LECTURE NOTES

ON

Engineering Optimization

III B. TECH V SEMESTER

(IARE-R16)

Prepared By:

Mrs.TVanaja,

Assistant Professor

DEPARTMENT OF MECHANICAL ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING

(AUTONOMOUS)

DUNDIGAL, HYDERABAD - 500 043

UNIT – I
INTRODUCTION

Optimization in the design process

In the design process (from the basic ideas to the detailed digital mockup of the product), many aspects should be taken into account.
These requirements are usually connected to safety or economy of the product, but also the manufacturing, using repairing considerations come to the fore. For example the designer has to develop cheapest product, while a large number of additional conditions should be met.

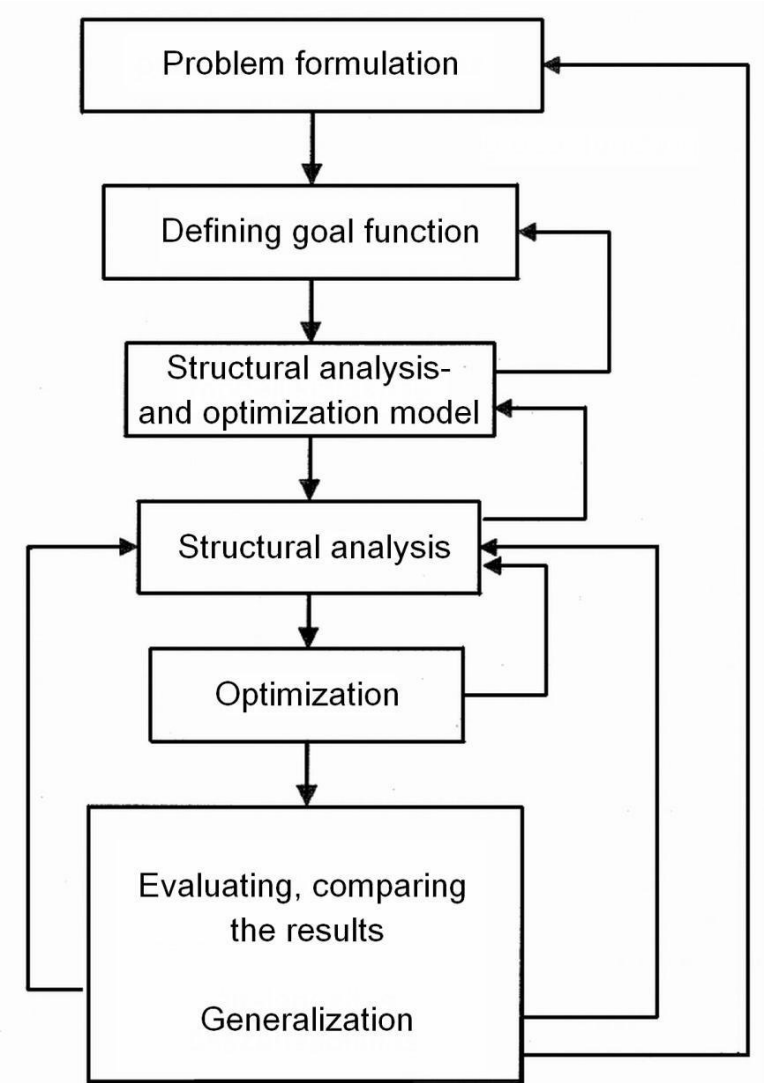


Figure 1.1.: The main steps of solving engineering optimization problem [1.1]

A real design process is usually not a linear sequence, but some steps are repeated, to obtain better solution. The main steps of solving engineering optimization can be seen in Figure 1. It is worth to remark, that the result of the design process, is depending on the accuracy of the introduced steps. For example, if a shape optimization should be taken and the structural responses are calculated numerically, than the error of the structural analysis, can result in accu-

racy in the optimized shape too. The error can come from mesh density or form the incorrect modeling of the real boundary conditions as well.

Last time the usage of 3D CAD systems and numerical structural analysis techniques has been increased in the industrial applications. This is because; using these tools wide range of design processes can be covered. On the other hand the ‒time to market‖ became first objective so the computer-aided technique arises in the early stages of the design process. The modern numerical simulation tools provide an opportunity to decrease the number of the costly and time consuming physical experiments. Although the advanced numerical simulation tools and optimization procedures has a significant role in the product development in the reduction of the time, but the whole design process cannot be automated.
In case of solving structural optimization task we can choose from analytical and numerical methods. The analytical methods are often not suitable for the analysis of complex engineering problems, so this educational material mainly deals with the numerical techniques of the structural optimization. The numerical procedures based on iterative searching techniques which lead to an approximate solution. The searching process continues until the so-called convergence condition is satisfied, so we are sufficiently near the optimal solution.

1.2. The basic elements of an optimization model

1.2.1. Design variables and design parameters

A structural system can be described by a set of quantities, some of which are viewed as variables during the optimization process. Those quantities defining a structural system that are fixed during the automated design are called pre assigned parameters or design parameters and they are not varied by the optimization algorithm. Those quantities that are not pre assigned are called design variables. The pre assigned parameters, together with the design variables, will completely describe a design.

From the mathematical point of view three types of design variables can be distinguished:

The design variables can be considered as continuous or discrete variables. Generally it is easier to work with continuous design variables, but a part of the real world problems contains discrete type of design variables. An intermediate solution, when we know that a large number of discrete design variable values should be considered, then it will be categorized as pseudo discrete. In this case, we solve the task considering this variable a continuous design variable and after the solution the closest possible discrete values will be checked.

From the physical point of view there are four types of the design variables:

1. Mechanical or physical properties of the material (Material design variable)
Material selection presents a special problem. Conventional materials; have discrete properties, as for example a choice is to be made from a discrete set of materials. If there have a few numbers of materials, the task is easier, if we perform the structural analysis for each material separately and comparing the results to choose the optimum material. A typical application is for reinforced composite materials to determine the angles of the reinforcements. Such type of design variables can be considered to be continuous ones.

2. topology of the structure, connecting members of the scheme, or the number of members of the interface schema; (Topology Design Variables)
The topology of the structure can be optimized automatically in certain cases when members are allowed to reach zero size. This permits elimination of some uneconomical members during the optimization process. An example of a topology design variable is if we looking for the optimal truss structure considering one design variable for each truss element (1 if the member exists or 0 if the member is absent). This type of design variables, according to the mathematical classification is not continuous.

