MEKDELA AMBA UNIVERSITY

COLLAGE OF NATURAL AND COMPUTATIONAL SCIENCE

DEPARTMENT OF MATHEMATICS

NON-LINEAR OPTIMIZATION LECTURE NOTE

Prepared by: Adane A. ***(MSc. in optimization)***

March, 2020 G.C

TULUAWLIA, ETHIOPIA

Table of Contents

CHAPTER ONE

Basic Notation of Convex Analysis

Objectives of the chapter

- On completion of the chapter, successful students will be able to:
  - define convex sets and convex functions,
  - define concave function,
  - determine continuity and differentiability of convex functions,

Convexity is a branch of mathematical analysis dealing with convex sets and convex functions. It also represents a foundation for optimization.

1.1. Affine and Convex sets

A set $C \subseteq R^n$is called convex if $(1-\lambda)x + \lambda y \in C$ whenever $x, y \in C$ and $0 \leq \lambda \leq 1$. Geometrically, this means that C contains the line segment between each pair of points in C,

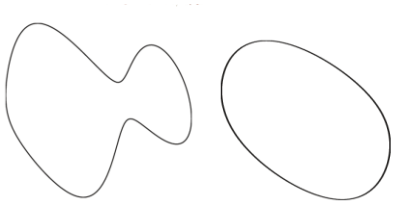


Figure 1 Example of a convex set (right) and a non-convex set (left).

Simple examples of convex sets are:

- ✓ The empty set $\emptyset$, the singleton set$\{x_0\}$, and the complete space R^n;
- ✓ Lines $\{a^T x = b\}$, line segments, hyper planes $\{A^T x = b\}$, and halfspaces $\{A^T x \leq b\}$;
- ✓ Euclidian balls:$B(a;\ \delta) = \{x \in R^n : ||x-a|| \leq \delta\}$.

Proof: Let us show Euclidian balls is convex set. Recall the triangle inequality which says that $||u+v|| \leq ||u|| + ||v|| \ whenever\ u, v \in R^n.\ Let\ x, y \in B(a;\ \delta)\ and\ \lambda \in [0,1]$.

Then $||\big((1-\lambda)x + \lambda y\big) - a|| = ||(1-\lambda)(x-a) + \lambda(y-a)||$

$$\leq ||(1-\lambda)(x-a)|| + ||\lambda(y-a)||$$

$$= (1-\lambda)||x-a|| + \lambda||y-a||$$

$$\leq (1-\lambda)\delta + \lambda\delta = \delta.$$

Therefore B(a; δ) is convex.

Every linear subspace is also a convex set,

Definition (affine Hull): let $V = \{x_1, x_2, \dots, x_n\} \subseteq R^n$ the affine hull of V is the set of

$$aff(V) = \{\lambda_I x_i + \lambda_2 x_2 + \cdots + \lambda_k x_k | x_i \in C, \lambda_i \in R^n, i = 1,2,\dots,k,\ \textstyle\sum_{i=1}^{k} \lambda_i = 1\}|$$

The point $x = \lambda_I x_i + \lambda_2 x_2 + \cdots + \lambda_k x_k | x_i \in V, \lambda_i \in R^n, i = 1,2, \dots, k, \sum_{i=1}^{k} \lambda_i = 1|$is called **affine combination** of the points $x_1, x_2, \dots, x_n$(the number of k points must be finite).

Definition (Convex Hull): let $V = \{x_1, x_2, \dots, x_n\} \subseteq R^n$ the convex hull of V is the set of

$conv(V) = \{\lambda_I x_i + \lambda_2 x_2 + \cdots + \lambda_k x_k | x_i \in C, \lambda_i \geq 0, i = 1,2, \dots, k, \sum_{i=1}^{k} \lambda_i = 1\}|$

The point $x = \lambda_I x_1 + \lambda_2 x_2 + \cdots + \lambda_k x_k | x_i \in V, \lambda_i \geq 0, i = 1,2, \dots, k, \sum_{i=1}^{k} \lambda_i = 1|$is called **convex combination** of the points $x_1, x_2, \dots, x_n$(the number of k points must be finite).

The convex hull of a set V is the smallest convex sets which include V:

- $Conv(V)$ is convex
- $V \subseteq Conv(V)$
- $\forall V', V'$ Convex,$V \subseteq V' \Rightarrow Conv(V) \subseteq V'$

1.2. Convex and concave functions

Convex and concave functions play an extremely important role in the study of nonlinear programming problems. Let $f(x_1, x_2 \dots x_n)$ be a function that is defined for all points (x_1, x_2, . . . , x_n) in a convex set *S*.

The notion of a convex function also makes sense for real-valued functions of several variables. Consider a real-valued function $f : C \rightarrow R$ where $C \subseteq R^n$ is a convex set. We say that f is convex provided that

$f((1 - \lambda)x + \lambda y) \leq (1 - \lambda)f(x) + \lambda f(y)\ (x, y \in R^n, 0 \leq \lambda \leq 1)$

It is called *strictly convex* if, for every two distinct points *x* and *y* and every $0 < \lambda < 1$,

$f\big((1 - \lambda)x + \lambda y\big) < (1 - \lambda)f(x) + \lambda f(y)\ (x, y \in R^n, 0 < \lambda < 1)$

(This inequality holds for all x, y and λ as specified). Due to the convexity of C, the point $(1 - \lambda)x + \lambda y$ lies in C, so the inequality is well-defined. The geometrical interpretation in one dimension is that whenever you take two points on the graph of f, say $(x, f(x))\ and\ (y, f(y))$, the graph of f restricted to the line segment [x, y] lies below the line segment in R^{n+1} between the two chosen points. A function g is called concave if $-g$ is convex. Every linear function is convex.

Some other examples of convex functions in n variables are

- $f(x) = L(x) + \alpha$ where L is a linear function from R^n into R (a linear functional) and α is a real number. Such a function is called an affine function and it may be written $f(x) = c^T x + \alpha$ for a suitable vector c.

- $f(x) = ||x||$ (Euclidean norm). That this is convex can be proved by writing $||(1-\lambda)x + \lambda y|| \leq ||(1-\lambda)x|| + ||\lambda y|| = (1-\lambda)||x|| + \lambda||y||$. In fact, the same argument can be used to show that every norm defines a convex function. Such an example is the l1-norm, also called the sum norm, defined by $||x||_1 = \sum_{j=1}^{n} |x_j|$.
- $f(x) = e^{\sum_{j=1}^{n} x_j}$ (see Exercise 7).
- $f(x) = e^{h(x)}$where $h : R^n \to R$ is a convex function.
- $f(x) = max_i\, g_i(x)$ where $g_i : R^n \to R$ is an affine function $(i \leq m)$. this means that the point wise maximum of affine functions is a convex function.

Concave functions are simply the negative of convex functions. In this case, linear interpolation underestimates the function. The definition above is altered by reversing the direction of the inequality. Strict concavity is defined analogously. Formally,

Definition. A function $f(x)$ is called *concave* if, for every x and y and every $0 \leq \lambda \leq 1$

$f((1-\lambda)x + \lambda y) \geq (1-\lambda)f(x) + \lambda f(y)\ (x, y \in R^n, 0 \leq \lambda \leq 1)$

It is called *strictly concave* if, for every two distinct points x and y and every $0 < \lambda < 1$,

$f\big((1-\lambda)x + \lambda y\big) > (1-\lambda)f(x) + \lambda f(y)\ (x, y \in R^n, 0 < \lambda < 1)$

Proposition : Let $C \subseteq R^n$ be a convex set and consider a convex function $f : C \to R$. Let $\alpha \in R$. Then the "sublevel" set $\{x \in C : f(x) \leq \alpha\}$ is a convex set.

Affine functions as an example: It can be shown that f is affine on R^n, as already defined, if and only if f is simultaneously convex and concave.

Note: The sum of two convex functions is convex and the sum of two concave functions is concave.

Let $\boldsymbol{S}$ be a nonempty convex set in R^n. The function $f: \boldsymbol{S} \to R$ is said to be ***quasi convex*** on S if for each $x_1, x_2 \in S$**,** the following inequality holds true:

$f[\lambda x_1 + (1-\lambda)x_2] \leq max(f(x_1), f(x_2)\}$ for each $\lambda \in (0, 1)$.

The function f is said to be **strictly quasi convex** on **S** if the above inequality holds as a strict inequality, provided that $f(x_1) \neq f(x_2)$ **.** The function f is said to be **strongly quasi convex** on **S** if the above inequality holds as a strict inequality for $x_1 \neq x_2$ **.**

The function f is called strictly quasi concave if $-f$ is strictly quasi convex. Strictly quasi convex functions are also sometimes referred to as semi-strictly quasi convex, functionally convex, or explicitly quasi convex.

1.3. Continuity and differentiable property of convex function

A convex function may not be differentiable in every point. However, one can show that a convex function always has one-sided directional derivatives at any point. But what about continuity?

Theorem: Let $f : C \to R$ be a convex function defined on an open convex set $C \subseteq R^n$. Then f is continuous on C.

However, a convex function may be discontinuous in points on the boundary of its domain. For instance, the function $f : [0,1] \to R$ given by $f(0) = 1 \; and \; f(x) = 0 \; for \; x \in (0,1]$ is convex, but discontinuous at x = 0. Next we give a useful technique for checking that a function is convex.

Theorem: Let $f : C \to R$ be a differentiable function defined on an open convex set $C \subseteq R^n$. Then the following conditions are equivalent:

i. f is convex.

ii. $f(x) \geq f(x_0) + \nabla f(x_0)^T (x - x_0) for \; all \; x, x_0 \in C.$

iii. $\left(\nabla f(x) - \nabla f(x_0)\right)^T (x - x_0) \geq 0 \; for \; all \; x, x_0 \in C.$

Signs of second derivatives in one dimension: For f twice differentiable on IR,

- f is convex $\Leftrightarrow f''(x) \geq 0$ for all x,
- f is strictly convex $\Leftarrow f''(x) > 0$ for all x.

Notice that the final condition is not equivalence but only an implication in one direction!

An example is $f(x) = x^4$, with $f''(x) = 12x^2$. This function is strictly convex on IR because $f'(x) = 4x^3$ is an increasing function. But $f''(x)$ fails to be positive everywhere: $f''(0) = 0$.

Tests for convexity using derivatives: The following facts help in identifying examples of convex functions that are differentiable. Later there will be other tactics available, which can be used to ascertain the convexity of functions that have been put together in certain ways from basic functions whose convexity is already known.

Monotonicity of first derivatives in one dimension: For f differentiable on IR,

- ✓ f is convex $\Leftrightarrow f'$ is non-decreasing,
- ✓ f is strictly convex $\Leftrightarrow f'$ is increasing,
- ✓ f is concave $\Leftrightarrow f'$ is non-increasing,
- ✓ f is strictly concave $\Leftrightarrow f'$ is decreasing,
- ✓ f is affine $\Leftrightarrow f'$ is constant.

For multi-dimension problems, the direction of steepest ascent is given by the *gradient vector* of the function $f(x) = f(x_1, x_2, \ldots, x_n)$, which is the multi-dimension analog of the single-dimension derivative. The gradient vector is defined as follows:

$$\nabla f(x) = [\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n}]$$

i.e. the vector of first partial derivatives at the point ***x*** where there are *n* variables. The length of the gradient vector is related to the rate of increase of the function in the gradient direction.

How can we determine whether a function $f(x_1, x_2, \ldots, x_n)$ of *n* variables is convex or concave on a set $S \subset R^n$? We assume that $f(x_1, x_2, \ldots, x_n)$ has continous second-order partial derivatives. Before stating the criterion used to determine whether $f(x_1, x_2, \ldots, x_n)$ is convex or concave, we require three definitions.

Definition: The **Hessian** of $f(x_1, x_2, \ldots, x_n)$ is the $n \times n$ matrix whose ij^{th} entry is

$$\frac{\partial^2 f}{\partial x_i \partial x_j}$$

Definition: An ***i*th principal minor** of an $n \times n$ matrix is the determinant of any $i \times i$ matrix obtained by deleting $n - i$ rows and the corresponding $n - i$ columns of the matrix.

Definition: The ***k*th leading principal minor** of an $n \times n$ matrix is the determinant of the $k \times k$ matrix obtained by deleting the last $n - k$ rows and columns of the matrix.

Theorem: Suppose $f(x_1, x_2, \ldots, x_n)$ has continuous second-order partial derivatives for each point $= (x_1, x_2, \ldots, x_n) \in S$. Then $f(x_1, x_2, \ldots, x_n)$. is a convex function on *S* if and only if for each $x \in S$, all principal minors of *H* are nonnegative.

Example : Show that $f(x_1, x_2) = {x_1}^2 + 2x_1x_2 + {x_2}^2$ is a convex function on $S = R^2$

Solution: we find that Hessian matrix.

$$H(x_1, x_2) = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

The first principal minors of the Hessian are the diagonal entries (both equal 2 ≥0). The second principal minor is $2(2) - 2(2) = 0 \geq 0$. For any point, all principal minors of *H* are nonnegative, so that $f(x_1, x_2)$ is a convex function on R^2.

Theorem: Suppose $f(x_1, x_2, \ldots, x_n)$ has continuous second-order partial derivatives for each point $= (x_1, x_2, \ldots, x_n) \in S$. Then $f(x_1, x_2, \ldots, x_n)$. is a concave function on *S* if and only if for each $x \in S$ and k= 1,2,…,n, all principal minors have the same sing $(-1)^k$.

