 2 - 1 & \text{for } \pi = \{acd, b\} \\
& x_{ab} + x_{ac} + x_{ad} \geq 2 - 1 & \text{for } \pi = \{bcd, a\} \\
& x_e \geq 0 & \forall e = 1..5
\end{array}
$$

And the corresponding dual is

max

$$
\begin{aligned}
3y_{\{a,b,c,d\}} + 2y_{\{ab,c,d\}} + 2y_{\{ac,b,d\}} + 2y_{\{ad,b,c\}} + 2y_{\{bc,a,d\}} + 2y_{\{bd,a,c\}} + 2y_{\{cd,a,b\}} \\
+ y_{\{ab,cd\}} + y_{\{ac,bd\}} + y_{\{ad,bc\}} + y_{\{abc,d\}} + y_{\{abd,c\}} + y_{\{acd,b\}} + y_{\{bcd,a\}}
\end{aligned}
$$

subj. to

$$
\begin{aligned}
e = ab : y_{\{a,b,c,d\}} + y_{\{ac,b,d\}} + y_{\{ad,b,c\}} + y_{\{bc,a,d\}} + y_{\{bd,a,c\}} + y_{\{cd,a,b\}} + y_{\{ac,bd\}} + y_{\{ad,bc\}} + y_{\{acd,b\}} + y_{\{bcd,a\}} \leq 1 \quad (1) \\
e = cd : y_{\{a,b,c,d\}} + y_{\{ab,c,d\}} + y_{\{ac,b,d\}} + y_{\{ad,b,c\}} + y_{\{bc,a,d\}} + y_{\{bd,a,c\}} + y_{\{ac,bd\}} + y_{\{ad,bc\}} + y_{\{abc,d\}} + y_{\{abd,c\}} \leq 2 \quad (2) \\
e = ac : y_{\{a,b,c,d\}} + y_{\{ab,c,d\}} + y_{\{ac,b,d\}} + y_{\{bc,a,d\}} + y_{\{bd,a,c\}} + y_{\{cd,a,b\}} + y_{\{ab,cd\}} + y_{\{ac,bd\}} + y_{\{abc,d\}} + y_{\{bcd,a\}} \leq 3 \quad (3) \\
e = ad : y_{\{a,b,c,d\}} + y_{\{ab,c,d\}} + y_{\{ad,b,c\}} + y_{\{bc,a,d\}} + y_{\{bd,a,c\}} + y_{\{cd,a,b\}} + y_{\{ab,cd\}} + y_{\{ad,bc\}} + y_{\{abd,c\}} + y_{\{bcd,a\}} \leq 4 \quad (4) \\
e = bc : y_{\{a,b,c,d\}} + y_{\{ab,c,d\}} + y_{\{ac,b,d\}} + y_{\{ad,b,c\}} + y_{\{bc,a,d\}} + y_{\{cd,a,b\}} + y_{\{ab,cd\}} + y_{\{ad,bc\}} + y_{\{abc,d\}} + y_{\{acd,b\}} \leq 5 \quad (5) \\
\forall \pi \qquad y_\pi \geq 0 \quad (6)
\end{aligned}
$$

3 The primal-dual algorithm

The algorithm that we designed is the following:

Algorithm 1 MST algorithm developed using LP duality

Initialize all $y_\pi \leftarrow 0$
Set $i \leftarrow 0, \pi_0 \leftarrow$ the partition in which each vertex is a part by itself
repeat
  Raise y_{π_i} until dual constraint corresponding to some edge e becomes tight
  Set $x_e \leftarrow 1$
  Set $\pi_{i+1} \leftarrow$ the partition got from π_i by merging end points of e
  Set $i \leftarrow i + 1$
until edges with $x_e = 1$ forms a spanning tree

Let us apply the technique on the example graph in figure 1(i). Intermediate steps of the algorithm 1 on this instance are given in table 1. Initially all $y_\pi = 0$. This corresponds to figure 1(ii).

Property \ Iteration	1^{st}	2^{nd}	3^{rd}
π considered	$\{a, b, c, d\}$	$\{ab, c, d\}$	$\{ab, cd\}$
[constraint involved, max raise in y_π possible]	[(1),1], [(2),2], [(3),3] [(4),4], [(5),5]	[(2),2-1], [(3),3-1], [(4),4-1],[(5),5-1]	[(3),3-1-1], [(4),4-1-1] [(5),5-1-1]
constraint tightens first ie for which max raise is min	(1)	(2)	(3)
y_π raised	$y_{\{a,b,c,d\}} = 1$	$y_{\{ab,c,d\}} = 2 - 1 = 1$	$y_{\{ab,cd\}} = 3 - 1 - 1 = 1$
x_e set	$x_{ab} = 1$	$x_{cd} = 1$	$x_{ac} = 1$
Vertices of e	a, b	c, d	a, c
New π	$\{ab, c, d\}$	$\{ab, cd\}$	$\{abcd\}$, MST found
Sub-figure no.	(iii)	(iv)	(v)

Table 1: Trace of the primal-dual algorithm on the example graph

CS 435 : Linear Optimization — Fall 2006

Lecture 23: Primal-dual algorithm for MST

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Keshri Nandan Nayak*
Indian Institute of Technology, Bombay

1 Last lecture

In last class we derived relaxed LP formulation for the MST of a graph $G(V, E)$, the derived LP was:

$$\begin{aligned} &\min \sum_e c_e x_e && \text{(ensures minimum cost)} \\ &\text{s.t. for partition } \pi \\ &\sum_{e \text{ crosses } \pi} x_e \geq \#(\pi) - 1 && \text{(ensures graph is connected)} \\ &x_e \geq 0 \qquad \forall e \in E \end{aligned}$$

where

- $c_e \geq 0$ is the cost of the edge $e \in E$
- π denotes a partition of the vertex set V and $\#(\pi)$ denotes the number of 'parts' in π.
- e crosses π means the end-points of e are in different parts of π.

Here the constraint x_e *integer*, $1 \geq x_e \geq 0$ was removed, and therefore LP was termed as relaxed. The dual of the above LP is given by

$$\begin{aligned} \max \quad & \sum_\pi y_\pi (\#(\pi) - 1) \\ \text{s.t.} \quad & \sum_{e \text{ crosses } \pi} y_\pi \leq c_e && \forall e \in E \\ & y_\pi \geq 0 && \forall \pi \end{aligned}$$

2 The primal-dual algorithm

As per the primal dual algo we start with any feasible dual solution. Thus the algorithm for finding MST of a graph is:

Initialize all $y_\pi := 0$
Set $i := 0, \pi_0 :=$ as the partition in which each vertex is a part by itself

Repeat:
  Raise y_{π_i} until dual constraint corresponding to some edge e becomes tight i.e. $\sum y_n = c_e$.
  Set $x_e := 1$
  Set $\pi_{i+1} :=$ the partition got from π_i by merging end points of e
  Set $i := i + 1$
Until: edges with $x_e = 1$ forms a spanning tree

In each iteration, we try and improve the dual solution. This we do while maintaining feasibilty. In this case we will raise the value of the dual variables one by one.

Once a constraint is tight, for an edge we will not raise the dual variable for partitions which this edge crosses. Note that implicitly we are doing what problem 2 in Quiz 2 dictates.

After each iteration, we take the edges for which the constraints are equalities. If this set of edges yield a connected graph, we are done. A spanning tree will yield a primal feasible solution. Note the use of complementary slackness.
Here is a description of the algorithm.
We start with zero partition (i.e. an empty tree initially) and add edges as we go along until dual constraint corresponding to some edge becomes tight. We then include e into our solution by setting $x_e = 1$ and partition is updated by merging end points of e. The algorithm terminates with a spanning tree having edges with $x_e = 1$ of the input graph.

2.1 Proof of optimality:

We can prove this as optimum solution by proving that cost of primal and dual are same and they both are feasible.
Since cost of primal is given by:

$$\sum_e c_e \quad \text{where e is chosen by the algorithm}$$

$$= \sum_e \sum_{\substack{\pi \\ \text{e crosses } \pi}} y_\pi$$

$$= \sum_\pi \sum_{\substack{e \\ \text{e crosses } \pi}} y_\pi$$

$$= \sum_\pi y_\pi \sum_{e \text{ crosses } \pi} 1$$

$$= \sum_\pi y_\pi \ (\#\text{ of edges chosen which crosses } \pi)$$

$$\Downarrow$$

$$= \sum_\pi y_\pi \ (\#(\pi) - 1) \qquad \text{Since when } y_\pi > 0 \text{ the \# of edges chosen which crosses } \pi \text{ is exactly } (\#(\pi) - 1) \text{ for this algo.}$$

$$= \text{cost of dual}$$

Thus there is a feasible primal and feasible dual and their cost are same. $\Rightarrow$ Solution is optimum.
Note: In this LP, the constraint is exponential and so the dual has an exponential number of variables.

2.2 Exercises:

Run the primal-dual algorithm for following cases:

- π restricted to 2 parts and x_e are not integral, manipulate dual variables to get Krusal's algo. We will get struck in some point for this case while trying the above proof of optimality.
- $\forall$ n, find an example where optimum(LP) $<$ weight of MST.
- Add a new constraint $\sum_e X_e = n - 1$. And try the above exercise.

CS 435 : Linear Optimization — Fall 2006

Lecture 23: Primal-dual algorithm for MST (contd..)