3. The shape of the structure (Configurationally or Geometric Design Variables)
This type of design variable leads us to the field of shape optimization. In case of machine design application, the geometry of the part should be modified close to the stress concentration areas, in order to reduce the stresses. On other hands, the material can be removed in the low stress areas, in order to make the structure lighter. So we are looking the best possible shape of the machine part. For example the variable surface of the structure can be described by B-Spline surfaces and the control nodes of such splines can be chosen as a design variable. This is a typical example for shape optimization, and these types of design variables are usually belong to the continuous category.

4. Cross-Sectional Design Variables or the dimensions of the built-in elements
Mainly for historical reasons, size-optimization is a separate category, which has got the simplest design variables. For example, the cross-sectional areas of the truss structure, the moment of inertia of a flexural member, or the thickness of a plate are some examples of this class of design variable. In such cases the design variable is permitted to take only one of a discrete set of available values. However, as discrete variables increase the computational time, the cross- sectional design variables are generally assumed to be continuous.

The design variables and design parameters are together clearly define the structure. If the design variables are known in a given design point, this completely defines the geometry and other properties of the structure. In order to guarantee this, the chosen design variables must be independent to each other.

1.2.2. Optimization constraints

Some designs are useful solutions to the optimization problem, but others might be inadequate in terms of function, behaviour, or other considerations. If a design meets all the requirements placed on it, it will be called a feasible design. In most cases, the starting design is a feasible design. The restrictions that must be satisfied in order to produce a feasible design are called constraints.
From a physical point of view the constraints can be separated into two groups:

Constraints imposed on the design variables and which restrict their range for reasons other than behaviour considerations will be called design constraints or side constraints. The geometrical optimization constraints describes the lower the upper limit of the design variables. These are expressed in an explicit form of the design variables. These could be for example a minimum and maximum thickness of the plate.

Constraints that derive from behaviour requirements will be called behaviour constraints. Limitations on the maximum stresses, displacements, or buckling strength are typical examples

Of behaviour constraints. This type of constraints based on a result of a structural analysis. Explicit and implicit behaviour constraints are both encountered in practical design. Explicit behaviour constraints are often given by formulas presented in design codes or specifications.

From the mathematical point of view, in most cases, constraints may usually be expressed as a set of inequalities:

$$G_j x_i = 0 \quad (j=1,...,m \; ; \; i=1,...,n), \tag{1.1}$$

Where m is the number of inequality constraints and $x_i$is the vector of design variables. In a structural design problem, one has also to consider equality constraints of the general form:

$$h_j x_i = 0 \quad (j=m+1,...,p), \tag{1.2}$$

Where p-mis the number of equalities. In many cases equality constraints can be used to eliminate variables from the optimization process, thereby reducing their number.

The equality-type constraints can be used to reduce the number of design variables. This type of constraints may represent also various design considerations such as a desired ratio between the width of a cross section and its depth.

We may view each design variable as one dimension in a design space and any particular set of variables as a point in this space. In case of two design variables the design space reduces to a plan, but in the general case of n variables, we have an n-dimensional hyperspace. A design which satisfies all the constraints is a feasible design. The set of values of the design variables that satisfy the equation $g_j(x_i) = 0$ forms a surface in the design space. It is a surface in the sense that it cuts the space into two regions: where one where $g_j(x_i) > 0$ and the other $g_j(x_i)<0$. The design space and the constraint surfaces for the three-bar truss example are shown in Figure 1.2. The set of all feasible designs form the feasible region. Points on the surface are called constrained designs. The j^{th} constraint is said to be active in a design point for which $g_j (x_i) = 0$ and passive if $g_j (x_i) <0$. If $g_j(x_i) > 0$ the constraint is violated and the corresponding design is infeasible.

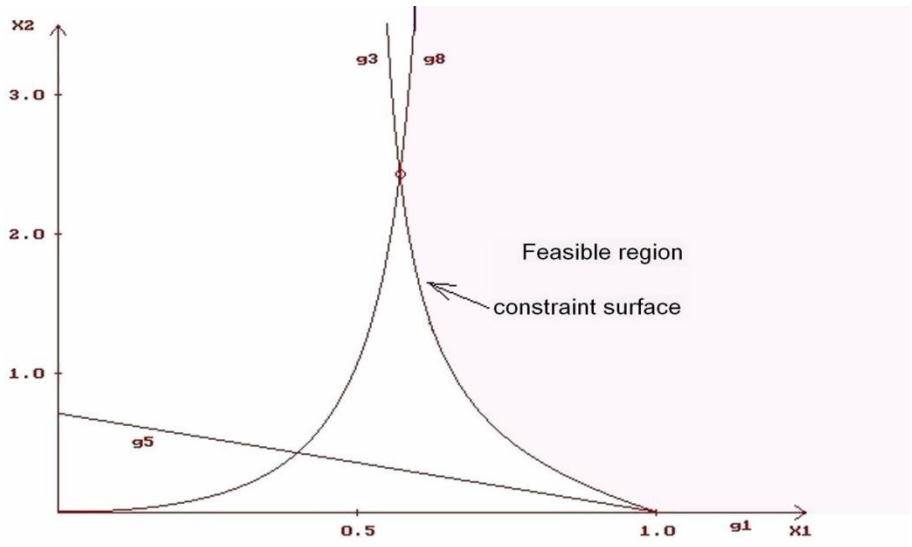


Figure 1.2.: Optimality constraints in the space of the design variables

1.2.3. The objective function

There usually exist an infinite number of feasible designs. In order to find the best one, it is necessary to form a function of the variables to use for comparison of feasible design alternatives. The objective function (or cost function) is the function whose least value is usually-wanted in an optimization procedure. It is a function of the design variables and it may represent the weight, the cost of the structure, or any other criterion by which some possible designs are preferred to others. We always assume that the objective function ($Z = F(x_i)$), is to be minimized, which entails no loss of generality since the minimum of $-F(x_i)$ occurs where the maximum of $F(x_i)$ takes place (see Figure 1.3). The selecting the objective function has got a significant impact on the entire optimization process. For example, if the cost of the structure is assumed to be proportional to its weight, then the objective function will represent the weight. The weight of the structure is often of critical importance, but the minimum weight is not always the cheapest. In general, the objective function represents the most important single property of a design, but it may represent also a weighted sum of a number of properties. A general cost function may include the cost of materials, fabrication, transportation, operation, repair, and many other cost factors. In this case, large numbers of members are considered in the form of the objective function, where it is appropriate to analyze the impact of certain members of the product price. Special attention should be paid to the components, which can result a "nearly constant" objective function. They are not worth to take into account. It is not true, that the most complex objective function gives the best results. In general, the objective function is a nonlinear function of the design variables.