Example: Show that $f(x_1, x_2) = -{x_1}^2 - x_1x_2 - 2{x_2}^2$ is a concave function on $S = R^2$

Solution: We find that Hessian matrix.

$$H(x_1, x_2) = \begin{bmatrix} -2 & -1 \\ -1 & -4 \end{bmatrix}$$

The first principal minors of the Hessian are the diagonal entries $(-2\ and\ -4)$. These are both nonpositive. The second principal minor is the determinant of $H(x_1, x_2)$ and equals $-2(-4) - (-1)(-1) = 7 > 0$. Thus $f(x_1, x_2)$ is a concave function on R^2.

Exercise: Determine whether the following function is convex, concave or neither.

1. $f(x_1, x_2) = {x_1}^2 - 3x_1x_2 + 2{x_2}^2$
2. $f(x) = x^{-1/2}\ for\ x \geq 0$
3. $f(x_1, x_2) = 2{x_1}^2 + {x_2}^2 - 2x_1x_2$
4. $f(x_1, x_2) = -{x_1}^2 - 3{x_2}^2 + x_1x_2 + 10x_1 - 10x_2$
5. $f(x_1, x_2, x_3) = {x_1}^4 + 2{x_2}^2 + 3{x_3}^2 - 4x_1 - 4x_2x_3$
6. $f(x_1, x_2) = -2{x_1}^2 - 3{x_2}^2 - 2{x_3}^2 + 8x_1x_2 + 3x_1x_3 + 4x_2x_3$

CHAPTER TWO

Nonlinear optimization problem

Objectives of the chapter

- On completion of the chapter, successful students will be able to:
  - understand the fundamental principles of nonlinear programming,
  - test necessary and sufficient optimality conditions,
  - understand Kuhn-Tucker conditions,
  - solve nonlinear programming problems,
    - ✓ apply penalty method,
    - ✓ apply Lagrange method,
  - understand separable programming problem
  - apply iterative methods for solving convex optimization problems,

2.1. Introduction

Non-Linear Programming is a mathematical technique for determining the optimal solution to many business problems. Knowledge of differential calculus is essential to do computational work in solving the problems. In linear programming problems, we use to deal with linear objective functions and constraints to find the optimal solution. The constraints we have used in linear programming technique are of $\leq \ or \ \geq$ type or a combination of these two. It is also assumed in linear programming that the cost of production or unit profit contribution or problem constraints do not vary for the planning period and also at different levels of production. But it is only an assumption to simplify the matter. But in real world problem the profit, requirement of resources by competing candidate all will vary at different levels of production. Also due to many economic behaviours of demand, cost etc. the objective function tends to be non-linear many a time.

A general **nonlinear programming problem** (NLP) can be expressed as follows:

Find the values of decision variables $x_1, x_2, \dots, x_n$ that

$$\begin{aligned} &max \ (or \ min) z = \ f(x_1, x_2, \dots, x_n) \\ &s.t. \ g_1(x_1, x_2, \dots, x_n) \ (\leq, =, or \ \geq) \ b_1 \\ &g_2(x_1, x_2, \dots, x_n) (\leq, =, or \ \geq) b_2 \\ &\vdots \\ &g_m(x_1, x_2, \dots, x_n) \ (\leq, =, or \ \geq) \ b_m \end{aligned} \qquad (2.1)$$

As in linear programming, $f(x_1, x_2, \dots, x_n)$ is **objective function,** of the problem and

$g_1(x_1, x_2, \ldots, x_n)(\leq, =, or\ \geq)b_1, \ldots, g_m(x_1, x_2, \ldots, x_n)\ (\leq, =, or\ \geq)\ b_m$ where some of the constraints and/ or the objective functions are nonlinear is known as nonlinear Programming (NLP's)**.** An NLP with no constraints is an **unconstrained NLP.** The set of all points $(x_1, x_2, \ldots, x_n)$ such that x_i is a real number is R^n.

General Non-Linear Programming Problem can be solved by a method very similar to Simplex algorithm. Also there are many methods have been developed to get the solution since the appearance of the fundamental theoretical paper by **Kuhn** and **Tucker** (1915). In the coming discussion some methods of solution to general non-linear programming problem are discussed.

In Linear Programming, the objective function and also the constraints were linear in decision variable. Though this linearity is justified in many real life situations, there do arise such problems in business and industry that the relationship between the decision variables and the objective function it may contain non-linear expression of the decision variables. One way seems to be to approximate the non-linear relationship is by replacing approximated linear relationships and view the given problem as a perturbed version of the ideal problem. But the conclusion in such a situation may not be valid for the given problem or present solutions which may not give the required optimality. Unfortunately, there is no known algorithm to effectively and efficiently solve a given general non-linear programming problem. A method that is found to be useful in one problem may not be useful in another. This is one of the reasons why all the non-linear programming problems cannot be grouped under the same title.

To approximate the difficulty in the approach let us distinguish between the factors that make Linear Programming Problem (LPP) more attractive and a Non-Linear Programming Problem (NLPP) as more complex.

- The algorithm for solving LPP is based on the property that optimal solutions are to be found at the extreme points of the convex polyhedron. This implied that we limit our search to corner points and this could be completed in a finite number of iterations. But in NLPP the optimal solution can be anywhere along the boundaries of the feasible region or anywhere within the feasible region.
- Linear relationship between the decision variables is very easily amendable to linear algebraic transformation but non-linear relationship should be dealt with extreme care resulting in complex situations.
- The non-linear nature of relationship results in distinction between local solutions and global solutions. This means that any solution that is locally optimal has to be tested

for its optimality over the entire feasible region and not only at the extreme points as has been possible in an LPP. This also means that Simplex type algorithms do not suffice to solve NLPPs.

The **feasible region** for NLP (2.1) is the set of points$(x_1, x_2, \ldots, x_n)$ that satisfy the *m* constraints in (2.1). A point in the feasible region is a *feasible point*, and a point that is not in the feasible region is an *infeasible point.*

Any point $\tilde{x}$ in the feasible region for which $f(\tilde{x}) \geq f(x)$ holds for all points *x* in the feasible region is an **optimal solution** to the NLP. [For a minimization problem, $\tilde{x}$ is the optimal solution if $f(\tilde{x}) \leq f(x)$ for all feasible *x*].

Of course, if $f, g_1, g_2, \ldots, g_m$ are all linear functions, then (2.1) is a linear programming problem and may be solved by the simplex algorithm or other method.

2.2. Convex optimization

A convex optimization problem is to minimize a convex function f over a convex set C in R^n. These problems are especially attractive, both from a theoretic and algorithmic perspective.

Consider the standard form of convex optimization problem

$$\begin{aligned} \text{(CP):} \quad & \text{Minimize} f(x) \\ & \text{Subject to} \quad g_i(x) \leq 0 \quad \text{for } i = 1,2,\ldots,m \\ & \qquad\qquad h_k(x) = 0 \quad \text{for } k = 1,2,\ldots,p \end{aligned} \tag{2.2}$$

Where f and g_i are convex continuous differentiable functions and h_k`s are affine

i.e. $h_k = a^T x + b$

With the assumption of differentiability, the KKT necessary conditions for optimality are also sufficient when applied to a convex program.

An important concept in convex optimization is duality. To briefly explain this introduce again the Lagrangian function $L: R^n \times R^m \times R^r{}_+ \to R$ R given by

$$L(x, \lambda, v) = f(x) + \sum_{i=1}^{m} u_i \nabla g_i(x^*) - \sum_{k=1}^{p} v_k \nabla h_k(x^*) \ (x \in R^n, u \in R^m, v \in R^r{}_+)$$

Theorem4.3: Consider the CP. Suppose that f,g_i and $h_k$are differentiable. For a feasible point,x^* if there exist u_i, v_k satisfying

$\nabla f(x^*) + \sum_{i=1}^{m} u_i \nabla g_i(x^*) - \sum_{k=1}^{p} v_k \nabla h_k(x^*) = 0$ (2.2a)

$u_i g_i(x^*) = 0$, for $i = 1,2,\ldots,m$

$u_i \geq 0$, for$i = 1,2, \dots, m$. Then x^* is a global optimum for CP.

Proof: Let x be any feasible point .we want to show that $f(x) \geq f(x^*)$

As f is convex,$f(x) \geq f(x^*) + \nabla f(x^*)(x - x^*)$ (2.2b)

By substituting (2.2a) in (2.2b) for $\nabla f(x^*)$,

$f(x) \geq f(x^*) + \left(\sum_{k=1}^{p} v_k \nabla h_k(x^*) - \sum_{i=1}^{m} u_i \nabla g_i(x^*)\right)(x - x^*)$

$$\geq f(x^*) + \sum_{k=1}^{p} v_k \nabla h_k(x^*)(x - x^*) - \sum_{i=1}^{m} u_i \nabla g_i(x^*)\,(x - x^*) \qquad (2.2c)$$

We have $\nabla h_k(x^*)(x - x^*) = 0$ (2.2d)

To show (2.2d)

x and x^* are feasible$\Rightarrow h_k(x) = h_k(x^*) = 0$

$$\Rightarrow a^T x + b = a^T x^* + b$$

$$\Rightarrow a^T(x - x^*) = 0$$

$$\Rightarrow \nabla h_k(x^*)(x - x^*) = 0$$

We also have $u_i \nabla g_i(x^*)(x - x^*) \leq 0$ (2.2e)

To show (2.2e),

As g_i is convex, $g_i(x) \geq g_i(x^*) + \nabla g_i(x^*)(x - x^*)$

$$\Rightarrow 0 \geq u_i \nabla g_i(x) \geq u_i \nabla g_i(x^*) + u_i \nabla g_i(x^*)(x - x^*)$$

$$\Rightarrow 0 \geq u_i \nabla g_i(x^*)(x - x^*)$$

Using (2.2d) and(2.2e),equation (2.2c) gives $f(x) \geq f(x^*)$

2.3. Necessary and sufficient optimality conditions

Theorem (First order necessary condition):

Let $f : \Re^n \to \Re$ be differentiable at x^* .If x^* is a local minimum to the problem (P), then

$$\nabla f\left(x^*\right) = 0$$

Proof Let $x = x^* - t\nabla f\left(x^*\right)$ for sufficiently small $t > 0$

Then x is a neighborhood of x^* .Therefore as x^* is a local minimum, $f(x) - f\left(x^*\right) \geq 0$ (2.3)

By first order Taylor expansion, for a neighbor $x^* + \delta h$,

$f(x^* + \delta h) = f(x^*) + \delta h \nabla f(x^*)^T + o(\delta)$, where $\frac{o(\delta)}{\delta} \to 0$ as $\delta \to 0$

Then $f(x^* - t\nabla f(x^*) = f(x^*) + (-t\nabla f(x^*)\nabla f(x^*)^T + o(t)$

By (2.3), $f(x^* - t\nabla f(x^*)) - f(x^*) = (-t\nabla f(x^*)\nabla f(x^*)^T + o(t) \geq 0$

As, $t > 0$ dividing t by gives $-\|\nabla f(x^*)\|^2 + \frac{o(t)}{t} \geq 0$

By taking a limit as $t \to 0$, we obtain $\|\nabla f(x^*)\|^2 \leq 0$

Therefore $\|\nabla f(x^*)\| = 0$ Hence $\nabla f(x^*) = 0$

x^* is a **stationary point** if $\nabla f(x^*) = 0$

<u>Theorem</u> (Second order necessary conditions):

Let $f : \Re^n \to \Re$ be twice differentiable at x^*

i) If x^* is a local minimum point, then x^* is a stationary point (i.e. $\nabla f(x^*) = 0$) and $H(x^*)$ is positive semi definite.

ii) If x^* is a local *maximum* point, then x^* is a stationary point (i.e. $\nabla f(x^*) = 0$) and $H(x^*)$ is negative semi definite.

<u>Theorem</u> (sufficient conditions):

Let $f : \Re^n \to \Re$ be twice differentiable at x^*

i) If x^* is a stationary point and $H(x^*)$ is positive definite, then x^* is a local minimum point.

ii) If x^* is a stationary point and $H(x^*)$ is *negative* definite, then x^* is a local *maximum* point.

- a saddle point if $H(x^*)$ is indefinite

Note: For a function of one variable, $\nabla f(x^*) = 0$ means $f'(x^*) = 0$ and $H(x^*)$ is positive definite means $f''(x^*) > 0$

<u>Exercise</u>. 1) $f(x_1, x_2, x_3) = x_1 + 2x_3 + x_2x_3 - x_1^2 - x_2^2 - x_3^2$

2) $f(x) = x^3 + 1$

2.4. Penalty Methods

Penalty methods are a certain class of algorithms for solving constrained optimization problems. It replaces constrained optimization problems by a series of unconstrained optimization problems whose solutions ideally converge to the solution of the original constrained optimization problem. The unconstrained problems are formed by adding a term called a penalty function, to the objective function that consists of a penalty parameter multiplied by a measure of violation of the constraints. The measure of violation is non zero when the constraints are violated and is zero in the region where constraints are not violated.

It generate a sequence of infeasible points whose limit is an optimal solution to the original problem. The original constrained problem is transformed into an unconstrained one (or a series of unconstrained problems). The constraints are incorporated into the objective by means of a penalty parameter which penalizes any constraint violation.