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Mahak Patidar*
Indian Institute of Technology, Bombay

1 A brief recap of last lecture

The *relaxed* LP formulation (i.e. after removing the constraints x_e *integer*, $x_e \leq 1$) for the MST of a graph $G(V, E)$ is the following:

$$\begin{aligned} \min \quad & \sum_e c_e x_e \\ \text{s.t.} \quad & \sum_{e \text{ crosses } \pi} x_e \geq \#(\pi) - 1 \quad \forall \pi \\ & x_e \geq 0 \qquad\qquad \forall e \in E \end{aligned}$$

where

- $c_e \geq 0$ is the weight of the edge $e \in E$
- π denotes a partition of the vertex set V and can be represented as the set of disjoint subsets of the vertex set V, i.e. $\pi \equiv \{p_1, p_2, \ldots, p_k\}, p_i \subset V, \cup p_i = V, p_i \cap p_j = \phi$. $\#$ is the number of 'parts' in π i.e. $= k$
- e crosses π means one end vertex of e belongs to p_i and other end belongs to $p_j, i \neq j$.

The dual LP is given by

$$\begin{aligned} \max \quad & \sum_\pi y_\pi (\#(\pi) - 1) \\ \text{s.t.} \quad & \sum_{e \text{ crosses } \pi} y_\pi \leq c_e \qquad \forall e \in E \\ & y_\pi \geq 0 \qquad\qquad \forall \pi \end{aligned}$$

2 The algorithm for MST

The algorithm that we designed is the following

Algorithm 1 MST algorithm developed using LP duality

```
Initialize all y_π ← 0
Set i ← 0, π_0 ← the partition in which each vertex is a part by itself
repeat
    Raise y_{π_i} until dual constraint corresponding to some edge e becomes tight
    Set x_e ← 1
    Set π_{i+1} ← the partition got from π_i by merging end points of e
    Set i ← i + 1
until edges with x_e = 1 forms a spanning tree
```

We start with any feasible solution for the dual LP and maintain feasibility in both primal and dual LP's. In each iteration, we try to improve the dual solution. For this, we raise the value of the dual

variables one by one. If a constraint becomes tight for an edge we donot raise the dual variable for partitions which this edge crosses. Note that implicitly we are doing what problem 2 in Quiz 2 dictates.

After each iteration, we take the edges for which the constraints are equalities. If this set of edges yield a connected graph, we are done. A spanning tree will yield a primal feasible solution. Note the use of complementary slackness.
We essentially start with zero partition (i.e. an empty tree initially) and add edges as we go along until dual constraint corresponding to some edge e becomes tight. We then include e into our solution by setting $x_e = 1$ and partition is updated by merging end points of e. The algorithm terminates with a spanning tree having edges with $x_e = 1$ of the input graph.

3 Proof of optimality

We can prove that the above algorithm gives optimum solution by proving that cost of primal and dual are same and they both are feasible.

Cost of primal is given by:

$$\begin{aligned}
&\sum_e c_e \quad \text{where e is chosen by the algorithm} \\
= &\sum_e \sum_{\substack{\pi \\ \text{e crosses } \pi}} y_\pi \\
= &\sum_\pi \sum_{\substack{e \\ \text{e crosses } \pi}} y_\pi \\
= &\sum_\pi y_\pi \sum_{e \text{ crosses } \pi} 1 \\
= &\sum_\pi y_\pi \ (\#\text{ of edges chosen which crosses } \pi)
\end{aligned}$$

$$\Downarrow$$

$$(\#(\pi) - 1) \leftarrow \text{where } y_\pi > 0$$

This is exactly the cost of the dual. So we have proved that the cost of primal is same as the cost of the dual $\Rightarrow$ both of them are optimal

4 Some comments and observations

- Optimality can be proved by using complimentary slackness
- This LP has an integral solution which is optimal
- Number of constraints is exponential(becaouse number of partitions is exponential)
- The edges which become tight are in increasing order of weights

5 Exercises

- Run the algoritm with π restricted to 2. Manipulate the dual variables so that you end up with Kruskal's algorithm. Apply the above proof of correctness, you will get struck.

- $\forall$ n, find an example where optimum of the LP is less than the weight of the minimum spanning tree.
- Prove that if we add a new constraint

$$\sum x_e = n - 1,$$

  there exist an integral solution to the LP
- Observe that the way we choose y_π corresponds to the last problem of quiz 2.

CS 435 : Linear Optimization Fall 2006

Lecture 24: ILP formulations for SAT and shortest path problem.

Lecturer: *Sundar Vishwanathan* Scribe: *Kshitij Bhardwaj*
Computer Science & Engineering Indian Institute of Technology, Bombay

Writing ILP(Integer Linear Program) for the following
a) SAT
b) Shortest Path

1 SAT :

For a given conjunction of clauses find an assignment of truth values for the literals so that the expression evaluates to true. Clause - Disjunction of literals

For Example - $((x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6))$
Now we have to assign truth values to x_i's such that expression evaluates to true. Clearly for this to hold each of the clause should evaluate to true.

Hence the input and output for the SAT problem are as follows :
Input : Set of clauses
Output : Setting of true/false values to the variables such that each clause evaluates to true.

2 Shortest Path :

Given a directed graph with positive edge weights (see Figure 1) find a shortest path between two given points.

If E is the edge set and w_e is the weight of the edge $e \in E$ then the set $S \subseteq E$ is the shortest path from s to t if $\sum_{e \in S} w_e$ is smallest among all such sets whose elements form a path from s to t.

Input : A set of directed edges, the two vertices(say s and t) between which the shortest path is to be found.
Output : A set of edges such that these edges form a path from s to t and this is the least cost path from s to t.

3 Formulating an ILP

Choice of variables :
The variables are usually the unknows in the problem. And some assignment of values to these will lead us to the desired solution.
In the case of SAT the literals are the ones to which values have to be assigned. And an appropriate assignment to these is what we are looking for. Hence in the ILP for SAT we have one variable for each literal.

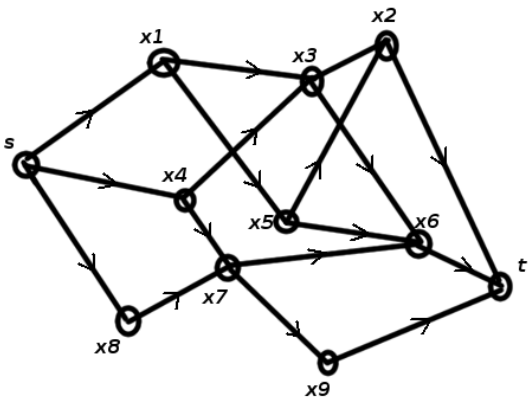


Figure 1: Directed Graph

Constraints :
The constraints in an ILP should be of the form $AX \leq b$. If we look at each constraint individually i.e. each row of the matrix at a time then it will be of the form $A_{i_1}x + A_{i_2}y + ... + A_{i_n}w \leq b_i$ where $A_{i_1}, A_{i_2}, ..., A_{i_n}$ and b_i are constants and $x, y, ..., w$ are the variables. The constraints in case of SAT must force at least one variable in each clause to take value true.
Cost :
The cost should be of the form max/min $C^T X$. In some cases this might not be of any use as in the case of SAT where the cost function is immaterial and we would not need it to solve the problem.

CS 435 : Linear Optimization Fall 2006

Lecture 24: ILP formulations for SAT and shortest path problem.

Lecturer: *Sundar Vishwanathan* Scribe: *Kshitij Bhardwaj*
Computer Science & Engineering Indian Institute of Technology, Bombay

Writing ILP(Integer Linear Program) for the following
a) SAT
b) Shortest Path

1 SAT :

For a given conjunction of clauses find an assignment of truth values for the literals so that the expression evaluates to true. Clause - Disjunction of literals

For Example - $((x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6))$
Now we have to assign truth values to x_i's such that expression evaluates to true. Clearly for this to hold each of the clause should evaluate to true.

Hence the input and output for the SAT problem are as follows :
Input : Set of clauses
Output : Setting of true/false values to the variables such that each clause evaluates to true.

2 Shortest Path :

Given a directed graph with positive edge weights (see Figure 1) find a shortest path between two given points.

If E is the edge set and w_e is the weight of the edge $e \in E$ then the set $S \subseteq E$ is the shortest path from s to t if $\sum_{e \in S} w_e$ is smallest among all such sets whose elements form a path from s to t.

Input : A set of directed edges, the two vertices(say s and t) between which the shortest path is to be found.
Output : A set of edges such that these edges form a path from s to t and this is the least cost path from s to t.

3 Formulating an ILP

Choice of variables :
The variables are usually the unknows in the problem. And some assignment of values to these will lead us to the desired solution.
In the case of SAT the literals are the ones to which values have to be assigned. And an appropriate assignment to these is what we are looking for. Hence in the ILP for SAT we have one variable for each literal.

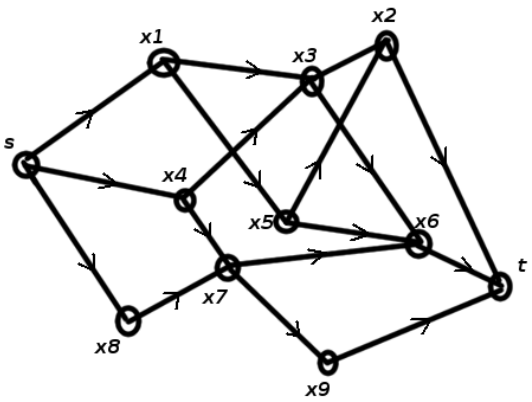


Figure 1: Directed Graph

Constraints :
The constraints in an ILP should be of the form $AX \leq b$. If we look at each constraint individually i.e. each row of the matrix at a time then it will be of the form $A_{i_1}x + A_{i_2}y + ... + A_{i_n}w \leq b_i$ where $A_{i_1}, A_{i_2}, ..., A_{i_n}$ and b_i are constants and $x, y, ..., w$ are the variables. The constraints in case of SAT must force at least one variable in each clause to take value true.
Cost :
The cost should be of the form max/min $C^T X$. In some cases this might not be of any use as in the case of SAT where the cost function is immaterial and we would not need it to solve the problem.

CS 435 : Linear Optimization — Fall 2006

Lecture 24: ILP formulations for SAT and shortest path problem

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Vinay Agarwal*
Indian Institute of Technology, Bombay

1 Formulating ILP for SAT

An instance of a SAT problem is a Boolean expression written using only AND, OR, NOT, variables, and parentheses in its CNF form. The question is: given the expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true?