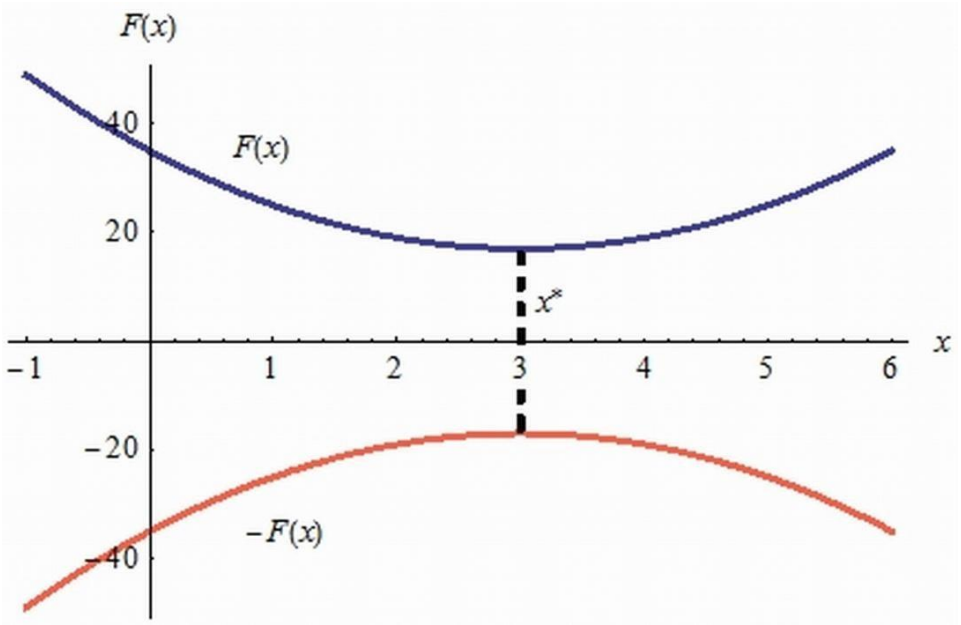


Figure 1.3.: The local extremum of the goal function

It is also possible to optimize simultaneously for multiple objective function, but this is only recommended if dominant objective function could not be selected, or the objective functions are in contradiction. This is the field of multi-objective optimization. The simplest solution technique is if we create a weighted sum of the objective functions and solve the problem as a standard optimization problem with only one objective function.

Pareto has developed the theory of the multi-objective optimization in 1896. It is important to note that the solution of an optimization problem with one goal function is generally a design point in the space of the design variables, while the solution of the Pareto-optimization problem is a set of design points, called Pareto front. A Pareto optimal solution is found if there is no other feasible solution that would reduce some objective function without causing a simul-

taneous increase in at least one other objective function.

We illustrated this complicated field with an example about „Optimization of Geometry for Car Bag‖ (this example was solved by company Sigma technology using the IOSO optimization software: www.iosotech.com), which can be found in the attached database for examples.

1.2.4. Formulating the optimization problem

The general formulation of the constrained optimization problem in the n dimensional Euclidian space is the following:

$$\begin{gathered} Z = F(x_i) \to \min \\ g_j(x_i) \le 0 \; i = 1,...,n \quad j = 1,...,m \\ h_j(x_i) = 0 \; j = m+1,...,p \,. \end{gathered} \tag{1.3}$$

This is an optimization problem with one goal function ($F(x_i)$), with inequality ($g(x_i)$) and equality($h(x_i)$) constraints is formulated for n design variable.

UNIT - II

SINGLE-VARIABLEOPTIMIZATION

Only single-variable functions are considered in this chapter. These methods will be used in later chapters for multi-variable function optimization.

2.1. Optimality Criteria

Sufficient conditions of optimality:
Suppose at point x*, the first derivative is zero and the first nonzero higher order derivative is denoted by n

(i) If n is odd, x* is a point of inflection
(ii) If n is even, x* is a local optimum
(a) If that derivative is positive, x* is a local minimum
(b) If that derivative is negative, x* is a local maximum.

2.2. Bracketing Algorithms

Finds a lower and an upper bound of the minimum point

2.2.1. Exhaustive Search

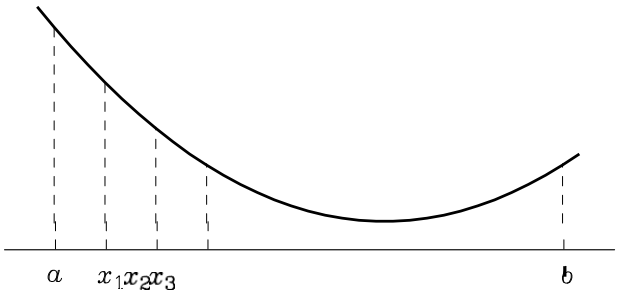


Figure 2.1 The exhaustive search method uses equally spaced points.

The exhaustive search method can be used to solve problems where the interval in which the optimum is known to lie is finite. Let xs and xf denote, respectively, the starting and final points of the interval of uncertainty.† The exhaustive search method consists of evaluating the objective function at a predetermined number of equally spaced points in the interval (xs , xf), and reducing the interval of uncertainty using the assumption of unimodality. Suppose that a function is defined on the interval (xs ,xf) and let it be evaluated at eight equally spaced interior points x1 to x8. Assuming that the function values appear as shown in Fig. 5.6, the minimum point must lie, according to the assumption of unimodality, between points x5 and x7. Thus the interval (x5, x7) can be considered as the final interval of uncertainty. In general, if the function is evaluated at n equally spaced points in the original interval of uncertainty of length L0 = xf− xs , and if the optimum value of the function(among the n function values) turns out to be at point xj , the final interval of uncertainty

Region-EliminationMethods

FIBONACCI METHOD

As stated earlier, the Fibonacci method can be used to find the minimum of a function of one variable even if the function is not continuous. This method, like many other elimination methods, has the following limitations:

1. The initial interval of uncertainty, in which the optimum lies, has to be known.

2. The function being optimized has to be unimodal in the initial interval of uncertainty.

3. The exact optimum cannot be located in this method. Only an interval known as the final interval of uncertainty will be known. The final interval of uncertainty can be made as small as desired by using more computations.