Consider the linearly-constrained optimization problem P:

$$\text{(P):} \qquad minimize\ f(x) \qquad (2.4)$$
$$subject\ to \qquad g_i(x) \leq 0 \ \ i = 1,2,\dots,m$$
$$\boldsymbol{x \geq 0}$$

Then the mathematical Procedures of solving linear optimization problems by penalty methods are:-

Step 1: Identify the objective function, constraints of the problems and rewrite in standard form

Step 2: Choose a penalty function P as follows

$$P(x) = \sum_{i=1}^{m}[max(0, g_i(x))]^2 \qquad (2.4a)$$

From the above equation (2.4), it is easy to see that

- $if\ g_i(x) \leq 0, then\ \ [max(0, g_i(x))] = 0$
- $if\ g_i(x) > 0, then\ \ [max(0, g_i(x))] = g_i(x)$

Step 3: We can obtain the unconstrained optimization problem Q by adding a term called a penalty function, to the objective function that consists of a penalty parameter c multiplied by a measure of violation of the constraints,

i.e $Q(x, c) = f(x) + \lambda p(x)$

$$= f(x) + \lambda \sum_{i=1}^{m}[max(0, g_i(x))]^2 \qquad (2.4b)$$

Hence the constrained optimization problem can be changed into standard form of unconstrained optimization problem, that is

$$minimize \quad Q(x,c) = f(x) + \lambda p(x)$$
$$s.t\ x \in R \qquad (2.4c)$$

Step 4: Solve the unconstrained optimization problem by applying the first- order necessary condition to obtain the individual values of unknowns (or optimal solutions of the problem),

$i.e\ \nabla Q(x,c) = 0$ this means that

$$= \nabla f(x) + 2\lambda[\sum_{i=1}^{m}[max(0, g_i(x))]\nabla(g_i(x))] = 0 \qquad (2.4d)$$

Example 3.1. Consider the problem

$$minf(x) = x$$
$$s.t\ g(x) = x - 2 \leq 0$$
$$x \geq 0$$

Let us choose the penalty function $P(x) = \begin{cases} [g(x)]^2 = (x-2)^2, if\ \ x \geq 0 \\ 0\ if\ \ x < 0 \end{cases}$

The unconstrained problem we obtained $Q(x,c) = f(x) + \lambda P(x) = x + \lambda(x-2)^2$

Then the constrained optimization problem changed into the standard form of unconstrained optimization problem

$$minimize \quad Q(x,c) = f(x) + \lambda p(x)$$
$$s.t\ x \in R$$

Since $Q(x,c) = f(x) + \lambda P(x) = x + \lambda(x-2)^2$

Finally we can solve the unconstrained problem by applying the first order necessary condition to find the optimal value of the objective function

$Q(x,c)_x = f_x(x) + \lambda P_x(x) = 1 + 2\lambda(x-2) = 0$. When we expressed in terms of parameter λ is

$x = 2 - \frac{1}{2\lambda}$ $\quad Q(x,c)$ has optimal solution at $\tilde{x} = 2 - \frac{1}{2\lambda}$

As $\lambda \to \infty$ $\tilde{x} = \lim_{\lambda\to\infty}\ 2 - \frac{1}{2\lambda} = 2$

Therefore $\tilde{x} = 2$

Hence the minimum value of the objective function is 2.

Exercise: solve the following problem using penalty method

a. Min $x_1 + x_2$

Subject to ${x_1}^2 + {x_2}^2 = 2$

2.5. Lagrange Multipliers and Karush-Kuhn-Tuker Conditions (KKT)

Lagrange multipliers can be used to solve NLPs in which all the constraints are equality constraints. We consider NLPs of the following type:

$$\begin{aligned} \text{(P):} \quad & \text{Minimize} f(x), \quad x \epsilon \Re^n \\ & \text{Subject to } g_j(x) \geq 0, j = 1, \dots J \\ & h_k(x) = 0, k = 1, \dots K \end{aligned} \tag{2.5}$$

To solve (2.5), we associate a **multiplier** $\boldsymbol{\mu_j}$ *and* $\boldsymbol{\lambda_k}$ with the j^{th} and k^{th} constraint respectively in (2.5) and form the **Lagrangian function is defined as:**

$$\mathcal{L}(x, \mu, \lambda) = f(x) - \sum_{j=1}^{J} \mu_j g_j(x) + \sum_{k=1}^{K} \lambda_k h_k(x) \tag{2.5a}$$

Where μ and λ are Lagrange multipliers. Then we attempt to find a point $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n, \tilde{\lambda}_1, \tilde{\lambda}_2, \dots, \tilde{\lambda}_m, \tilde{\mu}_1, \tilde{\mu}_2, \dots, \tilde{\mu}_m)$ that maximizes (or minimizes) $L(x_1, x_2, \dots, x_n, \lambda_1, \lambda_2, \dots, \lambda_m, \mu_1, \mu_2, \dots, \mu_m)$. In many situations, $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n)$ will solve (2.5). Suppose that (2.5) is a maximization problem.

If $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n, \tilde{\lambda}_1, \tilde{\lambda}_2, \dots, \tilde{\lambda}_m, \tilde{\mu}_1, \tilde{\mu}_2, \dots, \tilde{\mu}_m)$ maximizes *L*, then at

$(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n, \tilde{\lambda}_1, \tilde{\lambda}_2, \dots, \tilde{\lambda}_m, \tilde{\mu}_1, \tilde{\mu}_2, \dots, \tilde{\mu}_m)$.

The Karush-Kuhn-Tucker (KKT) optimality conditions are fundamental for many algorithms for constrained optimization problems. They can also be called first-order conditions because in their derivation properties of the gradients are used.

Definition: The inequality constraint $g_j(x) \geq 0$ is said to be active (or binding) constraint at the point x^* if $g_j(x^*) = 0$. It is said to be inactive (or nonbinding) if $g_j(x^*) > 0$.

Remark:

- If we can identify the inactive constraints at the optimum before solving the problem, we can delete those constraints from the model and reduce the problem size.
- Kuhn and Tucker developed the necessary and sufficient optimality conditions for problem (P).
- It is assumed that *f*, g_j, and h_k are differentiable

To solve (2.5), we use the Lagrange function (2.5a):

To find critical point of L,

$\frac{\partial \mathcal{L}}{\partial x_i} = 0$ for $i = 1, \dots, n$

The KKT conditions are stated as:

Find vectors x, μ and λ that satisfy

$$\nabla \mathcal{L}(x, \mu, \lambda) = 0$$
$$g_j(x) \geq 0, \forall j$$
$$h_k(x) = 0, \forall k$$
$$\mu_j g_j(x) = 0\ , \forall j$$
$$\mu_j \geq 0, \forall j$$

Observe that for the Maximum problem, all one needs to do is to change the *minus* sign in the Lagrangian to plus, because finding a maximum for f is the same as finding a minimum for $-f$.

Let $\bar{x}$ be a feasible solution to problem (P) and Let $I = \{j : g_j(\bar{x}) = 0\}$. Then $\bar{x}$ is said to satisfy **constraint qualification** if

$\nabla g_j(\bar{x})$ for $j \in I$ and $\nabla h_i(\bar{x})$ for $k = 1, \ldots, K$ are linearly independent

• The constraint qualification holds in the following cases:

1. When all constraints are linear.
2. When all the inequality constraints are concave functions and the equality constraint are linear and there exists at least one feasible *x* that is strictly inside the feasible region of the inequality constraints. In other words, there exists an *x* such that $g_j(x) > 0$ for $j = 1,\ldots, J$ and $h_k(x) = 0$ for $k = 1,\ldots,k$.

Example Consider the NLP problem

$$min f(x) = 1 - x^2$$
$$s.t\ -1 \leq x \leq 3$$

Check the optimality of $x = 2$.

Solution: First we have to write the problem in our standard form:

$$\min \quad f(x) = 1 - x^2$$
$$s.t \quad g_1(x) = x + 1 \geq 0$$
$$g_2(x) = 3 - x \geq 0$$

- ✓ Before we can apply the necessity theorem, we have to check that the constraint qualification holds at x = 2.
- ✓ Since $x = 2$ is feasible, g_1 and g_2 are linear functions, the constraint qualification holds. Therefore, we can apply the theorem.
- ✓ We need to check that there is *u* such that *x* and *u* are solutions to the KT problem.

The KKT conditions are stated as:

Find vectors $x, and\ u$ that satisfy

$$\mathcal{L}(x,u) = f(x) + u_1\, g_1(x) + u_2 g_2(x)$$
$$= 1 - x^2 + u_1(x+1) + u_2(3-x)$$
$$\nabla\mathcal{L}(x,u) = 0$$
$$g_j(x) \geq 0, \forall j$$
$$u_j g_j(x) = 0\ , \forall j$$
$$u_j \geq 0, \forall j$$

1. $-2x - u_1 + u_2 = 0$
2. $x + 1 \geq 0, \qquad 3 - x \geq 0$
3. $u_1(x+1) = 0,\ \ u_2(3-x) = 0$
4. $u_1 \geq 0, u_2 \geq 0$

For$x = 2$, condition (3) requires $u_1 = u_2 = 0$

Condition (1) is violated $(-4 \neq 0)$

There is no solution to the KKT condition for $x = 2$.

Hence $x = 2$ is not an optimal value of the given problem

Exercise: Solve the following optimization problem

b. $\max[-(x_1 - 4)^2 - (x_2 - 4)^2]$

Subject to $x_1 + x_2 \leq 4$

$3x_1 + x_2 \leq 9,$

$x_1, x_2 \geq 0$

c. $\min(x_1 - 1)^2 + (x_2 - 2)^2$

Subject to $x_1 - x_2 = 1$

$x_1 + x_2 \leq 2$

$x_1, x_2 \geq 0$

2.6. Quadratic Programming problem

Quadratic programming (QP) is a class of the nonlinear programming problems belonging to the class of constrained nonlinear optimization problems. The problem is described by the quadratic objective function subject to linear constraints in a form of equalities and inequalities.

A general quadratic programming problem (2.6), when the objective functional f is quadratic and the constraints are linear in $x \in R^n$ is defined as:

$$\min_{x\in R^n} f(x) = \frac{1}{2}x^T Bx + q^T x \qquad (2.6a)$$

Subject to $A_1 x = c$ (2.6b)

$A_2 x \leq d$ (2.6c)

Where $B \in R^{n\times n}$is symmetric,$A_1 \in R^{m\times n}$,$A_2 \in R^{p\times n}$ and $q \in R^n$, $c \in R^m$, d $\in R^p$.Depending on the definiteness of the matrix B, QP problems can be divided into two classes. The first class is convex QP (strictly convex QP) when B is positive semi definite (B is positive definite) and the second one is non-convex QP when B is not positive semi definite. The Lagrangian for the problem in (2.6) is defined as

$$\mathcal{L}(x, \mu, \lambda) = \frac{1}{2}x^T \mathcal{B}x + q^T x + \mu^T(A_1 x - c) + \lambda^T(A_2 x - c) \qquad (2.7)$$

Then necessary conditions for $x^*$to be solutions of (4.3) are that there exist vectors $\mu \epsilon \mathbb{R}^m$ and $\lambda \epsilon \mathbb{R}^p$ such that the KKT conditions are satisfied

Remark: The quadratic nature of the objective function means that it can include squared terms, or terms that are the product of two variables, or linear terms.

For example $f(x_1, x_2, x_3) = 12{x_1}^2 - 4x_1x_2 + 5x_3$

In this case we set $B = \begin{bmatrix} 24 & -4 & 0 \\ -4 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, constructed as follows

- For squared terms like $c{x_i}^2$,write $2c$ in the ii position of B. So that $12{x_1}^2$ term in our example objective function appears as a 24 in the 1-1 position of B.
- For quadratic term involving a pair of variables like cx_ix_j , write c in both the ij position and ji position of B. So the $-4x_1x_2$ term in our example objective function appears as two terms;-4 in the 1-2 position and -4 in 2-1 position of B.
- Purely linear terms(like the $5x_3$ in our example) do not appear in B at all, they appear in the standard linear part of the objective function $q^T x$

If the Hessian matrix B is positive semi definite, we say that (2.6) is a convex QP, and in this case the problem is often similar in difficulty to a linear program. Nonconvex QPPs, in which Q is an indefinite matrix, can be more challenging because they can have several stationary points and local minima.

Theorem 4.3: Let Ω be nonempty convex set in $\mathbb{R}^n$and let f: $\Omega \to \mathbb{R}$ be differentiable on Ω. Then f is convex if and only if for any $x, y \epsilon \Omega$,we have $f(y) - f(x) \geq \nabla f(x)(y - x)$

Proof:

($\Rightarrow$): suppose f is convex. Let x, y $\epsilon \Omega$ and put$z(\lambda) = (1 - \lambda)x + \lambda y$ for$\lambda \epsilon [0,1]$

Then$f\big(z(\lambda)\big) = f\big((1-\lambda)x + \lambda y\big) \leq (1-\lambda)f(x) + \lambda f(y)$

$$f\big(z(\lambda)\big) - f(x) \leq \lambda\big(f(y) - f(x)\big)$$

i.e., $\frac{f(z(\lambda))-f(x)}{\lambda} \leq f(y) - f(x)$ (2.8)

But $\nabla f(x) = \lim_{z\to x} \frac{f(z)-f(x)}{z-x} = \lim_{\lambda\to 0} \frac{f(z)-f(x)}{\lambda(y-x)} = \frac{1}{y-x} \lim_{\lambda\to 0} \frac{f(z)-f(x)}{\lambda}$

Thus as $\lambda \to 0^+$in equation (2.8), we obtain

$$(y-x)\nabla f(x) \leq f(y) - f(x)$$

(⇐)**: Suppose**$f(y) - f(x) \geq \nabla f(x)(y-x)\ \forall x, y \in \Omega$ and $0 \leq \lambda \leq 1$.

Let$z = \lambda x + (1-\lambda)y$. We note that z is also in Ω. For x and z we have

$f(x) - f(z) \geq \nabla f(z)(x-z)$ (2.9)

And for y and z we have

$f(y) - f(z) \geq \nabla f(z)(y-z)$ (2.10)

Multiplying(2.9) byλ and (2.10) by $(1-\lambda)$ and adding, we obtain

$$\lambda f(x) - \lambda f(z) + (1-\lambda)f(y) - (1-\lambda)f(z) \geq \lambda(x-z)\nabla f(z) + (1-\lambda)(y-z)\nabla f(z)$$

$$\Longrightarrow \lambda f(x) + (1-\lambda)f(y) \geq f(z) + [\lambda x + (1-\lambda)y]\nabla f(z) - z\nabla f(z) \quad (2.11)$$

Appling the definition of z, (2.8) becomes

$$\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$$

Which implies that f(x) is convex.