Input: A set of boolean variables z_1,z_2,z_3, ... ,z_{n-1},z_n, and a boolean expression in CNF for which a satisfying assignment is to be found.

$$\begin{gathered}(x_{11} \vee x_{12} \vee x_{13} \vee \ldots x_{1i_1}) \wedge \\ (x_{21} \vee x_{22} \vee x_{23} \vee \ldots x_{2i_2}) \wedge \\ (x_{31} \vee x_{32} \vee x_{33} \vee \ldots x_{3i_3}) \wedge \\ \ldots \\ (x_{k1} \vee x_{k2} \vee x_{k3} \vee \ldots x_{ki_k})\end{gathered}$$

Where each x_{ij} is either a variable(z_t), or its negation ($\bar{z_t}$)

Output: An assignment for each z_t, such that the above boolean expression evaluates to true.

Solution: ILP formulation of any problem has three parts

- **Variables**

  Here we attach each an integer variable y_t to each boolean variable z_t, which represents the assignment of true or false.

  $$y_t = \begin{cases} 0 & \text{if false,} \\ 1 & \text{if true} \end{cases}$$

  $y_t = 1$ - y_t if z_t is in negated $0 \leq y_t \leq 1$

- **Constraints**

  Constraints will be given by the conditions necessary for each clause of the expression to be true.

  $$\begin{gathered}y_{11} + y_{12} + y_{13} + \ldots y_{1i_1} \geq 1 \\ y_{21} + y_{22} + y_{23} + \ldots y_{2i_2} \geq 1 \\ y_{31} + y_{32} + y_{33} + \ldots y_{3i_3} \geq 1 \\ \ldots \\ y_{k1} + y_{k2} + y_{k3} + \ldots y_{ki_k} \geq 1\end{gathered}$$

  Where y_{ij} represents the variable y_t attached to the corresponding z_t.

- **Cost**

  Here cost is immaterial as any feasible solution gives a satisfying assignment.

2 Formulating ILP for shortest path problem

Input: A directed graph with positive integer weights (w_{uv}) and two vertices s and t from its vertex set.

Output: Shortest/min weight path from s to t

- **Variables**

  Attach one integer variable to each edge x_{uv} which indicates whether the edge $(u, v)$is chosen or not.

  $$x_{uv} = \begin{cases} 1 & \text{if edge } (u,v) \text{ chosen,} \\ 0 & \text{o/w} \end{cases}$$

  $0 \leq x_{uv} \leq 1$

- **constraints**

  If there is a path from s to t in the graph then for any two partitions, such that one of them contains s and other contains t, there must be an edge leaving the partition containg s to the partition containing t ($\exists$ u $\in$ U and v $\in$ V-U s.t u and v are connected).

  $$\forall U \subset V, s \in U, t \notin U, \forall u \in U \text{ and } v \notin U \Sigma x_{uv} \geq 1$$

- **Cost** We want to minimize the weight of the path from s to t, ie the sum of all edges in the path should be minimum,

  $$\min(\Sigma x_{uv} w_{uv})$$

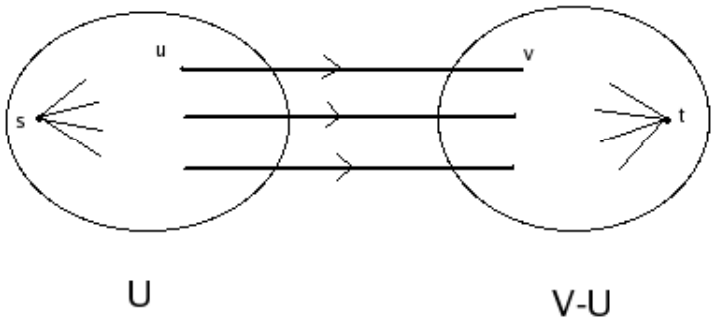


Figure 1: Example of a partition

This has an exponential number of constraints. In next class we will see how to write this using a polynomial number of constraints.

CS 435 : Linear Optimization — Fall 2006

Lecture 25: ILP formulations for SAT and shortest path problem (contd.)

Lecturer: *Sundar Vishwanathan* — Scribe: *Ajit Burad*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 SAT (*continuing with previous lecture*)

Input : Conjunction of Clauses
$(x_1 \vee x_2 \vee ...) \wedge (x_3 \vee x_4 \vee ...) \wedge (... \vee ...)$

Variables The literals, x_i's, in the clauses which are being assigned true or false are variables for the ILP.
But for the ILP we do not have True, False in our set so we will have one variable per boolean variable. For instance a variable y_i corresponding to each boolean variable x_i, So

x_i being false will be equivalent to y_i being zero.
x_i being true will be equivalent to y_i being one.

Here $0 \leq y_i \leq 1$, y_i is Integer forces y_i to take values either 1 or 0.

We will represent $\overline{y_i} = 1 - y_i$. So for each clause $(x_i \vee x_j \vee ...)$ the fact that one of the literals has to be true can be expressed as:

$$y_i + y_j + ... \geq 1$$

If we have $\overline{x_i}$, replace it with 1-y_i.
Here cost function is immaterial as for each set of y_i's we can get corresponding x_i's.

Here another approach (which does not work) could have been assigning :

$$y_i = +1 \quad \text{if} \quad x_i = \text{true}$$
$$y_i = \text{-1} \quad \text{if} \quad x_i = \text{false}$$

then -1 $\leq y_i \leq 1$ but $y_i = 0$ is not assigned anywhere.
Hence it is not possible to solve it this way.

2 Shortest Path

Input :
Directed Graph(G) (with each edge having positive integer weight)
Vertices s,t $\in$ V(G)

- Where s and t are start and end vertices respectively for the required path.
- V(G) is vertex set of G

Output : Shortest Path between s and t

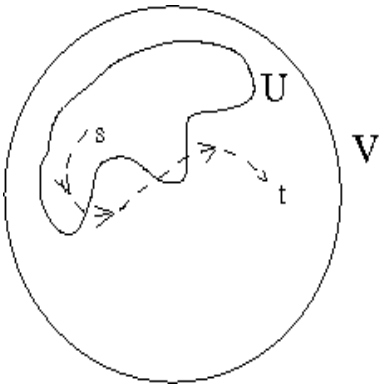


Figure 1:

To construct an ILP we first need to define variables. An assignment to the variables should tell us what the output should be.
For this problem we choose one variable (w_i) per edge.
$x_i = 1 \Rightarrow$ corresponding edge is chosen in shortest path from s to t.
$x_i = 0 \Rightarrow$ edge is not part of shortest path from s to t.

Cost : Let the cost of edge e be c_e
Cost of path $= \sum_e (x_e * c_e)$
Consider subsets of the vertex set which do not contain t but contain s. (See figure 1)
$\forall$ U $\subseteq$ V such that s $\in$ U, t $\notin$ U, at least one edge leaving U must be selected.

The following set of constraints force a path from s to t for every U $\in$ V(G), such that s $\in$U and v $\notin$ U,

$$\sum_{u \in U, v \notin U} X_{uv} \geq 1 \ ;$$

Note that we had an exponential number of costraints. There is another way to write the constraints, so that there are only a polynomial number of them.

$\sum_v x_{sv} = 1$, $\sum_u x_{ut} = 1$ (There are edges one starting from s and one terminating at t in our required path. We assume that there are no edges entering s and leaving t in G).
Also $\forall$ u $\neq$ s,t $\sum_w x_{wu} - \sum_v x_{uv} = 0$; $0 \leq Xuv \leq 1$.
Now we need to write dual for this and we will do the same thing as we did for SAT and we will end up with Dijkstra's.

CS 435 : Linear Optimization — Fall 2006

Lecture 25: ILP formulations for SAT and shortest path problem (contd.)

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Mohit Kansal*
Indian Institute of Technology, Bombay

In this lecture we will formulate Integer Linear Programs(ILPs) for some of the problems discussed in the tutorial.

1 SAT

A SAT problem can be defined as follows:

- **Input:** An expression of the form $(x_i \vee \overline{x_j} \vee ...) \wedge (... \vee ... \vee ...) \wedge ...$, where a variable x_k can take value T or F.
- **Output:** Assignment of TRUE/FALSE values to literals $x_i s$) for the expression to be true or NOT SOLVABLE if such an assignment is not possible.

Our task is to formulate this problem as an ILP:

- **Variables:** There will be one variable y_i per boolean variable x_i

$$y_i = \begin{cases} 0 & \text{if } x_i \text{ is False} \\ 1 & \text{if } x_i \text{ is True} \end{cases} \tag{1}$$

$0 \leq y_i \leq 1$, y_i is integral
This ensures that y_i is either 0 or 1.

Note: If we choose y_is to be -1 and 1 in place of 0 and 1, formulation in the above form is difficult.

- **Inequalities:** Consider a clause $x_i \vee x_j$ This will be true if either $x_i = 1$ or $x_j = 1$ or both. Clearly, the inequality will be $y_i + y_j \geq 1$. Similarly, for the clause $x_i \vee \overline{x_j}$, the inequality will be $y_i + 1 - y_j \geq 1$ or $y_i - y_j \geq 0$.
- **Inequality Matrix:** The inequality obtained from each clause will correspond to one row of the matrix.
- **Cost function:** Immaterial in this case as long as we can find feasible y_is satisfying $Ay \leq b$, $0 \leq y_i \leq 1$, y_i is integral.

2 Shortest Path

A Shortest Path problem can be defined as follows:

- **Input:** A Directed Graph with edge weights as positive integers and two special vertices s and t.
- **Output:** Set of edges which form a shortest path between s, t.