4. The number of function evaluations to be used in the search or the resolution required has to be specified before hand. This method makes use of the sequence of Fibonacci numbers, {Fn}, for placing the experiments. These numbers are defined as

F0 = F1 = 1

Fn= Fn−1 + Fn−2, n = 2, 3, 4, . . .

which yield the sequence 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,. . . .

Procedure. Let L0 be the initial interval of uncertainty defined by a ≤ x ≤ b and n be the total number of experiments to be conducted. DefineL∗2 =Fn−2FnL0 (5.5) and place the first two experiments at points x1 and x2, which are located at a distance of L∗2 from each end of L0.† This gives‡x1 = a + L∗2 = a +Fn−2Fn L0 x2 = b − L∗2 = b −Fn−2 FnL0 = a +Fn−1 FnL0

Discard part of the interval by using the unimodality assumption. Then there remains a smaller interval of uncertainty L2 given by§ L2 = L0 − L∗2 = L0_1 −Fn−2 Fn_=Fn−1 FnL0

Thus after conducting n − 1 experiments and discarding the appropriate interval in each step, the remaining interval will contain one experiment precisely at its middle point. However, the final experiment, namely, the nth experiment, is also to be placed at the center of the present interval of uncertainty. That is, the position of the nth experiment will be same as that of (n − 1)th one, and this is true for whatever value we choose for n. Since no new information can be gained by placing the nth experiment exactly at the same location as that of the (n − 1)th experiment, we place the nth experiment very close to the remaining valid experiment, as in the case of the dichotomous search method. This enables us to obtain the final interval of uncertainty to within Limits.

The Golden Section Search method is used to find the maximum or minimum of a unimodal function. (A unimodal function contains only one minimum or maximum on the interval [a,b]. To make the discussion of the method simpler, let us assume that we are trying to find the maximum of a function. The previously introduced Equal Interval Search method is somewhat inefficient because if the interval is a small number it can take a long time to find the maximum of a function. To improve this efficiency, the Golden Section Search method is suggested.

UNT -III
MULTI-VARIABLEOPTIMIZATION

Functions of multiple variables (N variables) are considered for minimization here. Duality principle can be used to apply these methods to maximization problems.

1.1. Optimality Criteria

A stationary point $\bar{x}$ is minimum, maximum, or saddle-point if $\nabla^2 f(x)$ is positive definite, negative definite, or $\nabla^2 f(x)$

Necessary Conditions: For x* to be a local minimum, $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive Semi definite.

Sufficient Conditions: $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite than, x* is an isolated local minimum of f (x) .

1.2. Direct Search Methods

Use only function values; no gradient information is used.

1.2.1. Simplex Search Method

- A simplex is a set of $(N-1)$ points. At each iteration, a new simplex is created from the current simplex by fixed transition rules.
- The worst point is projected a suitable distance through the centroid of the remaining points.

Algorithm:

Step1: Choose $L > 1$, $(0, 1)$, and a termination parameter β . Create an initialsimplex[1].

Step2: Find x_h (the worst point), x_l (the best point), and x_g (next to the worst point). Calculate x_c —

Step3: Calculate the reflected point $x_r = 2x_c - x_h$. Set $x_{new} = x_r$.

If $f(x_r) = f(x_l)$,set $x_{new} = (1+)x_c - x_h$ (expansion).

Else if $f(x_r) > f(x_h)$,set $x_{new} (<)x_c < x_h$ (contraction).

Else if $f(x_g) > f(x_r) > f(x_h)$, set $x_{new} > x_c < x_h$ (contraction).
Calculate $f(x_{new})$ and replace x_h by x_{new}. }

Else go to Step 2.

1.2.2. Powell's Conjugate Direction Method

- Most successful direct search method
- Uses history of iterations to create new search directions
- Based on a quadratic model
- Generate N conjugate directions and perform one-dimensional search in each direction one at a time

Parallel Subspace Property: Given a quadratic function q(x) , two arbitrary but distinct points $x^{(1)}$ and $x^{(2)}$, and a direction d. If $y^{(1)}$ is the solution to min q $(x^{(1)}$ d) and $y^{(2)}$ is the solution to min q $(x^{(2)}$ d) , then the direction $(y^{(2)} - y^{(1)})$ is C-conjugate to d or $(y^{(2)} - y^{(1)})^T C d$ Diagonal matrix.

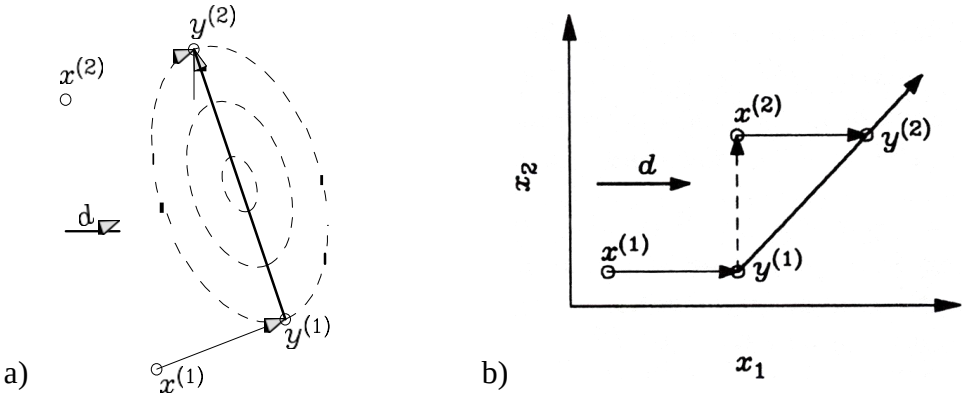


Figure 3.1 Illustration of the paralell subspace property with two arbitrary points and an arbitrary search direction in (a). The same can also be achieved from one point and two coordinate points in(b).