2.7. Separable Programming

Many NLPs are of the following form:

$$\text{Max (or min) } z = \sum_{j=1}^{n} f_j(x_j)$$

$$s.t \quad \sum_{j=1}^{n} g_{ij}(x_j) \leq b_i \ (i = 1,2,\dots,m)$$

Because the decision variables appear in separate terms of the objective function and the constraints, NLPs of this form are called **separable programming problems.** Separable programming problems are often solved by approximating each $f_j(x_j)$ and $g_{ij}(x_j)$ by a piecewise linear function. Before describing the separable programming technique, we give an example of a separable programming problem.

Example: Determine the following optimization problem are separable programming

a. Min $-{x_1}^2 + 2{x_2}^2 - x_1 - 6x_2$

Subject to $2x_1 + x_2 \leq 2$

$${x_1}^2 - x_2 \leq 3$$

b. Min ${x_1}^2 + {x_2}^2 - 6x_1 - 8x_2 - x_3$

Subject to $x_1 + x_2 + x_3 \leq 5$

$${x_1}^2 + x_2 \leq 8$$

c. Min $-{x_1}^2 + 2{x_2}^2 - x_1x_2 - 6x_2$

Subject to $2x_1 + x_2 \leq 0$

$${x_1}^2 - x_1x_2 \leq 1$$

Solution: a) we have that from the objective function $f(x_1, x_2) = -{x_1}^2 + 2{x_2}^2 - x_1 - 6x_2$

$f(x_1) = -{x_1}^2 - x_1$, $f(x_2) = +2{x_2}^2 - 6x_2$

Thus $f(x_1, x_2) = f(x_1) + f(x_2)$

Therefore $f(x_1, x_2)$ separable function

We have that from the first constraint $g_1(x_1, x_2) = 2x_1 + x_2$

$g_{11}(x_1) = 2x_1$, $g_{12}(x_2) = x_2$

Thus $g_1(x_1, x_2) = g_{11}(x_1) + g_{12}(x_2)$

Therefore $g_1(x_1, x_2)$ separable function

And also we have from the second constraint $g_2(x_1, x_2) = = {x_1}^2 - x_2$

$g_{21}(x_1) = {x_1}^2$, $g_{12}(x_2) = -x_2$

Thus $g_2(x_1, x_2) = g_{21}(x_1) + g_{22}(x_2)$

Therefore $g_2(x_1, x_2)$ separable function

Hence the problem is separable problem

Linear approximation a continuous function

Let $f: [x_0, x] \to R$ be a function and let $\{x_0, x_1, \dots, x_n\} \subseteq [x_0, x]$ such that

$$[x_1, x_2] \cup [x_2, x_3] \dots [x_{n-1}, x_n] = [x_0, x]$$

Actually, instead of solving the problem directly, we make an appropriate approximation so that linear programming can be utilized. In practice, two types of approximations, called the δ - *method* and the λ - *method*, are often used. Since we have introduced the δ - *method* when discussing integer programming, we consider the λ - *method* in this section.

Now define $f^a(x) = g_1(x) + g_2(x) + \cdots + g_n(x)$

Since $[x_0, x_1]$ is convex $\exists \lambda_i \in [0,1]$ such that for all $x \in [x_0, x_1]$, $x = \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_n x_n$

$\sum_{i=0}^{n} \lambda_i = 1, \quad \lambda_i \geq 0$ For at most $\lambda_i{}'s$

$f^a(x) = f^a(\sum_{i=0}^{n} \lambda_i x_i) = \sum_{i=0}^{n} \lambda_i f^a(x_i) = \sum_{i=0}^{n} \lambda_i f(x_i)$

For $|x_{i+1} - x_i| \to 0$, $f^a(x) \to f(x)$

$$f^a(x) = \sum_{i=0}^{n} \lambda_i f(x_i)$$

So if

$f(x) = \sum_{i=0}^{n} f_j(x_j)$, then f is approximated linearly by ${f_j}^a$

$$f^a(x) = \sum_{k=0}^{r} \lambda_{jk} f_j(x_{jk})$$

$$f(x) = \sum_{j=0}^{n} \sum_{k=0}^{r} \lambda_{jk} f_j(x_{jk})$$

The corresponding approximated optimization (P_A): is

$$(P_A): \min\{ \sum_{j=0}^{n} \sum_{k=0}^{r} \lambda_{jk} f_j(x_{jk}) \}$$

$$s.t \sum_{j=0}^{n} \sum_{k=0}^{r} \lambda_{jk} g_{jk}(x_{jk}) \leq b_i \; for \; 1 = 1,2, \ldots, m$$

With the conditions (i) $\sum_{k=0}^{r} \lambda_{jk} = 1$

(ii) $\lambda_{jk} > 0$ for at most two they must be consecutives

Thus general technique is motivated easily by solving a specific example.

Example: Approximate the following separable programming problem

$$Maximize \; f(x) = 2{x_1}^2 - 4x_1 + {x_2}^2 - 3x_3$$

$$Subject \; to: \; x_1 + x_2 + x_3 \leq 4$$

$${x_1}^2 - x_2 \leq 2$$

$$x_1, x_2, x_3 \geq 0.$$

Solution: The objective function is now written as $f(x) = f_1(x_1) + f_2(x_2) + f_3(x_3)$, where

$f_1(x_1) = 2{x_1}^2 - 4x_1$

$f_2(x_2) = {x_2}^2$

$f_3(x_3) = -3x_3$

Thus it is separable. Clearly, the linear constraints are also separable i.e. the constraint are separable.

To form the approximation problem, we approximate each nonlinear term by a piecewise-linear curve. We have used three segments to approximate the function f_1 and two segments to approximate the functions f_2 and f_3. Note that the constraints imply that

To find the grid points of x_1, x_2, x_3 from the constraints

For $x_1\ if\ x_2 = x_3 = 0$ and we get $0 \le x_1 \le 4$

For $x_2\ if\ x_1 = x_3 = 0$ and we get $0 \le x_2 \le 4$

But expression of x_3 is linear from objective and constraint function so no need to linearize function of x_3 put as it is.

i. Linearize the objective function using grid points $x_1 and\ x_2$ {0,2,4}

$x_1 = \{x_{10}, x_{11}, x_{12}\} = \{0,2,4\}$ Or we can use {0,1,2,3,4}

$x_2 = \{x_{20}, x_{21}, x_{22}\} = \{0,2,4\}$ Or we can use {0,1,2,3,4}

- The linear approximation of $f_1(x_1) = {f_1}^a(x_1)$

$$ {f_1}^a(x_1) = \lambda_{10} f_1(x_{10}) + \lambda_{11} f_1(x_{11}) + \lambda_{12} f_1(x_{12}) $$

Where $f_1(x_1) = 2{x_1}^2 - 4x_1$

$f_1(x_{10}) = 0, f_1(x_{11}) = 0\ and\ f_1(x_{12}) = 16$

$$ {f_1}^a(x_1) = 0\lambda_{10} + 0\lambda_{11} + 16\lambda_{12} \qquad (1) $$

- The linear approximation of $f_2(x_2) = {f_1}^a(x_2)$

$$ {f_2}^a(x_2) = \lambda_{20} f_2(x_{20}) + \lambda_{21} f_2(x_{21}) + \lambda_{22} f_2(x_{22}) $$

Where $f_2(x_2) = {x_2}^2$

$f_2(x_{20}) = 0, f_2(x_{21}) = 4\ and\ f_2(x_{22}) = 16$

$$ {f_2}^a(x_2) = 0\lambda_{20} + 4\lambda_{21} + 16\lambda_{22} \qquad (2) $$

But $f_3(x_3)$ does not need linearize because in objective function and constraint it is linear.

Therefore the linear approximation of $f(x) is\ \ f^a(x)$

$$ f^a(x) = {f_1}^a(x_1) + {f_2}^a(x_2) + f_3(x_3) $$

$$ = 0\lambda_{10} + 0\lambda_{11} + 16\lambda_{12} + 0\lambda_{20} + 4\lambda_{21} + 16\lambda_{22} - 3x_3 $$

$$ = 16\lambda_{12} + 4\lambda_{21} + 16\lambda_{22} - 3x_3 $$

ii. Linearize the constraints

- The linear approximation of $g_{11}(x_1) = {g_{11}}^a(x_1)$

$$ {g_{11}}^a(x_1) = \lambda_{10} g_{11}(x_{10}) + \lambda_{11} g_{11}(x_{11}) + \lambda_{12} g_{11}(x_{12}) $$

Where $g_{11}(x_1) = x_1$

$g_{11}(x_{10}) = 0, g_{11}(x_{11}) = 2\ and\ g_{11}(x_{12}) = 4$

$$g_{11}{}^{a}(x_1) = 0\lambda_{10} + 2\lambda_{11} + 2\lambda_{12} \quad (3)$$

- The linear approximation of $g_{12}(x_2) = g_{12}{}^{a}(x_2)$

$$g_{12}{}^{a}(x_2) = \lambda_{20}g_{12}(x_{20}) + \lambda_{21}g_{12}(x_{21}) + \lambda_{22}g_{12}(x_{22})$$

Where $g_{12}(x_1) = x_2$

$g_{12}(x_{20}) = 0, g_{12}(x_{21}) = 2 \; and \; g_{12}(x_{22}) = 4$

$$g_{12}{}^{a}(x_2) = 0\lambda_{20} + 2\lambda_{21} + 4\lambda_{22} \quad (4)$$

$$g_{13}(x_3) = x_3 \quad (5)$$

Then using eq(3)-eq(5) we get

$$g_1(x) = g_{11}{}^{a}(x_1) + g_{12}{}^{a}(x_2) + g_{13}(x_3) \leq 4$$

$$0\lambda_{10} + 2\lambda_{11} + 2\lambda_{12} + 0\lambda_{20} + 2\lambda_{21} + 4\lambda_{22} + x_3 \leq 4$$

$$2\lambda_{11} + 2\lambda_{12} + 2\lambda_{21} + 4\lambda_{22} + x_3 \leq 4 \quad (*)$$

- The linear approximation of $g_{21}(x_1) = g_{21}{}^{a}(x_1)$

$$g_{21}{}^{a}(x_1) = \lambda_{10}g_{21}(x_{10}) + \lambda_{11}g_{21}(x_{11}) + \lambda_{12}g_{21}(x_{12})$$

Where $g_{21}(x_1) = {x_1}^2$

$g_{21}(x_{10}) = 0, g_{21}(x_{11}) = 4 \; and \; g_{21}(x_{12}) = 16$

$$g_{21}{}^{a}(x_1) = 0\lambda_{10} + 4\lambda_{11} + 16\lambda_{12} \quad (6)$$

- The linear approximation of $g_{22}(x_2) = g_{22}{}^{a}(x_2)$

$$g_{22}{}^{a}(x_2) = \lambda_{20}g_{22}(x_{20}) + \lambda_{21}g_{22}(x_{21}) + \lambda_{22}g_{22}(x_{22})$$

Where $g_{12}(x_1) = -x_2$

$g_{22}(x_{20}) = 0, g_{22}(x_{21}) = -2 \; and \; g_{22}(x_{22}) = -4$

$$g_{22}{}^{a}(x_2) = 0\lambda_{20} - 2\lambda_{21} - 4\lambda_{22} \quad (7)$$

Then using eq(6) and eq(7) we get

$$g_2(x) = g_{21}{}^{a}(x_1) + g_{22}{}^{a}(x_2) \leq 2$$

$$0\lambda_{10} + 4\lambda_{11} + 16\lambda_{12} + 0\lambda_{20} - 2\lambda_{21} - 4\lambda_{22} \leq 2$$

$$4\lambda_{11} + 16\lambda_{12} - 2\lambda_{21} - 4\lambda_{22} \leq 2 \quad (**)$$

Thus linear approximation of the above problem (P) given by

$$(P_A): \min\{f_1{}^{a}(x_1) + f_2{}^{a}(x_2) + f_3(x_3)\}$$

$$s.t \; g_{11}{}^{a}(x_1) + g_{12}{}^{a}(x_2) + g_{13}(x_3) \leq 4$$

$$g_{21}{}^{a}(x_1) + g_{22}{}^{a}(x_2) \leq 2$$

$$x_1, x_2, x_3 \geq 0$$

Then $(P_A): \min \; 16\lambda_{12} + 4\lambda_{21} + 16\lambda_{22} - 3x_3$

$$s.t \; 2\lambda_{11} + 2\lambda_{12} + 2\lambda_{21} + 4\lambda_{22} + x_3 \leq 4$$

$$4\lambda_{11} + 16\lambda_{12} - 2\lambda_{21} - 4\lambda_{22} \leq 2$$

With conditions $\lambda_{10} + \lambda_{11} + \lambda_{12} = 1$

$\lambda_{20} + \lambda_{21} + \lambda_{22} = 1$

$\lambda_{10}, \lambda_{11}, \lambda_{12}, \lambda_{20}, \lambda_{21}, \lambda_{22} \geq 0$ at most two they must be consecutives i.e. $\begin{cases} \lambda_{10} \; and \; \lambda_{11} \; or \; \lambda_{11} and \; \lambda_{12} \\ \lambda_{20} \; and \; \lambda_{21} or \; \lambda_{21} and \; \lambda_{22} \end{cases}$

After this solve the problem (P_A) using simplex method or other, but in this material solve using simplex method.