Our task is to formulate this problem as an ILP:

2.1 Formulation 1

- **Variables:** Clearly the output is a subset of the original edge set. So there will be one variable per edge(e) between two vertices u and v, x_{uv}

$$x_{uv} = \begin{cases} 0 & \text{if the edge from vertex u to vertex v is not chosen for the shortest path} \\ 1 & \text{if the edge from vertex u to vertex v is chosen for the shortest path} \end{cases} \tag{2}$$

$$0 \leq x_{uv} \leq 1,$$
$$x_{uv} \text{ is integral}$$

Define

$$c_{uv}\text{- cost of the edge from vertex u to vertex v}$$

- **Cost function:** Clearly the cost function in this case is

$$min \sum x_{uv} c_{uv}$$

Before we formulate the inequality let us state a lemma.

LEMMA 1 *Let V be a set of vertices and let s, t be the two vertices for which the shortest path has to be found. For every $U \subset V$ s.t. $s \in U$, $t \notin U$ at least one of the arcs coming out of U must be selected.*

- **Constraints:** From the above lemma we can see that the following inequalities force a path from s to t

$$\forall \text{ subsets U of V, s.t s } \epsilon \text{ U, t } \notin \text{ U}$$
$$\sum_{u \epsilon U, v \notin U} x_{uv} \geq 1$$

2.2 Formulation 2

In this formulation we use a polynomial number of constraints

- **Constraints:** Clearly there will be only one edge coming out of the starting vertex s in the shortest path and there will be only one vertex coming into the terminating vertex e. This is represented as follows

$$\sum_v x_{sv} - \sum_p x_{ps} = 1$$
$$\sum_u x_{ut} - \sum_q x_{tq} = 1$$

For all the other vertices (including the ones which are not in the shortest path) the number of edges (of the shortest path) coming in should be equal to the number of edges going out.

$$\forall u \neq s, e \ \sum_p x_{pu} - \sum_q x_{uq} = 0$$

$$x_{uv} = \begin{cases} 0 & \text{if the edge from u to v is not chosen for the shortest path} \\ 1 & \text{if the edge from u to v is chosen for the shortest path} \end{cases} \tag{3}$$

$$0 \leq x_{uv} \leq 1,\ x_{uv} \text{ is integral}$$

Clearly each of the vertices in the Vertex Set will contribute to one row of the matrix.

CS 435 : Linear Optimization — Fall 2006

Lecture 26: Primal-Dual Algorithm for Shortest Path Problem

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering

Scribe: *Arindam Bose*
Indian Institute of Technology, Bombay

1 Overview

In the previous lecture, we saw the formulation of the Integer Linear Program for the shortest path algorithm. In this lecture we formulate and solve the dual.

2 The formulation of the shortest path problem

Input:

A directed graph with positive integer weights, $s, t \in V$

Output:

Shortest path from s to t

Variables:

We choose one variable per edge, x_e. If x_e is picked, $x_e = 1$ else $x_e = 0$. w_e is the value of the weight of edge e

Cost:

The cost function is min $\sum_{u,v} x_{uv} w_{uv}$

Constraints:

1. $\sum_u x_{su} - \sum_v x_{vs} = 1$ (The difference in the number of edges leaving s and entering into s is 1)
2. $\sum_v x_{vt} - \sum_u x_{tu} = 1$ (The difference in the number of edges entering into t and leaving t is 1)
3. $\forall p \in V - \{s, t\}$ $\sum_q x_{pq} - \sum_r x_{rp} = 0$ (For all other vertices, edges leaving them is equal to the edges entering into them)
4. $0 \leq x \leq 1,$ x is integer. We drop integer and upper bound constraints on x as in the case of MST.

3 The Dual

The resulting dual of the primal will have one variable for each vertex in the graph.

$$\max y_s - y_t$$
$$\forall u, v \; y_u - y_v \leq w_{uv}$$

Algorithm

We try to devise a strategy to solve for the values of y subject to the constraint that $y_u - y_v \leq w_{uv}$ and maximising the objective function $y_s - y_t$. The general scheme while trying to go about finding a solution would be as follows:

1. Start with a feasible dual solution.
2. Always maintain dual feasibility.
3. Try and improve.

Now, if y is a solution, $y \pm \delta$ will also be a solution since the constraints would be satisfied. So, the value of y_t is arbitrarily set to 0. Further, we start by setting $y_u = 0 \, \forall u$ and we try to improve the solution thereupon. We will facilitate this process using the marbles and strings analogy described below.

Marbles and Strings analogy

Consider the vertices of the graph as marbles and the edges as inextensible strings (wound to the marbles) with lengths equal to the value of the weights of the corresponding edges. The value y_u for vertex u is interpreted as the distance of node u from the node t ($y_t = 0$, as assumed before). We start by assigning the values of $y_u = 0 \, \forall u$, the situation which is represented by all the marbles being at the position of marble t. Thus, all the strings are slack at this initial point.

Now, maintaining the value of y_t at 0 (keeping the position of marble t fixed), the values of all other $y's$ are raised *simultaneously* (this ensures that the dual feasibilty is maintained). This would be analogous to pulling all the marbles simultaneously (in a straight line) away from marble t. This is done till one of the strings joining marble t becomes taut. Suppose it is the string that joins marble m to t that has become taut. At this point, $y_m = w_{mt}$, and we can no longer pull this particular marble away from t if dual feasibilty is to be maintained. So we fix the position of this marble m now and start pulling all the other marbles till another string becomes taut. We again fix the position of that corresponding marble and keep pulling the other marbles. This process is continued until we find that in due time one of the strings attached to marble s has become taut at which point that has to be fixed. At this point, all the strings corresponding to the edges that belong to the shortest path are taut. Among the edges that have become tight, there are some edges which are not part of the shortest path. These can be removed using the *reverse delete procedure* described below.

Reverse Delete Procedure:

Consider edges in reverse order of them becoming tight. Delete the edge and now check if a path exists from s to t. If a path does exist, then this edge is not a part of the shortest path and it should be removed from the solution set.

Proof that the value of primal optimal is equal to the value of dual optimal:

$$\sum_{(u,v)\, chosen} w_{uv} = \sum_{(u,v)\, chosen} (y_u - y_v) = y_s - y_t$$

CS 435 : Linear Optimization Fall 2006

Lecture 26: Primal-Dual algorithm for shortest path problem

Lecturer: *Sundar Vishwanathan* Scribe: *Pragya Goyal*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Recap of last lecture

In last class we had formed the ILP for the Shortest Path problem as follows:

$$\min \sum_e w_{uv} x_{uv} \tag{1}$$

$$\sum_u x_{su} = 1 \tag{2}$$

$$\sum_v x_{vt} = 1 \tag{3}$$

$$\forall p \sum_q x_{pq} - \sum_r x_{rp} = 0 \tag{4}$$

$$1 \geq x_{uv} \geq 0 \tag{5}$$

$$x_{uv} \text{is an integer} \tag{6}$$

Expression(1)is the summation of weight of all the edges selected, where x_i takes value 1 if edge (u, v) is selected otherwise remains 0. Equations (2) and (3), represent the constraints that in the shortest path the source s must have exactly one edge leaving it and destination t must have exactly one edge entering into it, respectively. Equation (4) is written to satisfy the constraint that all the intermediate vertices (i.e. apart from s and t) in the path must have one edge entering and one edge leaving them. Inequality(5) is to ensure that x_{uv} takes values in $[0, 1]$. To reduce the problem to LP we drop the $x_{uv} \leq 1$ constraint. This can be done because equalities (2) and (3) along with x_{uv} constraint ensure that each x_{uv} does not take value greater than 1.

Now let us look at the dual. We will be solving the dual and will observe that it is Dijkstra's algorithm.

2 Solving the Dual

In the dual all the constraints of Primal are represented by variables. Here we have one variable for each vertex. Hence dual can be represented as:

$$\max\ y_s - y_t \tag{7}$$

$$\forall u, v \qquad y_v - y_u \leq w_{uv} \tag{8}$$

Now let us try solving the dual. Here y_i can be thought of variable corresponding to ith vertex. To maximize $y_s - y_t$ let us begin with assuming $y_t = 0$. Keeping y_t fixed at 0 we raise all other variables by 1 at a time until for atleast one of them, say y_1, $y_1 - y_t = w_{1t}$. at this point stop increasing the value of y_1 and fix it to the last reached value. Repeat the procedure keeping y_t and y_1 fixed now. Include any

new vertex y_2 as soon as $y_2 - y_1$ equals w_{12}. We iterate over all vertex variables until we reach y_s. We stop the iteration when y_s is reached.

Marble-string analogy:
The procedure is similar to fixing a marble tied to many strings and then pulling the strings apart. Constraint being limited length of the strings. Fixed marble is a vertex in this case, strings are the edges entering the vertex and string lengths correspond to edge weights. We can observe here that the string with least length (which corresponds to least weight edge entering fixed vertex) will become taut first. Becoming taut means equality is satisfied in the constraint. Select this edge. Now fix the vertex from which this edge emanates. Apply similar raising method for this vertex. We iterate over the vertices until we reach the source vertex. The edges selected at each iteration form the shortest path from s to t. This algorithm is **Dijkstra's algorithm**.

2.1 Reverse Delete

The edges we selected in the above procedure contain the shortest path along with some additional edges. To remove those edges we apply the procedure of *Reverse Delete*. This is done by starting deletion of maximum weight edges, each time assuring that a path exists from s to t. As soon as we reach an edge, deleting which leaves no path from s to t, we stop the procedure. The path left (including this last egde seen) is the required shortest path.

CS 435 : Linear Optimization — Fall 2006

Lecture 27: Primal-dual algorithm for matching in bipartite graphs

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Mohit Gogia*
Indian Institute of Technology, Bombay

1 Primal Dual Method

Primal	Dual
$\max c^T x$	$\min y^T b$
$Ax \leq b$	$A^T y = c$
	$y \geq 0$

Now consider the following unweighted LP for the above primal.

$$\max c^T y$$
$$A'y \leq 0$$

Let $X' = X + \epsilon y$
where X satisfies the primal and ϵy satisfies the above unweighted LP

Then X' satisfies primal as well and the value of its objective function is greater than that for X. This gives us a method for solving the primal. We keep on increasing X_i's in the primal by y till the value of $c^T y$ becomes 0. In the next section, we frame bipartite matching as an ILP.