Instead of using two points and a direction vector to create one conjugate direction, one point and coordinate directions can be used to create conjugate directions (Figure3.1).

Extended Parallel Subspace Property: In higher dimensions, if from $x^{(1)}$ the point $y^{(1)}$ is found after searches along each of m(< n) conjugate directions, and similarly if from $x^{(2)}$ the point $y^{(2)}$ is found after searches along each of m conjugate directions, $s^{(1)}$, $s^{(2)}$,…, $s^{(m)}$, then the vector $(y^{(2)} - y^{(1)})$ will be the conjugate to all of the m previous directions.

Algorithm:

Step1: Choose a starting point $x^{(0)}$ and a set of N linearly independent directions; possibly $s^{(i)}$ $e^{(i)}$ for i=1, 2,…, N.

Step2: Minimize along N unidirectional search directions using the previous minimum point to begin the next search. Begin with the search $s^{(1)}$ direction and end with $s^{(N)}$. Thereafter, perform another unidirectional search along $s^{(1)}$.

Step3: Form a new conjugate direction d using the extended parallel subspace property.

Step4: If d is small or search directions are linearly independent, **Terminate.**

Else replace $s^{(j)}$ $s^{(j1)}$ for all j,N,N=1,...,2.Set $s^{(1)} = d / d$ and go to Step 2.

A test is required to ensure linear independence of conjugate directions. If the function is quadratic, exactly N loops through steps 2 to 4 are required. If the function is quadratic, exactly (N -1) loops through Steps 2 to 4 is required. Since in every iteration of the above algorithm exactly (N -1) unidirectional searches are necessary, a total of (N -1) □ (N -1)or (N^2-1) unidirectional searches are necessary to find N conjugate directions. Thereafter, one final unidirectional search is necessary to obtain the minimum point. Thus, in order to find the minimum of a quadratic objective function, the conjugate direction method requires a total of N^2 unidirectional searches.

Disadvantages:

- It takes usually more than N cycles for non quadratic functions
- One-dimensional searches may not be exact, so directions may not be conjugate
- May halt before the optima is reached

Consider the Himmelblau function:

Minimize

in the interval $0 = x_1, x_2 = 5$.

Step1: We begin with a point $x^{(0)}= (0, 4)^T$. We assume initial search directions as $s^{(1)}= (1, 0)^T$ and $s^{(2)}= (0,1)^T$.

Step2: We first find the minimum point along the search direction $s^{(1)}$. Any point along that direction can be written as $x^p= x^{(0)}= s^{(1)}$, where is a scalar quantity expressing the distance of the point x^p from $x^{(0)}$. Thus, the point x^p can be written as $x^p= (0, 4)^T$. Now the two-variable function $f(x_1,x_2)$ can be expressed in terms of one variable as

,

which represents the function value of any point along the direction $s^{(1)}$ and passing through $x^{(0)}$. Since we are looking for the point for which the function value is minimum, we may differentiate the above expression with respect to and equate to zero. But in any arbitrary problem, it may not be possible to write an explicit expression of the single-variable function $F(X)$ and differentiate. In those cases, the function $F(X)$ can be obtained by substituting each variable x_i by x^p. Thereafter, any single-variable optimization methods, as described in,can be used to find the minimum point. The first task is to bracket the minimum and the n the subsequent task is to find the minimum point. Here, we could have found the exact minimum solution by differentiating the single-variable function $F(X)$ with respect to and then equating the term to zero, but we follow the more generic procedure of numerical differentiation, a method which will be used in many real-world optimization problems. Using the bounding phase method in the above problem we find that the minimum is bracketed in the interval (1,4) and using the golden section search we obtain the minimum 2.083 with three decimal places of accuracy. Thus, $x^{(1)}$ $(2.083, 4.000)^T$.

Similarly, we find the minimum point along the second search $s^{(2)}$ from the point direction $x^{(1)}$. A general point on that line is 2.083..

The optimum point found using a combined application of the bounding phase and the golden section search method is 1,592 and the corresponding point is $x^{(2)}= (2.083, 2.408)^T$.

From the point $x^{(2)}$, we perform a final unidirectional search along the first search direction and obtain the minimum point $x^{(3)}= (2.881, 2.408)^T$.

Step3: According to the parellel subspace property, we find the new conjugate direction.

$$2.881, 2.408\ .083,4.000^T \quad 0.798$$

Step4: The magnitude of search vector d is not small.Thus, the new conjugate search directions are

.

This completes one iteration of Powell's conjugate direction method. Figure 3.2 shows the new conjugate direction on a contour plot of the objective function. With these new search directions we now proceed to Step2.

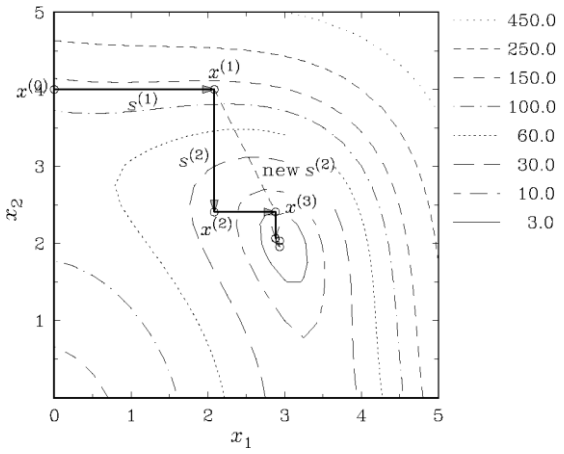


Figure 3.2 Two iterations of Powell's conjugate direction method.

Step2: A single- variable minimization along the search direction $s^{(1)}$ from the point $x^{(3)}= (2.881, 2.408)^T$ results in the new point $x^{(4)}= (3.063, 2.045)^T$. If the objective function had been a quadratic function, we would have achieved the optimum point at this step. However, it is interesting to note that the solution $x^{(4)}$ is close to the true minimum of the function. One more unidirectional search along $s^{(2)}$ from the point $x^{(4)}$ results in the point $x^{(5)}= (2.988,2.045)^T$. Another minimization along $s^{(1)}$ results in $x^{(6)} = (3.008,2.006)^T$ f $(x^{(6)} = 0.004)$.