Inducing Separability: If the problem is Non-separable problems frequently can be reduced to a separable form by a variety of formulation tricks. A number of such transformations are summarized in Table.

Term	*Substitution*	*Additional constraints*	*Restriction*
$x_1 x_2$	$x_1 x_2 = {y_1}^2 - {y_2}^2$	$y_1 = \frac{1}{2}(x_1 + x_2)$ $y_2 = \frac{1}{2}(x_1 - x_2)$	None
$x_1 x_2$	$y_1 = x_1 x_2$	$logy_1 = logx_1 + logx_2$	$x_1, x_2 \geq 0$
${x_1}^{x_2}$	$y_1 = {x_1}^{x_2}$	$y_1 = 10^{y_2 x_2}$ $x_1 = 10^{y_2}$	$x_1 \geq 0$
$2^{{x_1 + x_2}^2}$	$y_1 = 2^{{x_1 + x_2}^2}$	$logy_1 = \log_2(x_1 + {x_2}^2)$	None
$(x_1 + x_2)^2$	$x_3 = (x_1 + x_2)$	$x_1 + x_2 - x_3$	$x_3 \geq 0$

Exercise: Approximate the following non-separable programming problem

$$Maximize\ f\ (x) = 20x_1 + 16x_2 - {x_1}^2 - {x_2}^2 - (x_1 + x_2)^2,$$
$$Subject\ to:\ x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$

2.8. Iterative methods for solving convex optimization problems

In this section we use feasible direction methods solves nonlinear programming problem by moving from a feasible point an improved feasible point. Given a feasible point $x_k, a\ direction\ d_k$ is determined such that λ>0 and sufficiently small, the following two properties are true

i. $x_k + \lambda d_k$ is a feasible point
ii. The objective function value at $x_k + \lambda d_k$ is better than the objective function value at x_k.

Zoutendijk method

Definition: consider the problem f(x) subject to $x \in s$ where $f(x): R^n \to R$ and s is nonempty set i.e $(P): \min f(x), x \in s$

The nonzero vector d is called a feasible direction at $x \in s$ if there exist a $\delta > 0$ such that $x + \lambda d \in s \; \forall \lambda \in (0, \delta)$.

Furthermore d is called an improving feasible direction at $x \in s$ if there exist a $\delta > 0$ such that $f(x + \lambda d) < f(x) and \;\; x + \lambda d \in s \; \forall \lambda \in (0, \delta)$.

Consider the constraint is linear

Consider the case where the feasible region s is defined by a system of linear constraint, so that the problem under consideration is the form

$$(P): \min f(x)$$
$$s.t \; Ax \leq b$$
$$Qx = q$$

Where $A \; is \; m \times n, Q \;\; is \; an \; l \times n \; , bis \; an \; m \; vectors \; and \; q \; ia \; an \; l \; vectors$

Steps of Zoutendijk method

Initialization: Finding a starting feasible solution x_1 with $Ax_1 \leq b$ and $Qx = q$ let $k = 1$ and go to next step

Step 1: Given that x_k suppose that A^t and b^t are decomposed in to $({A_1}^t, {A_2}^t)$ and $({b_1}^t, {b_2}^t)$ so that $A_1 x_k \leq b_1$ and $A_2 x_k \leq b_2$.

Let d_k an optimal solution to the following

$$min \nabla f(x)^t d$$
$$s.t \; A_1 d \leq 0$$
$$Qd = 0$$
$$-1 \leq d_j \leq 1 \; fo \; j = 1,2,3 \ldots, n$$

If $\nabla f(x_k) d = 0$ stop, x_k is KKT point otherwise go to step 2

Step 2: let λ_k be an optimal solution to the line search problem

$$(P): min \; f(x_k + \lambda d_k)$$
$$s.t \qquad 0 \leq \lambda \leq \lambda_{max}$$

Where λ_{max} is determine using equation (*)

Let $x_{k+1} = x_k + \lambda_k d_k$, then identify the new set binding constraints at x_{k+1} and update $A_1 \; and \; A_2$ accordingly. Replace k by k+1 and go to step 1.

Exercise: Solve the following problem using **Zoutendijk method**

$$(P): min \;\; 2{x_1}^2 + 2\,{x_2}^2 - 2x_1x_2 - 4x_1 - 6x_2,$$

$$Subject\ to: \;\; x_1 + x_2 \leq 2$$

$$x_1 + 5x_2 \leq 5$$

$$-x_1 \leq 0$$

$$-x_2 \leq 0$$

Use ${x_1}^t = (0,0)$ as initial point

CHAPTER THREE

Discreet optimization

Objectives of the chapter

- On completion of the chapter, successful students will be able to:
  - solve discrete optimization problems using different methods,
    - ✓ The Knapsack problem
    - ✓ Branch-and-Bound method
  - understand the scope and limitation of modeling practical problems as nonlinear programs,

3.1. Dynamic programming

Definition:-optimization problem is maximizing or minimizing some function relative to some set often representing a range of choices available in a certain situation.

Definition:-Dynamic programming is a mathematical technique concerned with optimization of multi stage decision processes. In this technique the problem is divided in to small sub problem(stages) which are solve successively and thus forming a sequences of decision which leads to an optimal solution of the problem .This program has no standard mathematical formation of dynamic programming problem Rather, Dynamic programming is a general approach to solve optimization problems.

3.1.1. Basic features of Dynamic programming

The basic features of dynamic programming are:-

- The problem can be divided (decomposed) in to sub problem which are called stages.
- At each stage the system is characterized by a small set of parameters called state variables.
- At each stage there is a choice of a number of decisions.
- The effect of decision at each stage is to transform the current state in to a state of a system at the next stage.
- Given the current state, an optimal decision for the remaining stage is independent of the decision taken the previous stage.
- The purpose of the processes in to optimize a predefine function of the state variable called the objective function.

3.1.2. Classification of Dynamic Programming

Dynamic programming problem can be classified depend on the following conditions.

1. Dynamic programming problems can be classified depend on nature of the data available as Deterministic and stochastic or probabilistic models.
   - ✓ In deterministic model, the outcome at any decision stage is unique, determined known.
   - ✓ In probabilistic or stochastic model, there is a set of possible out comes with some probability distribution.
2. The possible decisions at any stage from which we are to choose one are called state it may be finite.
3. The total number of stage in the process is may be finite and may be known,.

3.1.3. Definition of some terms

Stage: - It signifies a portion of the total problem for which a decision can be taken. At each stage there are a number of alternatives and the best out of those is called stage decision , which may be optimal for that stage .But it contribute to obtain the optimal decision policy.

State: - The condition of the decision process at a stage is called state. The variable, which specifies the condition of the decision process that describes the status of the system at a particular stage are called state variables.

Policy: - A rule, which determines the decision at each stage is known as policy. A policy is optimal one if the decision made at each stage in a way that the result of the decision is optimal overall the stage and not only for the current stage.

3.1.4. Steps for Dynamic Programming Problem

- ➢ Mathematical formulation of the problem and to write the recursive equation or (recursive relation connecting the optimal decision function for the "n" stages problem with the optimal decision function for the (n-1) stage sub problem).
- ➢ To write the relation giving the optimal decision function for one stage sub-problem and solve it.
- ➢ To solve the optimal decision function for two stage, three stage, ..., n-1 stage and then n-stage problem.

There are three basic elements that characterizes dynamic programming algorithm.

1. Substructure

   Decompose the given problem into smaller sub-problems. It expresses the solution of the original problem in terms of solution for smaller problems.

2. Table structure

   After solving sub-problem, store the answer to sub-problems in a table form. This is done typically sub-problem solutions are reused many times and we do not want to repeatedly solve the same problem over and over again.

3. Bottom-up computation

   Using table combine solution of smaller sub-problem to solve large sub-problems and eventually arrive at a solution to the complete problem.

Bottom-up means

i. Start with the smallest sub-problems.
ii. Combining their solutions obtain the solutions to sub-problem of increasing size.
iii. Until arrive at the solution of the original problem.

3.2. Knapsack Problem

Consider a well-known problem of leading a vessel with different item, so that the total value of its content is maximum subject to the constraint that the total weight must not exceed a specific limit. Any IP that has only one constraint is referred to as a knapsack problem. Furthermore, the coefficients of this constraint and the objective are all non-negative. The traditional story is that: There is a knapsack. There are a number of items, each with a size and a value. The objective is to maximize the total value of the items in the knapsack.

Mathematically, the problem can be stated as:

$$\text{Max } z = V_1x_1 + V_2x_2 + \cdots + V_nx_n$$

$$\text{s.t} \quad w_1x_1 + w_2x_2 + \cdots + w_nx_n \leq W \qquad (3.1)$$

x_i(i=1, 2, 3…., m) are non-negative integers.

$x_i$is the number of unit of item i, i=1, 2, 3…..,m and W is the maximum allocated weight.

For dynamic programming formulation, let as consider the item as stage and the state of the system be defined as the weight capacity available.

Let $f_n(s_n, x_{n,})$ be the value of the load for the stages 1,2,...,n when the system is in the state $s_n$and a decision x_n is used.

$f_n^*(s_n)$=total value of the load, when the system is in the state S and an optimal decision is used. Then, $f_n^*(s_n) = max f_n(s_n, x_n) = f_n(s_n, x_n^*)$,where x_n^* is the value of x_n , which maximizes$f_n(s_n, x_n)$. Thus, for the first stage $f_1^*(s_1) = max f_1(s_1, x_1) = \max(v_1, x_1)$ and

$x_1 x_1 \in (0, 1, 2, \dots, \left\lfloor \frac{s_1}{w_1} \right\rfloor)$

$f_n^*(s_n) = \max [v_n x_n + f_{n-1}^*(s_n - w_n x_n)]$

$x_1 \in \left(0,1, \dots, \left\lfloor \frac{s_1}{w_1} \right\rfloor\right)$ Where, n=2, 3, … ,N and $\left\lfloor \frac{s_n}{w_n} \right\rfloor$ is the largest integer in $\left\lfloor \frac{s_n}{w_n} \right\rfloor$.

Theorem 2.1 If $x_0$isan optimal solution of the problem

$$max\, L(x, \lambda) = f(x) - \sum_{i=1}^{m} \lambda_i\, g_i(x)$$

$$\text{s.t } g_i(x) \le b_i, i = 1, 2, \dots, m \quad \dots\dots\dots\dots\dots\dots \quad (3.2)$$

$$x \in s$$

For a set of real non negative Lagrange multipliers (λ_1, λ_2, … , λ_n) = λ^T

Then x_0 is an optimal solution of the problem

$$\text{Max } f(x)$$

$$\text{s.t } g_i(x) \le g_i(x_0), i = 1, 2, \dots, m \quad \dots\dots\dots\dots\dots\dots \quad (3.3)$$

Proof: -since x_0 maximize $L(x, \lambda)$ over S, we have,

$f(x_0) - \sum_{i=1}^{m} \lambda_i g_i(x_0) \ge f(x) - \sum_{i=1}^{m} \lambda_i g_i(x)$ for all $x \in s$.

$f(x_0) - f(x) > \sum_{i=1}^{m} \lambda_i [g_i(x_0) - g_i(x)] \ge 0$ For all $x \in S$, satisfying $g_i(x) \le g_i(x_0)$, since all $\lambda_i \ge 0$.

Hence x_0 is the solution of (3.3).

Now, If $g_i(x_0)$ =b_i, i=1, 2, … , m, x_0 is also an optimal solution, an optimal solution of the original problem.

Theorem 2.2:- Let $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m) \ge 0$ and $\mu = (\mu_1, \mu_2, \dots, \mu_m) \ge 0$ be any two sets of Lagrange multipliers such that λ_i=μ_i, i=1, 2, … , $m; i \ne k$

$\lambda_k > \mu_k$ …….. (3.4)

And $x_{0\lambda}, x_{0\mu}$ are the corresponding optimal solutions of the problem. (3.4) Then $g_k(x_{0\lambda})$ is monotonically decreasing function of λ_k.

Proof: -since$x_{0\lambda}$ $maximizes\ L(x, \lambda)$, we have

$$f\left(x_{0\lambda}\right)-\sum_{i=1}^{m}\lambda_i g_i\left(x_{0\lambda}\right)\geq f(x_{0\mu})-\sum_{i=1}^{m}\lambda_i g_i\left(x_{0\mu}\right)$$

And since $x_{0\mu}\ maximizes\ L(x,\mu),we\ have$

$$f(x_{0\mu})-\sum_{i=1}^{m}\mu_i g_i(x_{0\mu})\geq f\left(x_{0\lambda}\right)-\sum_{i=1}^{m}\mu_i g_i\left(x_{0\lambda}\right)$$

Adding these inequalities and rearranging, we get

$$\sum_{i=1}^{m}(\lambda_i-\mu_i)[g_i(x_{0\lambda})-g_i(x_{0\mu})]\leq 0$$

Hence be (3.4) $g_k\left(x_{0\lambda}\right)-g(x_{0\mu})\leq 0$

Example: - Consider the following simple numerical problem with the data as given below and the total carry is 7.