2 ILP for Bipartite Matching

- **Primal formulation** Let the variable X_{uv} be defined as follows

$$X_{uv} = \begin{cases} 0 & \text{if edge (u,v) is not matched,} \\ 1 & \text{if edge (u,v) is matched} \end{cases}$$

  Then the ILP can be formulated as

$$\max \sum X_{uv}$$
$$\forall u \sum X_{uw} \leq 1$$
$$\forall v \sum X_{pv} \leq 1$$
$$\forall u, v \; 0 \leq X_{uw} \leq 1, X_{uv} \text{ is integral}$$

  Now, as before, we remove the conditions $X_{uw} \leq 1$ and X_{uv} is integral to get LP correspoding to the above ILP[1].

- **Dual formulation** The dual will have variables Y_u and Z_v corresponding to each vertex (Figure 1).

[1]Even after removing the integrality constraints, the solution after relaxation is same as that without relaxation

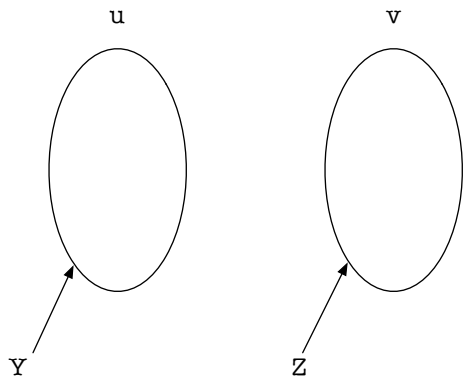


Figure 1: Variables in dual correspond to vertices of the two partitions

Hence the dual can be formulated as

$$\min \sum Y_u + \sum Z_v$$
$$Y_p + Z_q \geq 1 \text{ , } Y_p, Z_q \geq 1 \text{ for all edges (p,q)}$$

It can be clearly seen that the dual corresponds to finding the minimum vertex cover of the graph.

DEFINITION 1 *A vertex cover of an undirected graph is a subset of vertices of the graph which contains at least one of the two endpoints of each edge.*

Next we see the how to find a minimum vertex cover for a graph using a maximum mathcing for that graph.

3 Minimum Vertex Cover from Maximum Matching

If V denotes a vertex cover and M a maximum matching for the same graph, because any vertex cover has to include at least one endpoint of each matched edge, so

$$|V| \geq |M|$$

Also it can be shown that *size of minimum vertex cover for a graph is equal to the size of its maximum matching.*

Now, given a maximum matching how do we find a vertex cover?

Let G(V,E) be a graph and M be a maximum matching for the graph. Let V' be the set of those vertices in M which are at odd distance from unmatched vertices in G wrt M. Then V' is a minimum vertex cover for G

An intuitive way to understand the above is shown in Fig 2.

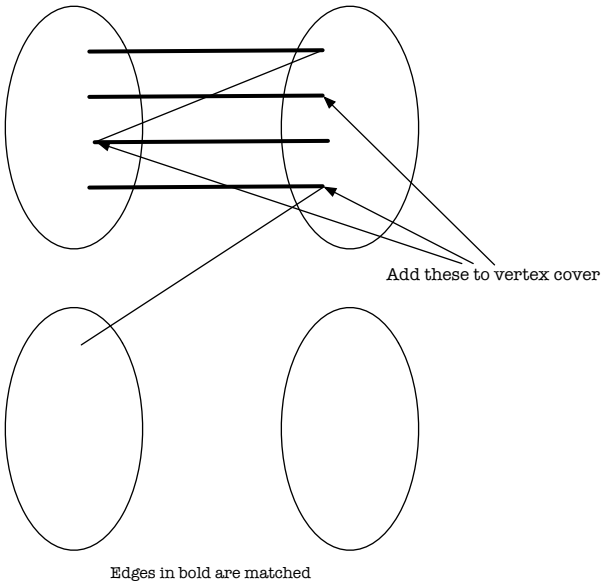


Figure 2: Minimum Vertex Cover from Maximum Matching

In the next lecture, we will look at the case of mathching in bipartite graphs with weights associated with edges.

CS 435 : Linear Optimization Fall 2006

Lecture 27: Primal-dual algorithm for matching in bipartite graphs

Lecturer: *Sundar Vishwanathan* Scribe: *Parthasarathi Roy*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 The Primal-Dual method

$$\begin{aligned} \text{P}: \quad \max\ & c^T x \\ \text{s.t. } & Ax \leq b \end{aligned} \qquad \begin{aligned} \text{D}: \quad \min\ & y^T b \\ \text{s.t. } & A^T y = c \\ & y \geq 0 \end{aligned}$$

Now consider the following unweighted LP for the above primal,

$$\begin{aligned} &\max\ c^T y \\ &\text{s.t. } A'y \leq 0 \end{aligned}$$

Let X satisfy the primal and Y satisfy the above LP. Let $X' = X + \epsilon Y$. Then X' satisfies the primal and the value of its objective function is greater than that for X. This gives us a method for solving the primal. We keep on increasing X_i's in the primal by y till the value of $c^T y$ becomes 0.

2 Matching in Bipartite Graphs

2.1 The Primal

Variables: One variable is used for each edge. An edge is either in or out in the bipartite matching.

X_{uv} represents edge $\{u, v\}$.

Cost: $max \sum_{uv} X_{uv}$

Constraints: For every vertex, there is at most one edge incident on it.

$$\forall u \in U, \sum_{w} X_{uw} \leq 1 \tag{1}$$

$$\forall v \in V, \sum_{p} X_{pv} \leq 1 \tag{2}$$

$$\forall u, v, 0 \leq X_{uv} \leq 1, X_{uv} \text{ is integral} \tag{3}$$

2.2 The Dual

For the dual, the number of variables is equal to the number of vertices in the bipartite graph.

Cost: $min \sum_u Y_u + \sum_v Z_v$. The first term in the sum stands for vertices of set U, while the second term stands for vertices in the set V.

Constraints: For edge $\{p, q\}$,

$$Y_p + Z_q \geq 1 \tag{4}$$

$$Y_p, Z_q \geq 0 \tag{5}$$

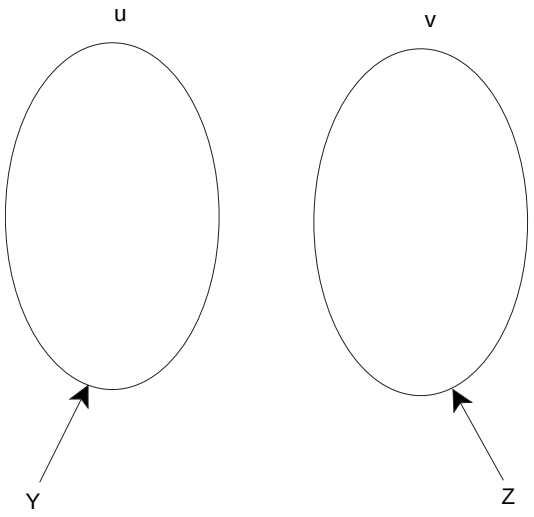


Figure 1: Variables in the dual represent the vertices

This represents the fact that for each edge, one of the vertices must be picked. The dual of the maximum matching problem in bipartite graphs is the minimum vertex cover problem.

DEFINITION 1 *A **vertex cover** is a set of vertices V in an undirected graph where every edge connects at least one vertex in the set V.*

3 Minimum vertex cover from the maximum matching in bipartite graphs

If V is a vertex cover and M is a maximum matching for a bipartite graph,

$$|V| \geq |M| \tag{6}$$

This is because, any vertex cover has to include at least one end-point of an edge of a maximum matching.

In any bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover.

Finding a minimum vertex cover from a maximum matching:

Let $G(V, E)$ be a graph and M be a maximum matching for the graph. Let V' be the set of those vertices in M which are at odd distance from unmatched vertices in G. Then V' is a minimum vertex cover for G (Refer Fig. 2).

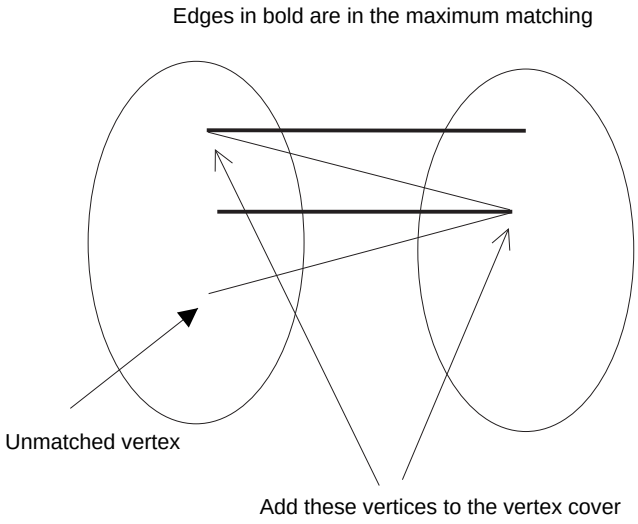


Figure 2: Minimum vertex cover from maximum matching

CS 435 : Linear Optimization Fall 2006

Lecture 28: Primal-dual algorithm for matching in bipartite graphs (contd.)

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Abhiroop Medhekar*
Indian Institute of Technology, Bombay

1 Recap

In the previous lecture, we discussed the ILP formulation for *unweighted*[1] bipartite matching. It was observed that the dual for this particular case corresponds to the *minimum vertex cover* for the graph.

In this lecture we will talk about the more general case of matching in a weighted bipartite graph.