Step3: The new conjugate direction is

The unit vector along this direction is $(0.816, 0.578)^T$.

Step4: The new pair of conjugate search directions are $s^{(1)}= (0.448, , 0.894)^T$ and $s^{(2)}=(0.055, 0.039)^T$ respectively. The search direction d (before normalizing) may be considered to be small and therefore the algorithm may terminate.

We observe that in one iteration of Step 2, $(N -1)$ unidirectional searches are necessary. Thus, computationally this method may be expensive. In terms of the storage requirement, the algorithm has to store (N -1) points and N search directions at any stage of the iteration.

1.3. Gradient-based Methods

- Direct search methods are expensive
- Gradient-based methods assume existence of $f(x)$, $f(x)$, and $^2f(x)$
- All methods employ

Where $x^{(k)}$ is the step length and $s(x^{(k)})$ is the search direction.

– Usually, $x^{(k)}$ is selected to minimize the function in $s(x^{(k)})$ direction. Update search direction iteratively.

Numerical Gradient Approximation

- In real-world engineering problems, gradients are often difficult to calculate.
- Even if they are available, their computation could be erro rprone
- Numerically gradients are computed:

Thecomputationofthefirstderivativewithrespecttoeachvariablerequirestwofunction evaluations , thustotaling2N function evaluations forth complete first derivative vector. The computation of the second derivative $x^2 f / x^2_i$ requires three function evaluations, but the second-order partial derivative $^2 f / (x_i, x_j)$ requires four function evaluations. Thus, the computation of Hessian matrix requires $(2N^2 1)$ function evaluations (assuming the symmetry of the matrix).

Cauchy's Method (Steepest Descent)

Greatest decrease in $s(x^{(k)}) f(x)$ direction.

Algorithm:

Step1: Choose a maximum number of iterations M to be performed, an initial point $x^{(0)}$, two termination parameters and set k = 0 .

Step2: Calculate $f(x^{(k)})$, the first derivative at the point $x^{(k)}$.

Step3: If $\| f(x^{(k)}) \| \ x_1$, **Terminate.**

Else if k M ,**Terminate.**
Else go to Step 4.

Step4: Perform a unidirectional search to find $x^{(k)}$ usingx_2 such that /($f(x^{(k-1)})$ $f(x^{(k)}$ $^{(k)}$ f $(x^{(k)}))$ is minimum. One criterion for termination is when $(x^{(k-1)})$ f $(x^{(k)})$ $_2$.

Step5: Is $\left\| \frac{x^{(k+1)} - x^{(k)}}{x^{(k)}} \right\|$ $_1$? If yes, **Terminate.**

Else set k , k-1 and go to Step 2.

Cauchy's method works well when $x^{(0)}$ is far away from x*.

1.3.1. Newton's Method

Use a search direction $s(x^{(k)}) >^2 f(x^{(k)})^1 > f(x^{(k)})$.

Algorithm: Same as Cauchy's method except step 4 is modified:

Step4: Perform a line search to find $^{(k)}$ using $_2$ such that $f(x^{(k-1)})$ $f(x^{(k)}$ $^{(k)}{}^2 f(x^{(k)})^1 f(x^{(k)}))$ is minimum.

Newton's method works well when $x^{(0)}$ is close to x*.

1.3.2. Marquardt's Method

– A compromise between Cauchy's and Newton's method.
– When the point is far away from the optimum, Cauchy's method is used and when the point is close to the optimum Newton‘s method is used.

Algorithm:

Step1: Choose a starting point, $x^{(0)}$, the maximum number of iterations, M, and a termination parameter, . Set k = 0 and $x^{(0)} = 10^4$ (a large number).

Step2: Calculate f $(x^{(k)})$.

Step3: If f $(x^{(k)})$ or k= M , **Terminate.**

Else go to Step 4

Step 4:

Set $x^{(k\ 1)}$? $x^{(k)} > s(x^{(k)})$.

Step5: Is $f(x^{(k-1)})$ $f(x^{(k)})$? If yes, go to Step 6.

Else go to Step7.

Step6: Set $^{(k-\ 1)}$ $\underline{1}^{(k)}$, k_2 ,k1, and go to Step2.

Step7: Set $^{(k)}$ $2^{(k)}$ and go to Step4.

Need to compute Hessian matrix, which may be cumbersome.

UNIT-IV

MULTI VARIABLE CONSTARINED OPTIMIZATION

In the previous section we optimized (i.e. found the absolute extrema) a function on a region that contained its boundary. Finding potential optimal points in the interior of the region isn't too bad in general, all that we needed to do was find the critical points and plug them into the function. However, as we saw in the examples finding potential optimal points on the boundary was often a fairly long and messy process.

In this section we are going to take a look at another way of optimizing a function subject to given constraint(s). The constraint(s) may be the equation(s) that describe the boundary of a region although in this section we won't concentrate on those types of problems since this method just requires a general constraint and doesn't really care where the constraint came from.

So, let's get things set up. We want to optimize (i.e. find the minimum and maximum value of) a function, f(x,y,z) subject to the constraint g(x,y,z)=k. Again, the constraint may be the equation that describes the boundary of a region or it may not be. The process is actually fairly simple, although the work can still be a little overwhelming at times.

1. Solve the following system of equations.$\nabla f(x,y,z)=\lambda\nabla g(x,y,z)g(x,y,z)=k\nabla f(x,y,z)=\lambda\nabla g(x,y,z)g(x,y,z)=k$
2. Plug in all solutions, (x,y,z) from the first step into f(x,y,z) and identify the minimum and maximum values, provided they exist and $\nabla g\neq\rightarrow 0\nabla g\neq 0\rightarrow$ at the point.

The constant λ, is called the **Lagrange Multiplier**.

Notice that the system of equations from the method actually has four equations, we just wrote the system in a simpler form. To see this let's take the first equation and put in the definition of the gradient vector to see what we get.

$\langle fx,fy,fz\rangle=\lambda\langle gx,gy,gz\rangle=\langle\lambda gx,\lambda gy,\lambda gz\rangle$

In order for these two vectors to be equal the individual components must also be equal. So, we actually have three equations here.