Table 3.1

Item(i)	Weight(w)	Value(v)
1	**3**	**5**
2	4	8
3	2	4

Solution: - The Mathematical model of this problem is:

Max Z=$V_1x_1+V_2x_2+V_3x_3$

s.t $w_1x_1+w_2x_2+w_3x_3\leq W$

$x_1,x_2,x_3\geq 0$

Then, Max Z = $5x_1+8x_2+4x_3$

s.t $3x_1+4x_2+2x_3\leq 7$

$x_1,x_2,x_3\geq 0$

Stage 1: For the first stage process the largest x_1 value is$\left\lfloor\frac{s_1}{w_1}\right\rfloor$, Since s_1 = {0, 1, 2,…., 7}, the largest value of x_1 is $\left\lfloor\frac{s_1}{w_1}\right\rfloor=\left\lfloor\frac{7}{3}\right\rfloor=2$

Table 3.2

$f_1(s_1,x_1)=v_1x_1=5x_1$				Optimal solution	
s_1 \ x_1	0	1	2	$f_1^*(s_1)$	x_1^*
0	0	-	-	0	0
1	0	-	-	0	0
2	0	-	-	0	0
3	0	5	-	5	1
4	0	5	-	5	1
5	0	5	-	5	1
6	0	5	10	10	2
7	0	5	10	10	2

Stage 2: For the two stage process the largest value of x_2 is $\left\lfloor \frac{s_2}{w_2} \right\rfloor = \left\lfloor \frac{7}{4} \right\rfloor = 1$. the state variables are s_2=0, 1, 2……7

Table 3.3

$f_2(s_2, x_2) = v_2x_2 + f_1^*(s_2 - w_2x_2) = 8x_2 + f_1^*(s_2 - 4x_2)$			Optimal solution	
s_2 \ x_2	0	1	$f_2^*(s_2)$	x_2^*
0	0+0=0	-	0	0
1	0+0=0	-	0	0
2	0+0=0	-	0	0
3	0+5=5	-	5	0
4	0+5=5	8+0=8	8	1
5	0+5=5	8+0=8	8	1
6	0+10=10	8+0=8	10	0
7	0+10=10	8+5=13	13	1

Stage 3: For the three stages process the largest value of x_3 is $\left\lfloor \frac{s_3}{w_3} \right\rfloor = \left\lfloor \frac{7}{2} \right\rfloor = 3$

Table 3.4

$f_3(s_3, x_3) = v_3x_3 + f_2^*(s_3 - w_3x_3) = 4x_3 + f_2^*(s_3 - 2x_3)$					Optimal solution	
s_3 \ x_3	0	1	2	3	$f_3^*(s_3)$	x_3^*
0	0+0=	-	-	-	0	0
1	0+0=0	-	-	-	0	0
2	0+0=0	4+0=4	-	-	4	1
3	0+5=5	4+0=4	-	-	5	0
4	0+8=8	4+0=4	8+0=8	-	8	0,2
5	0+8=8	4+5=9	8+0=8	-	9	1
6	0+8=8	4+8=12	8+0=8	12+0=12	12	1,3
7	0+13=13	4+8=12	8+5=13	12+0=12	13	0,2

The optimal solutions are;

If x_3=0, we have a chance to load the total weight 7 in x_1 *and* x_2.

When we read the max $(f_2^*(s_2))$=13 is occurred at x_2=1, since its weight is 4 the remaining weight is 3 units out of 7 units (weight).Thus, we can add x_1=1 that means 3 units fromx_1.

Therefore,$x_1 = 1, x_2 = 1$ *and* $x_3 = 0$ .The optimal solution 13 that means

Max Z=$v_1x_1 + v_2x_2 + v_3x_3$=$5x_1 + 8x_2 + 4x_3$=5+8+0=13

3.3. Branch-and-Bound Method

The most effective general purpose optimal algorithm is an LP-based tree search approach called as ***branch and bound*** (B&B). The method was first put forward in the early 1960's by Land and Doig. This is *a way of systematically (implicitly) enumerating* feasible solutions such that the optimal integer solution is found.

A **branch-and-bound algorithm** consists of a systematic enumeration of candidate solutions by **means** of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The **algorithm** explores branches of this tree, which **represent** subsets of the solution set.

In this section we provide a relatively fast algorithm for solving IPs and MILPs. The general idea of the algorithm is to solve LP relaxations of the IP or MILP and to look for an integer solution by branching and bounding on the decision variables provided by the LP relaxations.

Steps (Branch-and-bound algorithm).

Step 1 Solve the LP relaxation of the problem. If the solution is integer where require, then we are done. Otherwise create two new sub problems by **branching** any fractional variable that is required to be integer.

Step 2 A sub problem is **not active** when any of the following occurs:

a. You used the sub problem to branch on, i.e., the sub problem is not a leaf in the tree.
b. All variables in the solution that are required to be integers are integers.
c. The sub problem is infeasible.
d. You can fathom the sub problem by a **bounding** argument.

Choose an **active** sub problem and branch on a fractional variable that should be integer in the final solution. Repeat until there are no active sub problems.

Step 3 Solution to the IP/MILP is the best IP/MILP solution of the sub problems you have created. It is found in one of the leaves of the tree representing the sub problems.

Example (Furniture with integrity). The Integrity Furniture Corporation manufactures tables and chairs. A table requires 1 hour of labor and 9 units of wood. A chair requires 1 hour of labor and 5 units of wood. Currently 6 hours of labor and 45 units of wood are available. Each table contributes 8 to profit, and each chair contributes 5 to profit. Integrity

Furniture's want to maximize its profit. This problem would be a classical product selection problem, but we insist that only integral furniture is produced. So, the problem is no longer an LP but an IP.

Solution:

Let us formulate and solve the (pure) IP

Let x_1 = number of tables manufactured;

x_2 = number of chairs manufactured:

Since x_1 and x_2 must be integers, Integrity Furniture wishes to solve the following (pure) IP:

$$\begin{aligned} \max z &= 8x_1 + 5x_2 \\ s.t\ x_1 + x_2 &\le 6 \qquad (labor) \\ 9x_1 + 5x_2 &\le 45 \qquad (wood) \\ x_1, x_2 &\ge 0 \qquad \text{Integer} \end{aligned} \tag{3.5}$$

The first step in the branch-and-bound method is to solve the LP relaxation of the IP (3.1)

$$\begin{aligned} \max z &= 8x_1 + 5x_2 \\ s.t\ x_1 + x_2 &\le 6 \qquad (labor) \\ 9x_1 + 5x_2 &\le 45 \qquad (wood) \\ x_1, x_2 &\ge 0 \end{aligned} \tag{3.6}$$

Then change into standard of linear programming problem using slack variables, s_1 and s_2.

$$\begin{aligned} \max z &= 8x_1 + 5x_2 \\ s.t\ x_1 + x_2 + s_1 &= 6 \qquad (labor) \\ 9x_1 + 5x_2 + s_2 &= 45 \qquad (wood) \\ x_1,\ x_2,\ s_1,\ s_2 &\ge 0 \end{aligned} \tag{3.7}$$

If we are lucky, all the decision variables (x_1 for tables and x_2 for chairs in the example we are considering) in the LP relaxation's optimum turn out to be integers. In this lucky case the optimal solution of the LP relaxation is also the optimal solution to the original IP.

In the branch-and-bound algorithm we use next, we call the LP relaxation (3.7) of the IP (3.6) sub problem 1 (SP 1). After solving SP 1 we find the solution to be

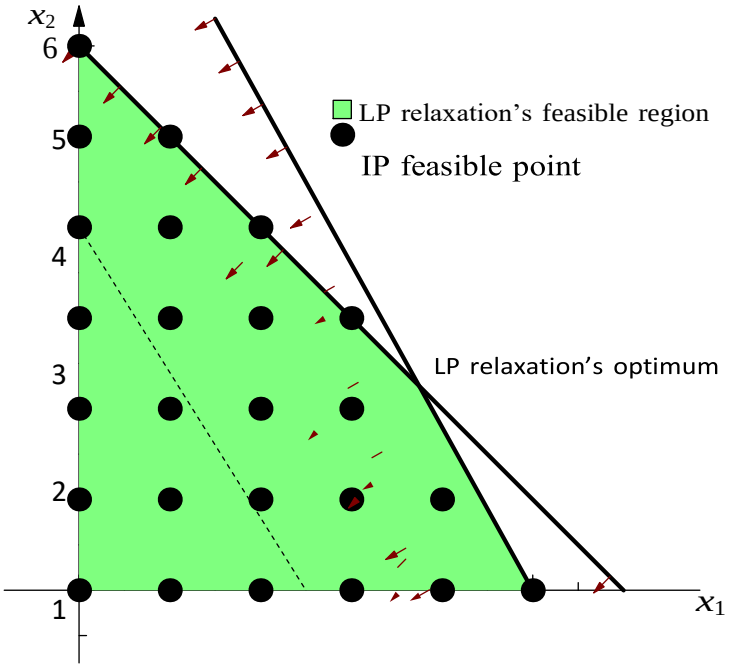
$$z = \frac{165}{4}$$

$$x_1 = \frac{15}{4}$$

$$x_2 = \frac{9}{4}$$

This means that we were not lucky: the decision variables turned out to be fractional.

So, the LP relaxation (3.7) has (possibly) a better optimum than the original IP (3.6). In any case, we have found an upper bound to the original IP: Integrity Furniture Corporation of Example cannot possibly have better profit than $\frac{165}{4}$.



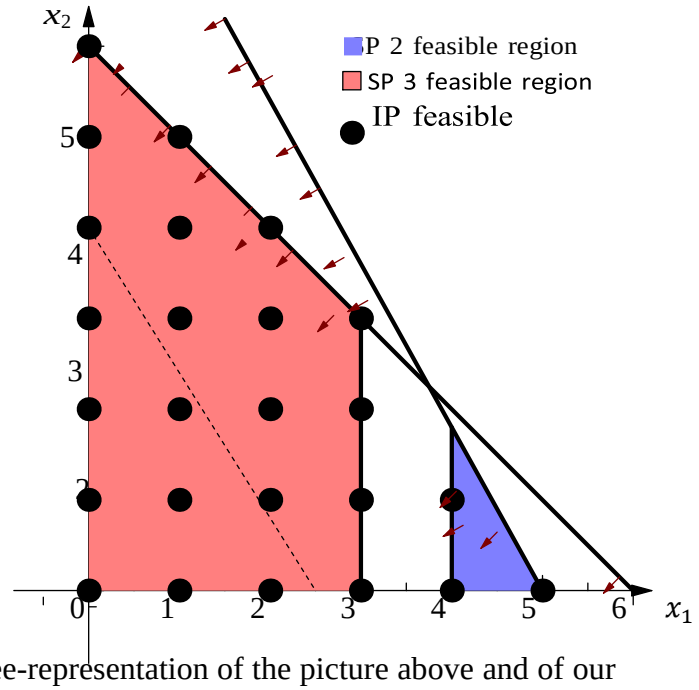
Next we split the feasible region (painted light green in the picture above) of the LP relaxation (3.7) in hope to find a solution that is an integer one. We arbitrarily choose a variable that is fractional at the optimal solution of the LP SP 1 (the first LP relaxation). We choose x_1. Now, x_1 was $15/4 = 3.75$.

at the optimal solution to the IP we have either $x_1 \geq 4$ or $x_1 \leq 3$, since third alternative $3 < x_1 < 4$ is out of the question for IPs. So, we consider two possible cases $x_1 \geq 4$ and $x_1 \leq 3$ as separate sub problems. We denote sub problem as SP2 and SP3

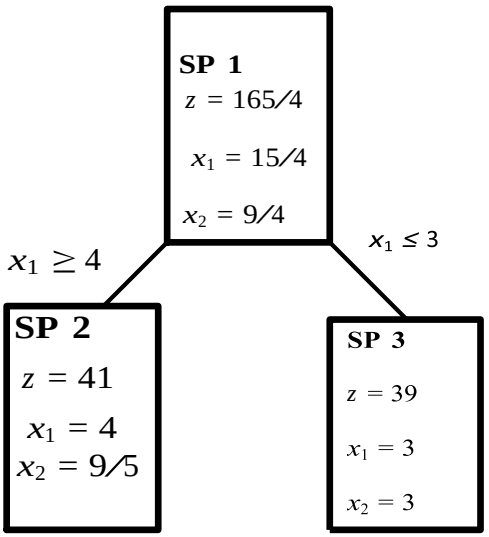
$$SP2 = SP\ 1\ +\ "\ x_1 \geq 4",$$

$$SP3 = SP\ 1\ +\ "x_1 \leq 3".$$

In the next picture we see that every possible feasible solution of the Integrity Furniture's IP (3.6) (the bullet points) is included in the feasible region of either SP 2 or SP 3. Since SP 2 and SP 3 were created by adding constraints involving the fractional solution x_1, we say that SP 2 and SP 3 were created by **branching** on x_1



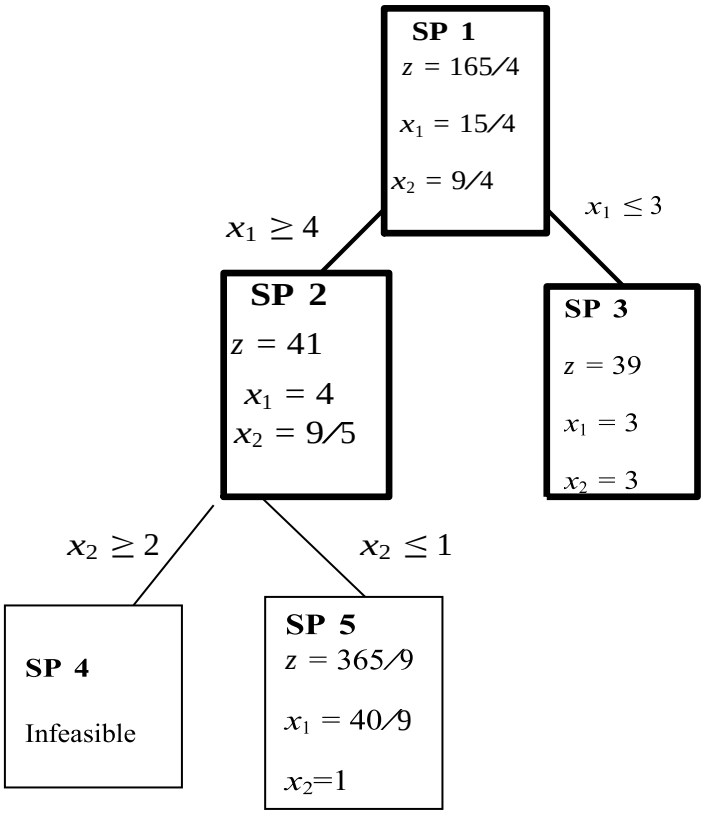
Below we have a tree-representation of the picture above and of our branching- and-bounding so far.