2 ILP formulation of Matching in a Weighted Bipartite Graph

The input is a graph with each edge having a positive weight W_{uv}.

Definition 1 *A maximum weighted bipartite matching is defined as a perfect matching where the sum of the values of the edges in the matching have a maximal value.*

As we will later see, the size of such a matching would be n^2. Note that, if the graph is not complete bipartite, missing edges are inserted with value zero.

The variable X_{uv} is defined as,

$$X_{uv} = \begin{cases} 0 & \text{if edge (u,v) belongs to the matching,} \\ 1 & \text{if edge (u,v) does not belong to the matching} \end{cases}$$

Again, as we have added zero-weighted edges to *complete* the graph, we assert that the summations $\forall u \sum X_{uw}$ and $\forall v \sum X_{pv}$ be *exactly* equal to 1.

Therefore, the ILP (*primal*) for this problem is:

$$\max \sum W_{uv} X_{uv}$$

$$\forall u \sum X_{uw} = 1$$

$$\forall v \sum X_{pv} = 1$$

$$\forall u, v \; X_{uv} \geq 0, X_{uv} \text{ is integral}$$

The resultant dual is:

$$\min \sum Y_u + \sum Z_v$$
$$Y_u + Z_v \geq W_{uv}$$

[1] Unweighted: The weight of each edge in the graph is 1
[2] where n is the number of vertices in the graph

3 Algorithm to find Maximum Weighted Bipartite Matching

- **Step 1 - Initialization:** For all vertices v, initialize Y_v and Z_v to the highest weight edge incident on it.
- **Step 2 - Iteration:** Now, locate all edges *(u,v)* for which the equality

$$Y_u + Z_v = W_{uv}$$

holds.
Let E represent the set of such edges. Then the following dual can be formulated for edges in E,

$$\begin{gathered}\text{for each edge (u,v) in E} \\ \min \sum Y'_u + \sum Z'_v \\ Y'_u + Z'_v \geq 0\end{gathered}$$

This version of the dual is unweighted, and hence easier to solve. In the optimum solution, we have $\sum Y'_u + \sum Z'_v = 0$.

In the next lecture we complete this algorithm.

CS 435 : Linear Optimization — Fall 2006

Lecture 28: Primal-dual algorithm for matching in bipartite graphs (contd.)

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Shakeb Sagheer*
Indian Institute of Technology, Bombay

1 Recap of last lecture

In the previous lecture we looked at the ILP for matching in unweighted bipartite graphs. Let the variable X_{uv} be defined as follows

$$X_{uv} = \begin{cases} 0 & \text{if edge (u,v) is not matched,} \\ 1 & \text{if edge (u,v) is matched} \end{cases}$$

Then the primal and dual formulations of the ILP are

Primal	Dual
$\max \sum X_{uv}$	$\min \sum Y_u + \sum Z_v$
$\forall u \sum X_{uw} \leq 1$	$Y_p + Z_q \geq 1$, $Y_p, Z_q \geq 0$
$\forall v \sum X_{wv} \leq 1$	
$\forall u, v\ X_{uv} \geq 0$	

It was observed that the above dual formulation corresponds to finding the minimum vertex cover of the graph.

2 ILP for Matching in Weighted Bipartite Graph

In weighted matching problem, a number $w_{uv} \geq 0$ is associated with each edge of the graph, called the *weight* of that edge and we are suppposed to find a matching with the largest possible sum of weights. Thus our task is to maximize $\sum w_{uv} X_{uv}$.. The constraints remain as such.

The ILP, therefore, can be formulated as

$$\max \sum w_{uv} X_{uv}$$

$$\forall u \sum X_{uw} \leq 1 \tag{1}$$

$$\forall v \sum X_{wv} \leq 1 \tag{2}$$

$$\forall u, v\ X_{uv} \geq 0$$

The dual of above primal can be formulated as

$$\min \sum Y_u + \sum Z_v$$
$$Y_u + Z_v \geq w_{uv}\ ,\ Y_u, Z_v \geq 0 \text{ for all edges (u,v)}$$

It can be seen that in the above problem, we can do away by adopting the convention that the underlying graph is always complete, and letting the weights of the missing edges be equal to zero. Thus after adding appropriate zero weight edges, the inequations (1) and (2) can be replaced by following equations.

$$\forall u \sum X_{uw} = 1$$
$$\forall v \sum X_{wv} = 1$$

Since we have dropped the inequalities in the primal the inequalities $Y_u, Z_v \geq 0$ can be dropped in the dual.

Hence the primal and dual formulations of the ILP are

Primal	Dual
$\max \sum w_{uv} X_{uv}$	$\min \sum Y_u + \sum Z_v$
$\forall u \sum X_{uw} = 1$	$Y_u + Z_v \geq w_{uv}$
$\forall v \sum X_{wv} = 1$	
$\forall u, v \; X_{uv} \geq 0$	

3 Algorithm for finding Maximum Matching

- **Initialization** For each vertex initialize the variables Y_u and Z_v to the highest weight edge incident on it
- **Iterative Step** Consider all u,v's for which the equality

  $$Y_u + Z_v = w_{uv}$$

  holds. Let E' be the set of edges for which we have the above equality, then for these edges the following dual can be formulated,

  $$\text{for each edge (u,v) in E'}$$
  $$\min \sum Y'_u + \sum Z'_v$$
  $$Y'_u + Z'_v \geq 0$$

  This is the unweighted version of the dual and is easier to solve. When we have an optimal solution the minimum value of $\sum Y'_u + \sum Z'_v$ is 0 and this cannot be furthur improved upon.

  Our task is to find a matching of size n in E'. If we can find one then we are done.

  In the next lecture we will discuss how to find such a matching.

CS 435 : Linear Optimization — Fall 2006

Lecture 29: Primal-dual algorithm for matching in bipartite graphs (contd.)

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering
Scribe: *Abhimanyu Rawal*
Indian Institute of Technology, Bombay

Problem

In weighted matching problem a number $w_{uv} \geq 0$ is associated with each edge of the graph, called the *weight* of that edge and we are suppposed to find a matching with the largest possible sum of weights.

Let the variable X_{uv} be defined as follows:

$$X_{uv} = \begin{cases} 0 & \text{if edge (u,v) is not matched,} \\ 1 & \text{if edge (u,v) is matched} \end{cases}$$

Our task is to maximize $\sum w_{uv} X_{uv}$.

The Primal for matching in weighted bipartite graph is :

$$\begin{aligned} & max \sum W_{uv} X_{uv} \\ & \sum X_{uv} = 1 \\ & X_{uv} \geq 0 \end{aligned} \tag{1}$$

The Dual for the above is :

$$\begin{aligned} & min \sum_u Y_u + \sum_v Z_v \\ & \forall\ edge\ \{u,v\} \quad Y_u + Z_v \geq W_{uv} \end{aligned} \tag{2}$$

Algorithm

1. We start with any feasible dual solution, which may not necessarily be the optimal solition.
2. Let E' be the set of those edges for which the inequalities are tight. For these edges the dual is rewritten as :

$$\begin{aligned} & min \sum Y'_u + \sum Z'_v \\ & \forall\ edge\ \{u,v\}\ \in\ E', \quad Y'_u + Z'_v \geq 0 \end{aligned} \tag{3}$$

Clearly this problem is unbound, hence to make it bounded, the following equations need to added to the system :

$$\begin{aligned} -1\ \leq\ Y'_u\ \leq\ 1 \\ -1\ \leq\ Z'_v\ \leq\ 1 \end{aligned} \tag{4}$$

3. The optimal solution to the above system is the point at which the costs of the primal feasible and dual feasible coincide.

   Treating the above equation as primal, B = 0. Hence, the dual of the above dual thus is:

$$\begin{aligned} & max\ (0) \\ & \forall\ edge\ \{u,v\}\ \in\ E',\ \sum X_{uv} = 1 \\ & X_{uv}\ \geq\ 0 \end{aligned} \tag{5}$$

   This equation will give a matching of size n in E' by the following algorithm repeated recursively :

   - If the current matching is of size n, then the dual is optimal and we have found the matching of maximum size.
   - Else, if the Dual is not optimal, then the matching in E' is less than size n.
   - Let the size of the Dual be t. Then, in the vertex cover comprising of the t vertices, we increase the value of all the vertices by δ. And for all the remaining vertices (not in the vertex cover), we decrease the size by δ. Since there are 2n vertices and t<n, the total cost will decrease.

Lecture 29: Primal-dual algorithm for matching in bipartite graphs (contd.)

Lecturer: *Sundar Vishwanathan* Scribe: *Gaurav Meena*
Computer Science & Engineering Indian Institute of Technology, Bombay

1. Primal

$$\begin{aligned} & max \sum W_w X_{uv} \\ & \sum X_{uv} = 1 \\ & X_{uv} \geq 0 \end{aligned} \tag{1}$$

2. Dual

$$\begin{aligned} & min \sum_u Y_u + \sum_v Z_v \\ & \forall \; edge \; \{u,v\} \quad Y_u + Z_v \geq W_{uv} \end{aligned} \tag{2}$$

Algorithm

1. Start with any feasible dual solution.
2. Let E' be the set of edges for which the inequalities are held.

$$\begin{aligned} & min \sum Y'_u + \sum Z'_v \\ & \forall \; edge \; \{u,v\} \; \in \; E', \quad Y'_u + Z'_v \geq 0 \end{aligned} \tag{3}$$

$$\begin{aligned} -1 \; \leq \; Y'_u \; \leq \; 1 \\ -1 \; \leq \; Z'_v \; \leq \; 1 \end{aligned} \tag{4}$$

   Equation 4 is just to make sure that the LP is bounded.

3. If we find a primal feasible and dual feasible with the same cost then we are done. It is the optimal solution.

   The dual of equation 3 :

$$\begin{aligned} & max \; 0 \\ & \forall \{u,v\} \; \in \; E', \; \sum X_{uv} = 1 \\ & X_{uv} \; \geq \; 0 \end{aligned} \tag{5}$$

   Equation 5 must be matching of the size n in E'.