Fx=λgxfy
These three equations along with the constraint, g(x,y,z)=cg(x,y,z)=c, give four equations with four unknowns x, y, z, and λ. Note as well that if we only have functions of two variables then we won't have the third component of the gradient and so will only have three equations in three unknowns x, y, and λ. As a final note we also need to be careful with the fact that in some cases minimums and maximums won't exist even though the method will seem to imply that they do. In every problem we'll need to make sure that minimums and maximums will exist before we start the problem.

The Method of Lagrange Multipliers is a useful way to determine the minimum or maximum of a surface subject to a constraint. It is an alternative to the method of substitution and works particularly well for non-linear constraints. For the following examples, all surfaces will be denoted as f (x, y) and all constraints as g(x, y) = c . The process follows an algorithm of steps which reveal another relationship between the input variables that was neither given nor evident beforehand. We can use this relationship in conjunction with the constraint to determine critical points of the surface on the constraint.

The Method of Lagrange Multipliers follows these steps:

1) Given a multivariable function f (x, y) and a constraint g(x, y) = c , define the Lagrange function to be L(x, y) = f (x, y) − λ(g(x, y) − c), where λ (lambda) is multiplied (distributed) through the constraint portion.
2) Determine the partial derivatives Lx and Ly .
3) Set the partial derivatives Lx and Ly equal to zero.
4) Make note of any immediate solutions for x and y that satisfy = 0 Lx and Ly = 0 . Often there are none, so proceed to step 5.
5) Isolate the λ in each equation.
6) Equate the two λ equations, dropping out the λ altogether.
7) Reduce the equation from step 6 as far as possible. This is the relationship between x and y that shall be substituted into the constraint.
8) Substitute this equation into the constraint and algebraically solve for the variable that remains. This is where the critical points start to be determined.
9) Solve for the other variable by re-evaluating your results in step 8 back into the constraint.

If just one critical point results, you cannot necessarily determine if it is a minimum or maximum just by itself, so you must study the shape of the surface and make inferences based on its shape and the relative location of the constraint (this is where a contour map helps). If two or more points result, then minima and maxima are easily determined by comparing their z-values. Some points may be neither minima nor maxima and are then ignored.

Kuhn – Tucker conditions for unconstrained optimization with inequality constraints.

In mathematical optimization, the Karush–Kuhn–Tucker (KKT) conditions, also known as the Kuhn–Tucker conditions, are first derivative tests (sometimes called first-order) necessary conditions for a solution in nonlinear programming to be optimal, provided that some regularity conditions are satisfied.

Allowing inequality constraints, the KKT approach to nonlinear programming generalizes the method of Lagrange multipliers, which allows only equality constraints. Similar to the Lagrange approach, the constrained maximization (minimization) problem is rewritten as a Lagrange function whose optimal point is a saddle point, i.e. a global maximum (minimum) over the domain of the choice variables and a global minimum (maximum) over the multipliers, which is why the Karush–Kuhn–Tucker theorem is sometimes referred to as the saddle-point theorem

Suppose we have a function f, which we wish to maximise, together with some constraints, $g_i \leq c_i$, which must be satisfied. This leads us to the problem

Max f(**x**) subject $g_1(\mathbf{x}) \leq c_1 g_2(\mathbf{x}) \leq c_2$ $g_m(\mathbf{x}) \leq c_m$

$x_i \geq 0$

We have seen how to solve this problem with equality constraints; we introduce a number of multipliers $\lambda_1, \ldots, \lambda_m$ and form the Lagrangian which we partially differentiate with respect to each x_i and each λ_j and set equal to 0, to get a system of m + n equations with n m variables.

For the inequality case, we do a similar thing. First we form the Lagrangian, L:

$$L = f(\mathbf{x}) + \lambda_1(c_1 - g_1(\mathbf{x})) + \cdots + \lambda_m(c_m - g_m(\mathbf{x}))$$

Result1 The Kuhn-Tucker conditions, which are **necessary** (but not sufficient) for **a point** to be a maximum are:

$$\frac{\partial L}{\partial x_i} \leq 0 \qquad x_i \geq 0 \qquad x_i \frac{\partial L}{\partial x_i} = 0 \quad \text{for all } i=1...n$$

$$g_j(\mathbf{x}) \leq c_j \qquad \lambda_j \geq 0 \qquad \lambda_j(c - g_j(\mathbf{x})) = 0 \text{ for all } j=1...m$$

So, for each variable x_i we have 3 conditions which must be met; similarly for each constraint (and hence λ), we have 3 conditions which must be met. Each point that is a solution to this equation system is a possible candidate for the maximum.

Once we have established all the points, we need to check them individually to see which is the real maximum.

The conditions are called the **complementary slackness** conditions. This is because for each set of three conditions, either the first or these condition can be slack (i.e. not equal to zero),but the third condition ensures that they cannot both be non-zero.

Notes: This is a maximum only problem. To do a minimisation, you need to maximise the function $-f(\mathbf{x})$. Secondly, notation in books varies, so some state the constrant conditions as $g_j(\mathbf{x}) \geq c_j$, in which case the signs of some of the terms in the Lagrangian are altered.

Example 1 max $f(x_1, x_2) = 4x_1 + 3x_2$ subject to $g(x_1, x_2) = 2x_1 + x_2 \leq 10$ and $x_1, x_2 \geq 0$. We first form the Lagrangian:

$$L = 4x_1 + 3x_2 + \lambda(10 - 2x_1 - x_2)$$

Hence, the necessary conditions for a point to be a maximum are:

$L_{x_1} = 4 - 2\lambda \leq 0$	$x_1 \geq 0$	$x_1(4 - 2\lambda) = 0$
$L_{x_2} = 3 - \lambda \leq 0$	$x_2 \geq 0$	$x_2(3 - \lambda) = 0$
$2x_1 + x_2 - 10 \leq 0$	$\lambda \geq 0$	$\lambda(10 - 2x_1 - x_2) = 0$

We solve this set of inequalities and equations to find points which may be maxima. Let 1 (ii) denote the second condition on the first line etc.

$$1(iii): \quad x_1(4 - 2\lambda) = 0 \Rightarrow x_1 = 0 \text{ or } \lambda = 2$$