We see that SP 3 has an integer solution. So, we do not have to branch SP 2 anymore. Unfortunately, the integer solution of SP 3, $z = 39$, is suboptimal when compared to the non-integer solution, $z = 41$, of SP 2. So, it may turn out that the SP 2 has a further sub problem that has better integer solution than SP 3. So, we have to branch SP 2 to find out what happens. Since $x_1 = 4$ is an integer, we branch on $x_2 = 9/5 = 1.8$. So, we have the new sub problems of SP 2:

$$SP\ 4 = SP\ 2 + "\ x_2 \geq 2",$$
$$SP\ 5 = SP\ 2 + "\ x_2 \leq 1".$$

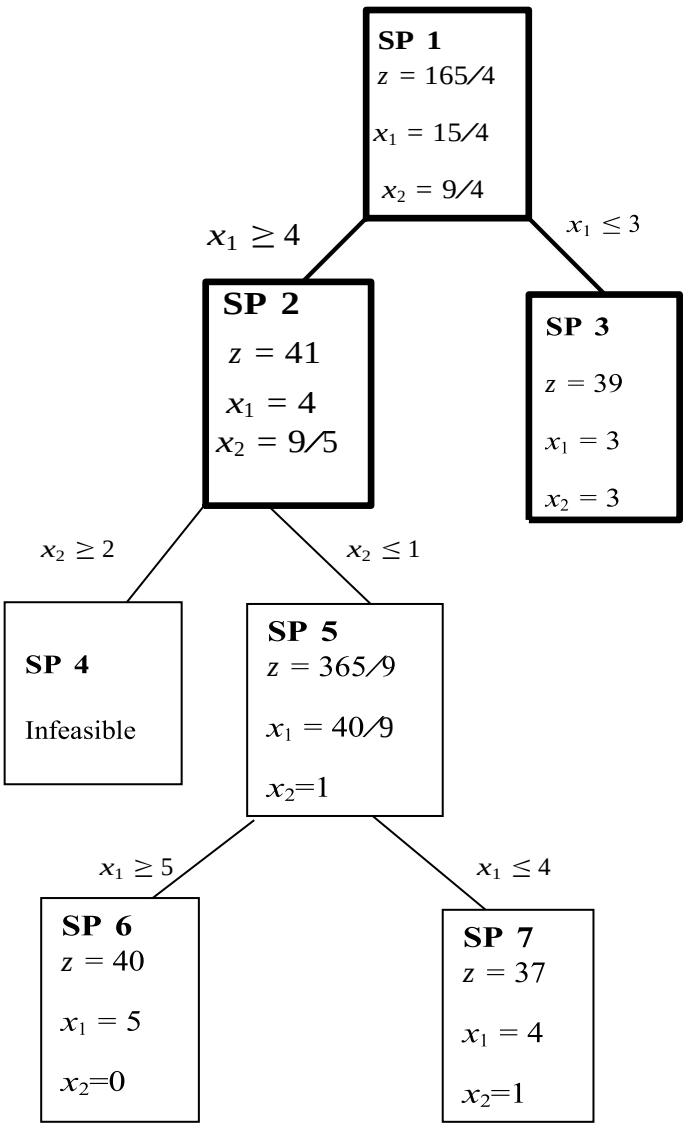
When the solutions to these sub problems are added to the tree above we get the following tree.



We see that **SP 4** Infeasible So, the optimal solution is not there. However, SP 5 give us a non-integer solution that is better than the integer solution of SP 3. So, we have to branch on SP 5. Since $x_2 = 1$ is already an integer, we branch on $x_1 = 40/9 = 4.444$. So, we get two new sub problems of SP 5:

$$SP\ 6 = SP\ 5 + "x_1 \geq 5",$$
$$SP\ 7 = SP\ 5 + "x_1 \leq 4".$$

After this branching we finally arrive at the final solution where all the sub problems are either infeasible, have integer solutions, or are suboptimal to some sub problems that have integer solution, i.e. we have integer bounds for the sub problems. This is the bound part of the branch–and–bound.



From the tree above can read the solution to the IP: The SP 6 is the optimal sub problem with integer solution. So, the solution to the IP (3.6) is

$$\begin{aligned} z &= 40, \\ x_1 &= 5, \\ x_2 &= 0. \end{aligned}$$

General Branch-And-Bound Algorithm

We have solved a pure IP (Integer programming) with branch and bound. To solve (MILP Mixed Integer Linear programming) with branch- and-bound one follows the same steps as in the pure IP case **except** one only branch on decision variables that are required to be integers. So, solving MILPs is actually somewhat easier than solving pure IPs!

Exercise: 1. solve the following pure integer programming problem using branch-and-bound method.

$$\text{Max } Z = 7x_1 + 3x_2$$
$$s.t\ 2x_1 + 5x_2 \leq 30$$
$$8x_1 + 3x_2 \leq 48$$
$$x_1, x_2 \geq 0 \quad integers$$

2. Solve the following mixed integer programming problem using branch- and-bound method.

$$\text{Max } Z = 2x_1 + x_2 + 3x_2$$
$$s.t\ 4x_1 + 3x_2 - 3x_3 \leq 6$$
$$2x_1 + 3x_2 + 3x_2 \leq 4$$
$$x_i \geq 0 \quad for\ i = 1,2,3,\ x_1, x_2\ integers$$

Chapter Four

Graph Theory and Network Optimization

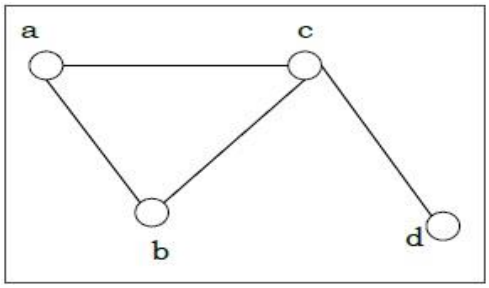
Objectives of the chapter

- On completion of the chapter, successful students will be able to:
  - Define a graph and also directed graph,
  - Understand the concept of Paths and circuits,
  - Define Trees and forests,
  - Apply graph theory to solve network oriented problems,

4.1. Basic notations of graph theory

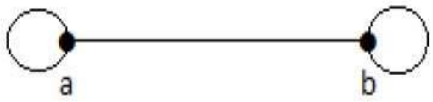
Graph: A **graph** G consists of a collection V of **vertices** and a collection **edges** E, for which we write G = (V, E). Each edge $e \in E$ is said to **join** two vertices, which are called its **end points**. If e joins $u, v \in V$, we write $e = <u, v>$. Vertex u and v in this case are said to be **adjacent**. Edge e is said to be **incident** with vertices u and v, respectively. We will often write V(G) and E(G) to denote the set of vertices and edges associated with graph G, respectively. A graph that does not have loops or multiple edges is called **simple**. A set of *vertices* (also called *nodes* or *points*).

Example: Let us consider, a Graph is G = (V, E) where V = {a, b, c, d} and E = {{a, b}, {a, c}, {b, c}, {c, d}}



Loop: If an edge is drawn from vertex to itself, it is called a loop.

Example



In this graph, there are two loops which are formed at vertex a, and vertex b.

Definition: The number of edges incident with a vertex v is called the **degree** of v, denoted as deg(V). Loops are counted twice.

In a simple graph with n number of vertices, the degree of any vertices is $\deg(v) \leq n - 1 \; \forall v \in G$

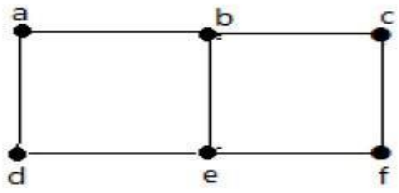
A vertex can form an edge with all other vertices except by itself. So the degree of a vertex will be up to the **number of vertices in the graph minus 1**. This 1 is for the self-vertex as it cannot form a loop by itself. If there is a loop at any of the vertices, then it is not a Simple Graph.

Adjacency

Here are the norms of adjacency

- In a graph, two vertices are said to be **adjacent,** if there is an edge between the two vertices. Here, the adjacency of vertices is maintained by the single edge that is connecting those two vertices.
- In a graph, two edges are said to be adjacent, if there is a common vertex between the two edges. Here, the adjacency of edges is maintained by the single vertex that is connecting two edges.

Example



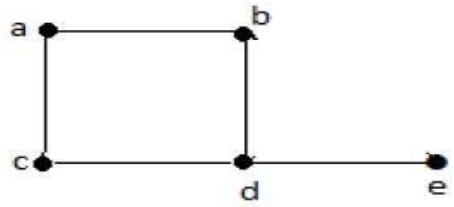
In the above graph

- ‘a’ and ‘b’ are the adjacent vertices, as there is a common edge ‘ab’ between them.
- ‘a’ and ‘d’ are the adjacent vertices, as there is a common edge ‘ad’ between them.
- ab’ and ‘be’ are the adjacent edges, as there is a common vertex ‘b’ between them.
- be’ and ‘de’ are the adjacent edges, as there is a common vertex ‘e’ between them.

Degree Sequence of a Graph

If the degrees of all vertices in a graph are arranged in descending or ascending order, then the sequence obtained is known as the degree sequence of the graph.

Example



Vertex	A	b	c	d	e
Connecting to	b,c	a,d	a,d	c,b,e	d
Degree	2	2	2	3	1

In the above graph, for the vertices {d, a, b, c, e}, the degree sequence is {3, 2, 2, 2, 1}.

4.2. Paths and circuits

Definition. A path is a sequence of edges that begins at a vertex, and travels from vertex to vertex along edges of the graph. The number of edges on the path is called the **length** of the path.

A **path** in a graph is a succession of adjacent edges, with no repeated edges, that joins two vertices.

Definition. A **circuit** is a **path** which joins a node to itself.

Definition. An Euler **path** in a graph without isolated nodes is a **path** that contains every edge exactly one. If such a path is also a circuit, it is called an Euler circuit.

4.3. Trees and forests

If G is a connected graph without any cycles then G is called a **tree**. (If |V | = 1, then G is connected and hence is a tree.) A tree is also called a free tree.

A *forest* is an undirected graph, in which any two vertices are connected by *at most one* path, or equivalently an acyclic undirected graph, or equivalently a disjoint union of trees.

Spanning Trees

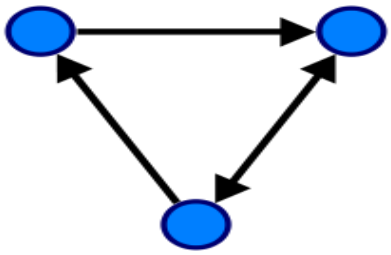
Given a graph $G = (V, E_G)$, a spanning tree T of G is a connected acyclic sub graph of G with the same vertex set V. i.e., $T = (V, E_T)$ for some subset of edges $E_T \subseteq E_G$, and is a tree.

Note that if G is not itself connected, it has no spanning trees. Else it has at least one spanning tree. To see this, just take the connected graph, and if dropping some edge from it does not cause the graph to become disconnected, drop it and continue. What remains must be a spanning tree.

4.4. Directed graphs

Definition: A **directed graph** or **digraph** D consists of a collection **vertices** V, and a collection of **arcs** A, for which we write D = (V, A). Each arc $a = (\overrightarrow{u,v})$ is said to join vertex $u \in V$ to another (not necessarily distinct) vertex v. Vertex u is called the **tail** of a, whereas v is its **head**.

A directed graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network. In contrast, a graph where the edges are bidirectional is called an undirected graph.



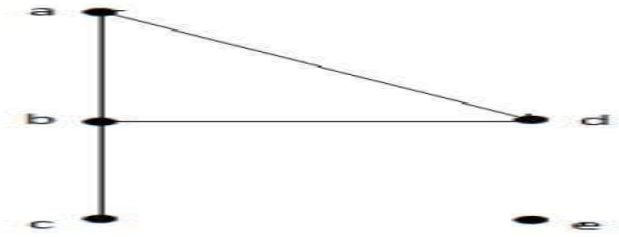
This graph is a directed graph with three vertices and four directed edges (the double arrow represents an edge in each direction).

Degree of vertex can be considered fewer than two cases of graphs.

- Undirected Graph
- Directed Graph

Degree of Vertex in an Undirected Graph: An undirected graph has no directed edges.

Example



In the above Undirected Graph,

- deg(a) = 2, as there are 2 edges meeting at vertex ‘a’.
- deg(b) = 3, as there are 3 edges meeting at vertex ‘b’.
- deg(c) = 1, as there is 1 edge formed at vertex ‘c’

  So ‘c’ is a **pendent vertex**.

- deg(d) = 2, as there are 2 edges meeting at vertex ‘d’.
- deg(e) = 0, as there are 0 edges formed at vertex ‘e’.

  So ‘e’ is an **isolated vertex**.

Degree of Vertex in a Directed Graph

In a directed graph, each vertex has an **indegree** and an **outdegree**.