4. If we find a matching of maximum size then we are done.

5. If we know that the Dual is not optimal, then it means that matching in E' is not of size n max and we need to improve the solution.

6. What if the matching is not of size n in E'? How to improve?

   Suppose the matching is of size t then there is a vertex cover of size t. In this vertex cover, for all the vertices increase the value of the vertex by δ. For vertices not in the vertex cover, decrease the value by δ.

   Since there are total 2n vertices and t $<$ n so the total cost will decrease.

   We increase with best δ we can, such that one more edge tighten and another edge comes in E'.

CS 435 : Linear Optimization Fall 2006

Lecture 30: Primal-Dual Algorithm for Matching in Bipartite Graphs (Conclusion)

Lecturer: *Sundar Vishwanathan* Scribe:*Ankit Aggarwal*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Primal-Dual Method

Primal	Dual
$\max \sum w_{uv} x_{uv}$	$\min \sum Y_u + \sum Z_v$
$\sum x_{uv} = 1$	$\forall \{u, v\},\ y_u + z_v \geq w_{uv}$
$x_{uv} = 0$	

2 Procedure

- If there doesn't exist any constraint such that the equality holds in it, we decrease all dual variables by some δ, such that atleast one of the constraints satisfy equality.
- Let $E^{'}$ be the edges such that $y_u + z_v = w_{uv}$
- Now solve,

$$\min \quad \sum y'_u + \sum z'_v \tag{1}$$

$$\forall u, v, \exists E, \quad y'_u + z'_v \geq 0 \tag{2}$$

- Find a maximum matching in this graph using the "original" algorithm.
- Now, suppose there are 2n vertices, then there are 2 cases:
  - If that matching has size n, the weight of the matching is

$$\sum_{(u,v) \text{in the matching}} w_{uv} = \sum y_u + \sum z_v = \text{Dual Cost} \tag{3}$$

  - If the matching is of size t, which is less than n, find a vertex cover S of size t.

- Vertex Cover : **(Algorithm)**
  - Decrease the vetex label in (V - S) by δ
  - Increase the vertex label in S by same δ
  - There is a net decrease in the cost.
  - Take those vertices which are at odd distances from unmatched vertices (see in Figure 1.

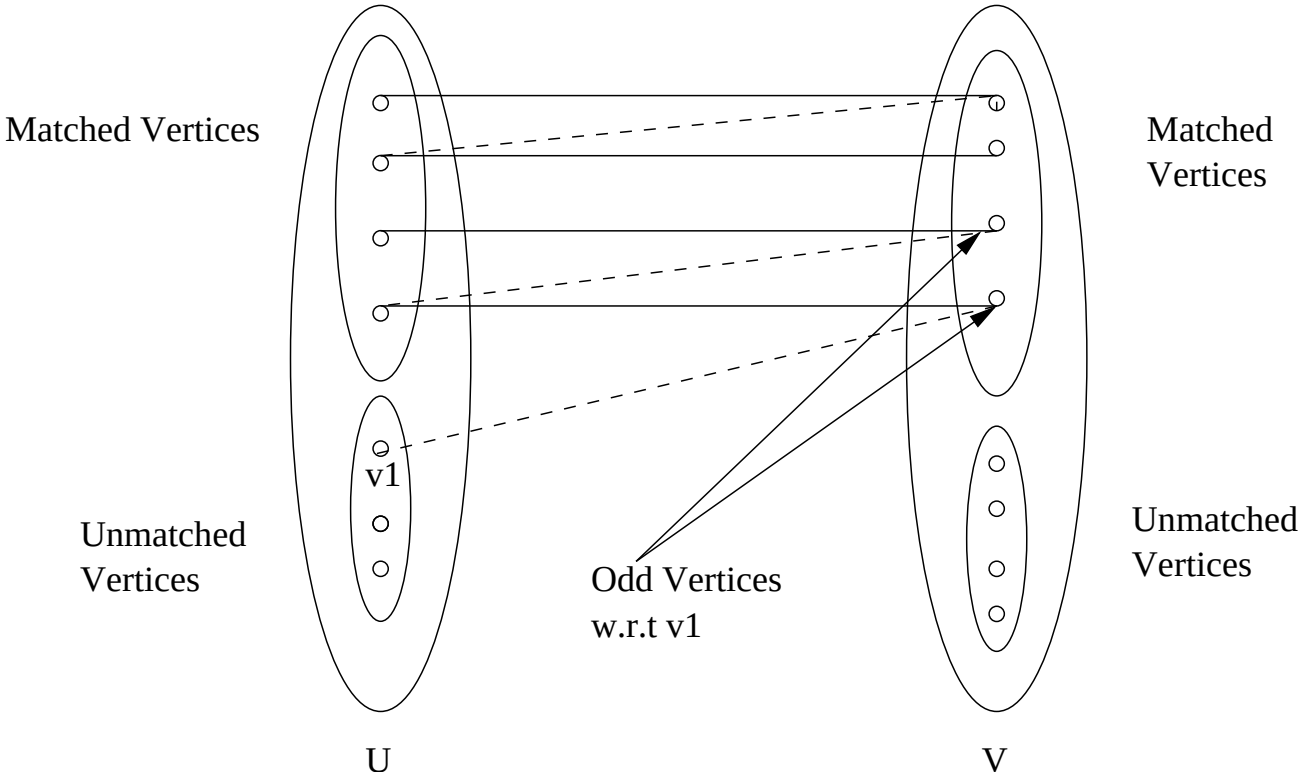


Figure 1: Showing the odd distant vertices w.r.t. unmatched vertex v1.

  - After this, there can be some matched edges for which neither vertices are taken.

- We also show that these decrease and increase in labels don't violate the constraints:

$$y_{\text{new}} = y_{\text{old}} + y' \tag{4}$$

$$z_{\text{new}} = z_{\text{old}} + z' \tag{5}$$

  - Because y_{old} and z_{old} are equal to corresponding w_{uv}, and y' and z' are positive, hence y_{new} and z_{new} are greater than equal to w_{uv}.

3 Time Complexity

Two Cases:

1. Either we increase the size of matching, or
2. we increase the size of the component.

So, by the end, atmost n^2 steps will be taken. Hence, the algo is $O(n^2)$, i.e., polynomial.

4 Few things to take over

1. At every step, we need not find maximal matching (because we could use the augmenting path from previous step).
2. This can be used for unweighted case also. See, how this looks overall.

CS 435 : Linear Optimization Fall 2006

Lecture 31: Network Flows

Lecturer: *Sundar Vishwanathan* Scribe:*Anshu Agrawal*
Computer Science & Engineering Indian Institute of Technology, Bombay

In this lecture we introduced network flows and primal-dual algorithm for same.

1 Definition

A network flow is an assignment of flow to the edges of a directed graph with each edge havinf a capacity, such that the amount of flow along an edge doesnot exceed its capacity.

The Network problem can also be seen as one wants to send as many trucks from s to t. Trucks can't stop at any node. What is the number of trucks on each of these roads?

Hence, we have to solve the following problem.

2 Input

- Directed Graph G(V,E) with nodes V and edges E.
- Positive integer capacity on each edge c_{uv}.
- Two special vertices s(source) and t(sink).

NOTE: Parallel edges are disallowed for convinience. They can be easily converted into non-parallel edges by adding one vertex between them.

3 Output

For each edge uv, flow at that edge f_{uv} such that

$$\sum_u f_{su} \text{ is maximum}$$

A network flow has following three properties for all nodes u and v :

$$\forall u, v \ f_{uv} \leq C_{uv}$$

i.e. the flow along an edge cannot exceed its capacity.

$$\forall u, v \ f_{uv} + f_{vu} = \ 0$$

The net flow from u to v must be opposite of the net flow from v to u.

$$\forall u \neq s, t \ \sum_v f_{uv} = 0$$

i.e. The net flow to a node is zero.

4 Algorithm

Here, it is better to solve the primal itself than the corresponding dual.

4.1 Initialization

Initialize :

$$\forall u, v \ f_{uv} = 0$$

Now, we need a code that can run recursively, resulting in improvement of the above solution till we find the optimal. The Primal-Dual algorithm will ne helpful in finding that.

4.2 Recursion

New network is :

$$max \sum_u f'_{su}$$

$$\text{s.t. } \forall u, v \text{ where } (\ f_{uv} = C_{uv}\ )\ f'_{uv} \leq \ 0 \tag{1}$$

$$\forall u, v \ f'_{uv} + f'_{vu} = \ 0 \tag{2}$$

$$\forall u \neq s, t \ \sum_v f'_{uv} = 0 \tag{3}$$

Now the relation between original problem and this is :

$$\forall u, v \ f^{old}_{uv} + \epsilon f'_{uv} = f^{new}_{uv}$$

Our aim is to keep raising ϵ until we reach an equality of the form $f^{new}_{uv} = C_{uv}$

NOTE: If $f_{uv} \leq 1$, then the problem will be just to find is there a path from s to t.

4.3 Example

Let us now consider the case given in Fig. 1(a). Let all capacities be 1.
Now let us assume that current flow is ABCD (i.e. $f_{AB} = f_{BC} = f_{CD} = 1$). Then the new

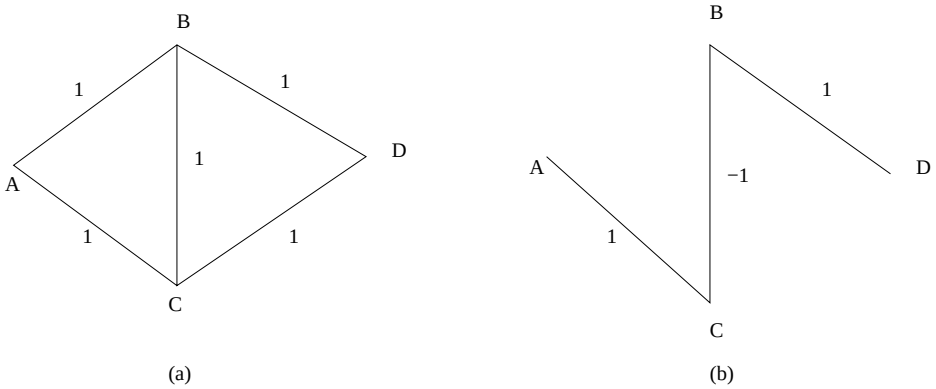


Figure 1: Network with all capacities equal to 1

network is Fig. 1(b). Here $-1 \leq 1$, hence the edge BC will be there in new network.

5 Next class

We will continue with this in next class and look at the algorithm more closely.

CS 435 : Linear Optimization — Fall 2006

Lecture 31: Network Flow

Lecturer: *Sundar Vishwanathan* — Scribe: *Amit Singh*
Computer Science & Engineering — Indian Institute of Technology, Bombay

1 Inroduction

Let G = (V,E) be a directed graph.For all edges e,define capacity $C_e \geq 0$.Let the variable f_e denote amount which flows along edge e.

$$\textit{feasibility: } f_e \leq C_e$$

In addition ,There will always be one vertex s with the property that all arcs containing s are directed away from s and one vertex t with the property that all arcs containing t are directed toward t .

constraints over networks.

- For each edge e ,the amount of flow f which flows along edge must be less than or equal to capacity of that edge.

  feasibility: $f_e \leq C_e$

- *Conservation rule* for Network flows,

  For each u,v $\in$ V, $f_{uv} + f_{vu} = 0$

  At each vertex u,v $\neq$ s,t $\sum f_{uv} = 0$

Our problem is to calculate maximum value of a flow for a given network.

$$\max \sum f_{su}$$

2 Primal-Dual Method for Max Flow

Step1: Initialize with any feasible dual solution.Let $f_{uv} = 0$ for all (u,v) $\in$ E

Step2: for all edges e for which equality

$$f_e = C_e$$

holds.

Let E' is set of such edges.Then DRP becomes,

$$\max \sum f'_{su}$$
$$f'_{uv} \leq 0 \text{ , for each edge (u,v)} \in \text{E'}$$
$$f'_{uv} + f'_{vu} = 0$$
$$\sum f'_{uv} = 0 \text{ for all u} \neq \text{s,t}$$
$$f_{uv} \leqslant 1$$

We can observe that (DRP) has the following interpretation. Find a path from s to t (with a flow of value 1) that uses only the following arcs in the following ways: saturated arcs in the backward direction; arcs with zero flow in the forward direction; and other arcs in either direction.We need to find a path in the residual graph.

CS 435 : Linear Optimization — Fall 2006

Lecture 32: Network Flows (contd.)

Lecturer: *Sundar Vishwanathan* — Scribe: *Ankit Jain*
Computer Science & Engineering — Indian Institute of Technology, Bombay

In this lecture we continue with the primal-dual algorithm for network flows.

1 Primal Dual Algorithm

The network flow problem as defined in the previous lecture is as follows :

$$max \sum_u f_{su}$$

$$\text{s.t. } \forall u, v \; f_{uv} \leq C_{uv} \tag{1}$$

$$\forall u, v \; f_{uv} + f_{vu} = 0 \tag{2}$$

$$\forall u \neq s, t \; \sum_v f_{uv} = 0 \tag{3}$$

where f_{uv} is the flow from vertex u to vertex v
and C_{uv} is the (non negative) capacity of the link u - v

In this case, we will solve the primal itself rather than the corresponding dual.

1.1 Initialization

We begin with a feasible solution.
$\forall u, v \; f_{uv} = 0$

Now, we need a piece of code that we can run again and again recursively, improving the above solution till we find the optimal. The Primal-Dual algorithm helps us find just that.

1.2 Recursion

The Primal Dual Algorithm

$$max \sum_u f'_{su}$$

$$\text{s.t. } \forall u, v \text{ where } (\; f_{uv} = C_{uv} \;) \; f'_{uv} \leq 0 \tag{4}$$

$$\forall u, v \; f'_{uv} + f'_{vu} = 0 \tag{5}$$

$$\forall u \neq s, t \; \sum_v f'_{uv} = 0 \tag{6}$$

This is related to the original problem by the following equation

$$\forall u, v \; f^{old}_{uv} + \epsilon f'_{uv} = f^{new}_{uv} \tag{7}$$

Our aim is to keep raising ϵ until we reach an equality of the form $f^{new}_{uv} = C_{uv}$

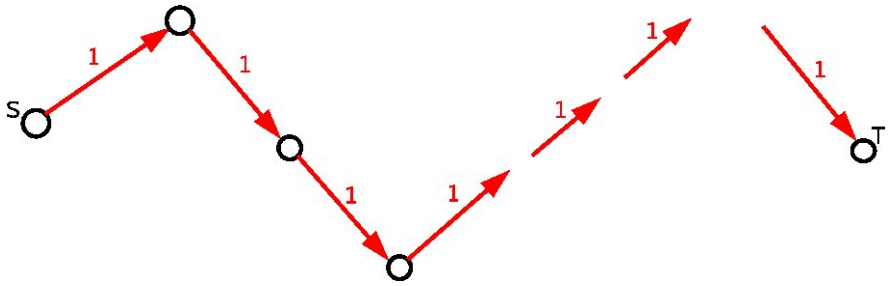


Figure 1: Path from s to t

$$\text{If } \sum_u f'_{su} = 1$$

the network flow must be of the form as shown in Figure1 since we have a net flow of 1 leaving from s which cannot disappear at any node except the destination.
All these edges have $f_{uv} < C_{uv}$

So all we need to do is to find one such path and raise the flow till one of the paths reaches its capacity. For any path, maximum raise possible $= C_{uv} - f_{uv}$ This tells us the new capacity if all the edges in the graph. So we construct a graph on same set of vertices with capacity $= C_{uv} - f_{uv}$

One interesting fact to note here is that the number of edges can infact increase on doing the above step. This can happen when we have a capacity of a edge = 0 and corresponding flow $(f_{uv}) < 0$. However there can only be a maximum of 2m edges. (where m is number of edges in input graph)

So given new capacities, we find any path from s to t. On the path, find the minimum capacity and keep ϵ = minimum capacity on that path. This will give us a new equality of the form $f_{uv} = C_{uv}$. Hence we have a required equality, and we move to the next recursion.

1.3 Order of Algorithm

If in Figure 2 we choose the path containing edge with weight 1 in each iteration it will take O(W) time to complete the execution. However the input size is O(ln W). (No of bytes required to represent total weight). So clearly the runtime is exponential in size of input. Hence choosing the appropriate path is critical. However LP cannot help us with this and we must look for a combinatorial algorithm. One such algorithm has been discussed in the next section.

1.4 Optimality

Another step of algorithm is to check if we have reached the optimal solution after each recursion. For this purpose, we look at all vertices accessible from s. We then seperate the graph into 2 parts, s on on side and t on the other. This is also know as the *cut*. For optimality we cannot have any flow through any cut. (The dual infact asks for the cut of minimum capacity.)

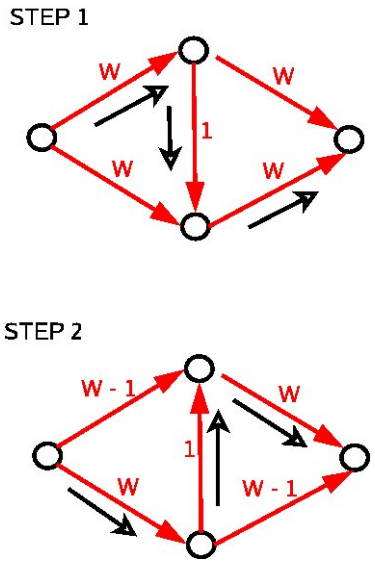


Figure 2: Example Graph

2 Improved Algorithm

2.1 Algorithm to find optimal path

The optimal path can be thought of as the path connecting s to t such that its minimum capacity edge is maximum among all paths from s to t. This can done with a simple modification to the Dijkstra's Algorithm (Tutorial 3 - Question 2).

2.2 Order of Algorithm

For calculating the order, we first make two important observations.

Observation 1

Let f be the current flow
and f^* be a flow of maximum value.

Claim : $f^* - f$ is a flow in the graph
Proof : Both f and f^* are flows in the original graph and hence must satisfy equations (1), (2) and (3). Therefore we have,

$$\forall u, v \; f^*_{uv} - f_{uv} \leq C_{uv} - f_{uv} \tag{8}$$
$$\forall u, v \; (f^*_{uv} - f_{uv}) + (f^*_{vu} - f_{vu}) = \; 0 \tag{9}$$
$$\forall u \neq s, t \; \sum_v f^*_{uv} - f_{uv} = 0 \tag{10}$$

which satisfies all conditions ((4),(5) and (6)) required by the flow in new graph.

Observation 2

Claim : The above flow $(f^* - f)$ can be written as a sum of *atmost* 2m flows, where each of them is a path from s to t.

Proof : This is simply because there can be a maximum of 2m different edges, and even if each represents a different flow we can only have 2m different flows from s to t.

Using the second observation I can say that, in every iteration I increase my flow by atleast $(|f^*|-|f|)/2m$. Hence

$$|f^{new}| \geq |f^{old}| + (|f^*| - |f|)/2m$$
$$|f^*| - |f^{new}| \leq (|f^*| - |f^{old}|)(1 - 1/2m)$$

So, in t iterations

$$|f^*|(1 - 1/2m)^t < 1$$
$$\text{as we began with } f^{old} = 0$$

This gives us,

$$t = 2m\ ln|f^*|$$

which is polynomial in size of input