Indegree of a Graph

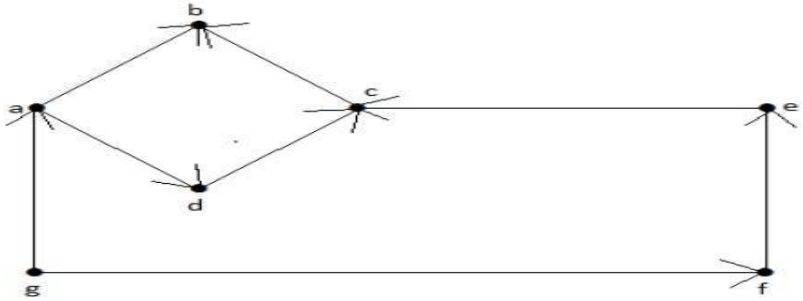
- Indegree of vertex V is the number of edges which are coming into the vertex V.
- **Notation**: $\deg^-(V)$.

Outdegree of a Graph

- Outdegree of vertex V is the number of edges which are going out from the vertex V.
- **Notation**: $\deg^+(V)$.

Example

Take a look at the following directed graph. Vertex ‘a’ has two edges, ‘ad’ and ‘ab’, which are going outwards. Hence its outdegree is 2. Similarly, there is an edge ‘ga’, coming towards vertex ‘a’. Hence the indegree of ‘a’ is 1.

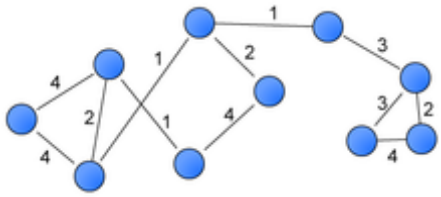


The indegree and outdegree of other vertices are shown in the following table:

Vertex	Indegree	Outdegree
a	1	2
b	2	0
c	2	1
d	1	1
e	1	1
f	1	1
g	0	2

Weighted graph

A *weighted graph* or a *network* is a graph in which a number (the weight) is assigned to each edge. Such weights might represent for example costs, lengths or capacities, depending on the problem at hand. Such graphs arise in many contexts, for example in shortest path problems such as the traveling salesman problem.



4.5. Matrix representation of a graph

4.5.1. Adjacency Matrices

Carrying out graph algorithms using the representation of graphs by lists of edges, or by adjacency lists, can be cumbersome if there are many edges in the graph. To simplify computation, graphs can be represented using matrices. Two types of matrices commonly used to represent graphs will be presented here. One is based on the adjacency of vertices, and the other is based on incidence of vertices and edges.

Suppose that G = (V, E) is a simple graph where |V | = n. Suppose that the vertices of G are listed arbitrarily as v_1, v_2,..., v_n. The adjacency matrix A (or A_G) of G, with respect to this listing of the vertices, is the n x n zero–one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent. In other words, if its adjacency matrix is

A = [a_{ij}], then $a_{ij} = \begin{cases} 1 \; if \; \{v_i, v_j\} is \; an \; edge \; of \; G, \\ 0 \qquad\qquad otherwise. \end{cases}$

Example: - Use an adjacency matrix to represent the graph shown in Figure

Solution: We order the vertices as a, b, c, d. the matrix representing this graph is

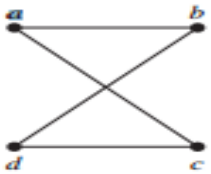
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Example: Draw a graph with the adjacency matrix

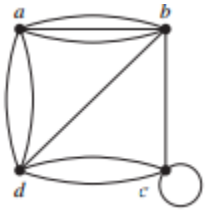
$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

with respect to the ordering of vertices a, b, c, d.

Solution: A graph with this adjacency matrix is shown



Example: Use an adjacency matrix to represent the pseudo graph shown in Figure



Solution: The adjacency matrix using the ordering of vertices a, b, c, d is

$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}.$$

We used zero–one matrices to represent directed graphs. The matrix for a directed graph G = (V, E) has a 1 in its (i, j) th position if there is an edge from v_i to v_j, where v_1, v_2,..., v_n is an arbitrary listing of the vertices of the directed graph. In other words, if $A = [a_{ij}]$ is the adjacency matrix for the directed graph with respect to this listing of the vertices, then

$$a_{ij} = \begin{cases} 1 \; if \; \{v_i, v_j\} is \; an \; edge \; of \; G, \\ 0 \qquad\qquad\qquad otherwise. \end{cases}$$

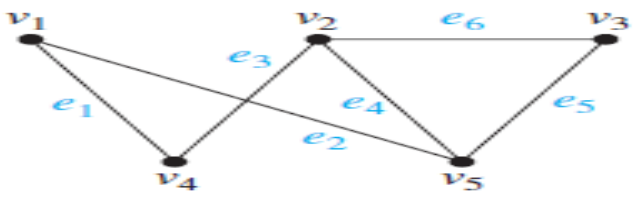
The adjacency matrix for a directed graph does not have to be symmetric, because there may not be an edge from v_j to v_i when there is an edge from v_i to v_j . Adjacency matrices can also be used to represent directed multi graphs. Again, such matrices are not zero–one matrices when there

are multiple edges in the same direction connecting two vertices. In the adjacency matrix for a directed multi graph, a_{ij} equals the number of edges that are associated to (v_i, v_j)

4.5.2. Incidence Matrices

Another common way to represent graphs is to use **incidence matrices**. Let $G = (V, E)$ be an undirected graph. Suppose that $v_1, v_2, \ldots v_n$ are the vertices and $e_1, e_2, \ldots, e_m$ are the edgesof G. Then the incidence matrix with respect to this ordering of V and E is the $n \times m$ matrix $\mathbf{M} = [m_{ij}]$, where $m_{ij} = \begin{cases} 1 & when\ edge\ e_j\ is\ incident v_i, \\ o & otherwise. \end{cases}$

Example: - Represent the graph shown in the fig below

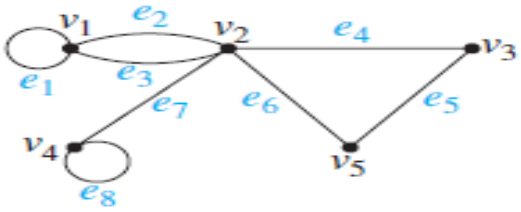


Solution: The incidence matrices is

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}.$$

Incidence matrices can also be used to represent multiple edges and loops. Multiple edges are represented in the incidence matrix using columns with identical entries, because these edges are incident with the same pair of vertices. Loop is represented using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with loop.

Example: Represent the pseudo graph shown in the fig below using an incidence matrices



Solution: The incidence matrices for this graph is

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}.$$

Note: the incidence matrix of a graph is not unique for it depends on the orderings of the vertices and the edges of the graph.

4.5. Network flow

A **network** is a graph $G = (V, E)$, where V is a set of vertices and E is a set of V's edges – a subset of $V \times V$ together with a non-negative function $c: V \times V \to \mathbb{R}_\infty$, called the **capacity** function. Without loss of generality, we may assume that if $(u, v) \in E$ then (v, u) is also a member of E, since if $(v, u) \notin E$ then we may add (v, u) to E and then set $c(v, u) = 0$.

If two nodes in G are distinguished, a source s and a sink t, then (G, c, s, t) is called a **flow network**.

A flow network is a connected, directed graph G = (V; E).

- Each edge e has a non-negative, integer capacity c_e.
- A single source $s \in V$.
- A single sink $t \in V$.
- No edge enters the source and no edge leaves the sink.

4.6. Minimum and critical path problems

The sources, destinations, and intermediate points are collectively called *nodes* of the network, and the problem links connecting nodes are termed *arcs*.

The vertices with equal earliest and latest starting times define the critical path.

The minimum cost flow problem holds a central position among network optimization models, both because it encompasses such a broad class of applications and because it can be solved extremely efficiently. Like the maximum flow problem, it considers flow through a network with limited arc capacities. Like the shortest-path problem, it considers a cost (or distance) for flow through an arc.

The minimum cost flow problem is described below.

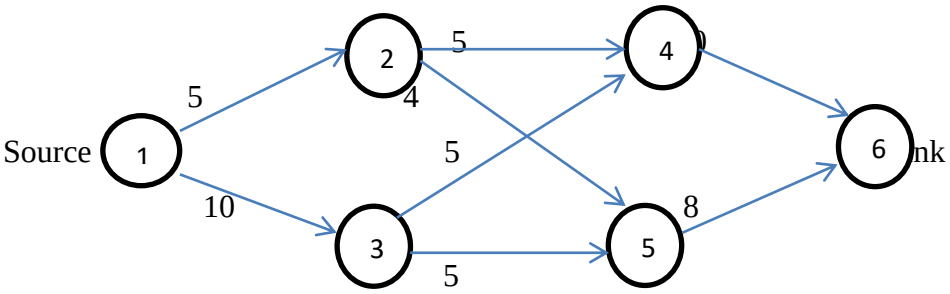
1. The network is a *directed* and *connected* network.
2. *At least one* of the nodes is a *supply node.*
3. *At least one* of the other nodes is a *demand node.*
4. All the remaining nodes are *transshipment nodes.*
5. Flow through an arc is allowed only in the direction indicated by the arrowhead, where the maximum amount of flow is given by the *capacity* of that arc. (If flow can occur in both directions, this would be represented by a pair of arcs pointing in opposite directions.)
6. The network has enough arcs with sufficient capacity to enable all the flow generated at the *supply nodes* to reach all the *demand nodes.*
7. The cost of the flow through each arc is *proportional* to the amount of that flow, where the cost per unit flow is known.
8. The objective is to minimize the total cost of sending the available supply through the network to satisfy the given demand. (An alternative objective is to maximize the total profit from doing this.)

4.7. Maximal flow problems

This problem involves a directed network with arcs carrying flow. The only relevant parameter is the upper bound on arc flow, called *arc capacity*. The problem is to find the maximum flow that can be sent through the arcs of the network from some specified node *s*, called the source, to a second specified node *t*, called the sink. Applications of this problem include finding the maximum flow of orders through a job shop, the maximum flow of water through a storm sewer system, and the maximum flow of product through a product distribution system, among others.
Let a directed with a capacitated network (V, E, C) connecting a source (origin) node with a sink (destination) node.

- ✓ The set V is the set of nodes in the network.
- ✓ The set E is the set of directed links (i, j)
- ✓ The set C is the set of capacities $c_{ij} \geq 0$ of the links $(i, j) \in E$.

The problem is to determine the maximum amount of flow that can be sent from the source node to the sink node.



This is Max-Flow Problem for single-source and single-sink

We want to formulate the max-flow problem.

- For each link $(i,j) \in E$, let x_{ij} denote the flow sent on link (i,j),
- For each link $(i,j) \in E$, the flow is bounded from above by the capacity c_{ij} of the link: $c_{ij} \geq x_{ij} \geq 0$

❖ We have to specify the balance equations

- All the nodes in the network except for the source and the sink node are just "transit" nodes (inflow=outflow)

  $\sum_{\{l|(l,i)\in E\}} x_{li} - \sum_{\{j|(i,j)\in E\}} x_{ij} = 0$, for all $i \neq s,t$

- The objective is to maximize the outflow from the source node s

$$\sum_{\{j|(s,j)\in E\}} x_{sj}$$

- Alternatively: to maximize the inflow to the sink node t

$$\sum_{\{l|(l,t)\in E\}} x_{lt}$$

❖ MAX-FLOW FORMULATION

$$\text{Maximize} \quad \sum_{\{j:(s,j)\in E\}} x_{sj}$$

$$s.t \sum_{\{l:(l,i)\in E\}} x_{li} - \sum_{\{j|(i,j)\in E\}} x_{ij} = 0 \text{ for all } i \neq s,t$$

$$0 \leq x_{ij} \leq c_{ij} \; for \; all \; (i,j) \in E$$

$$maximize \; x_{12} + x_{13}$$

$$s.t \; x_{12} - x_{25} - x_{24} = 0 \; balance \; for \; node \; 2$$

$$x_{13} - x_{35} - x_{34} = 0 \; balance \; for \; node \; 3$$

$$x_{24} + x_{34} - x_{46} = 0 \; balance \; for \; node \; 4$$

$$x_{25} + x_{35} - x_{56} = 0 \; balance \; for \; node \; 5$$

$$0 \le x_{12} \le 5 \qquad 0 \le x_{13} \le 10$$

$$0 \le x_{24} \le 4 \qquad 0 \le x_{25} \le 5$$

$$0 \le x_{34} \le 5 \qquad 0 \le x_{35} \le 5$$

$$0 \le x_{46} \le 8 \qquad 0 \le x_{56} \le 10$$

Max-Flow is an LP problem: we could use a simplex method

Some Applications

Here are some typical kinds of applications of the maximum flow problem.

- ✓ Maximize the flow through a company's distribution network from its factories to its customers.
- ✓ Maximize the flow through a company's supply network from its vendors to its factories.
- ✓ Maximize the flow of oil through a system of pipelines.
- ✓ Maximize the flow of water through a system of aqueducts.
- ✓ Maximize the flow of vehicles through a transportation network.

4.8. Application of graphs theory

There are many situations where the study of graphs can be used in the Real World.

- Transportation like Airlines, Bus, or Train routes between cities.
- Delivery service like US Postal Service or UPS.
- Telecommunication or computer networks.
- Community planning snow removal or street cleaning.

❖ **Textbook:**

- Bazaraa, Sherali and Shetty, Nonlinear programming, 3rd Edition, Wiley 2006

❖ **References:**

- R. Deumlich, Lecture note for theory of optimization, Dept. of mathematic AAU
- P. Whittle, Optimization under constraints
- Bertsekas, Nonlinear programming, Athena Scientific, 1999
- Fletcher, Practical methods of optimization, Wiley 2000
- Rockafellar, Convex analysis, Princeton, 1970
- Mangasarian, Nonlinear programming, SIAM, 1994
- David G. Luenberger and Yinyu Ye .2008.Linear and Nonlinear Programming, 4^{th} Ed
- Linear Programming: Penn State Math 484Lecture Notes
- Frederick S. Hillier, Gerald J. Lieberman. Introduction to operations research, 2010. 9^{th} ed.