Suppose $\lambda = 2$. Then:

$$2(i): \quad 3 - \lambda \leq 0 \Rightarrow 1 \leq 0$$

which is clearly false. Hence, we must have that $x_1 = 0$. Then:

$$2(iii): \quad x_2(3-\lambda) = 0 \Rightarrow x_2 = 0 \text{ or } \lambda = 3$$

We can see that, if x_2=0 (to get x_1=0):

$$3(iii): \quad 10\lambda = 0 \Rightarrow \lambda = 0$$

But this contradicts 1 (i). So, we have $x_1 = 0$ and $\lambda = 3$.Hence:

$$3(iii): \quad 3(10 - x_2) = 0 \Rightarrow x_2 = 10$$

So, we have solved the system and found the only solution, which is $x_1 = 0$, $x_2 = 10$, $\lambda = 3$ and hence $f(x_1, x_2) = 30$. As before with equality constraints, λ measures the increase in $f(x_1, x_2)$ for a one unit increase in the constraint(i.e.increasingc_1=10by1).

In general,solving such systems can be very tedious (although not difficult) if there are more than 4 variables and constraints in total. What you need to do is just plug away at every possible combi- nation and eliminate those that do not fitth e conditions.

f(**x**) to find whether they are the maximum.

For certain types of systems, we can assume the point(s) we find are maxima. This occurs if the function and the functions g_i satisfy some more conditions, known as the Kuhn-Tucker sufficiency conditions to check them individually to see which is the real maximum.

The conditions are called the **complementary slackness** conditions. This is because for each set of three conditions, either the first or these condition can be slack (i.e.not equal to zero), but the third condition ensures that they cannot both be non-zero.

Notes: This is a maximum only problem. To do a minimisation, you need to maximise the function $-f(\mathbf{x})$. Secondly, notation in books varies, so some state the constrant conditions as $g_j(\mathbf{x}) \geq c_j$, in which case the signs of some of the terms in the Lagrangian are altered.

Example 1 max $f(x_1, x_2) = 4x_1 + 3x_2$ subject to $g(x_1, x_2) = 2x_1 + x_2 \leq 10$ and $x_1, x_2 \geq 0$. We first form the Lagrangian:

$$L=4x_1+3x_2+\lambda(10-2x_1-x_2)$$

Hence, the necessary conditions for a point to be maximum are:

$L_{x1} = 4 - 2\lambda \leq 0$	$x_1 \geq 0$	$x_1(4 - 2\lambda) = 0$
$L_{x2} = 3 - \lambda \leq 0$	$x_2 \geq 0$	$x_2(3 - \lambda) = 0$
$2x_1+x_2-10 \leq 0$	$\lambda \geq 0$	$\lambda(10-2x_1-x_2) = 0$

We solve this set of inequalities and equations to find points which may be maxima. Let 1 (ii) denote the second condition on the first line etc.

$$1(iii): \quad x_1(4 - 2\lambda) = 0 \Rightarrow x_1 = 0 \text{ or } \lambda = 2$$

Suppose $\lambda = 2$. Then:

2(i): $3 - \lambda \leq 0 \Rightarrow 1 \leq 0$

which is clearly false. Hence, we must have that $x_1 = 0$. Then:

$$\text{(iii): } x_2(3 - \lambda) = 0 \Rightarrow x_2 = 0 \text{ or } \lambda = 3$$

We can see that, if $x_2=0$ (to get $x_1=0$):

$$\text{(iii): } 10\lambda = 0 \Rightarrow \lambda = 0$$

But this contradicts 1 (i). So, we have $x_1 = 0$ and $\lambda = 3$.

Hence:

$$\text{iii): } 3(10 - x_2) = 0 \Rightarrow x_2 = 10$$

So, we have solved the system and found the only solution, which is $x_1 = 0$, $x_2 = 10$, $\lambda = 3$ and hence $f(x_1, x_2) = 30$. As before with equality constraints, λ measures the increase in $f(x_1, x_2)$ for a one unit increase in the constraint (i.e. increasing c_1=10by1).

In general, solving such systems can be very tedious (although not difficult) if the are more than 4 variables and constraints into. What you need to do is just plug away at every possible combination and eliminate those that do not fitth conditions. When you have eliminated all impossible points, you will be left with a few candidate points which you must then check by substitution into $f(\mathbf{x})$ to find whether they are the maximum.

For certain types of systems, we can assume the point(s) we find are maxima. This occurs if the function f and the functions g_i satisfy some more conditions, known as the Kuhn-Tucker sufficiency conditions.

Result 2 If the following conditions are satisfied:

1. f ($\mathbf{x}$) is differentiable and concave in the non negative or than t
2. each constraint function $g_i(\mathbf{x})$ is differentiable and convex in the non negative or than
3. a point $\mathbf{x}_0$ satisfies the Kuhn-Tucker conditions

then the point $\mathbf{x}_0$ is a maximum. [The nonnegative or than t is the region where each $x_i \geq 0$]

Example2 Suppose $f(x,y)=2x+3y$ and $g(x,y)=x^2+y^2 \leq 2$. Show that f and g satisfy the Kuhn-Tucker sufficiency conditions and hence find the maxima of f(x, y).

Well, all linear functions are both convex and concave, so f is certainly concave, and is clearly differentiable. As for g(x, y): we can show g is convex by showing the differential d^2g is positive definite. The Hessian matrix associated with g is:

So clearly $|H_1|$, $|H_2|>0$, so g is convex. Hence, any candidate points we find will automatically be maxima. As before, we form the Lagrangian:

$$L = 2x + 3y + \lambda(2 - x^2 - y^2)$$

The Kuhn-Tucker conditions imply that:

$$\begin{array}{llll} (1) & L_x = 2 - 2\lambda x \leq 0 & x \geq 0 & x(2 - 2\lambda x) = 0 \\ (2) & L_y = 3 - 2\lambda y \leq 0 & y \geq 0 & y(3 - 2\lambda y) = 0 \quad (3) \\ & x^2 + y^2 \leq 2 & \lambda \geq 0 & \lambda(2 - x^2 - y^2) = 0 \end{array}$$

(iii) implies that either $x=0$ or $\lambda x=1$. If $x=0$, then 1 (i) cannot be satisfied, hence $\lambda x=1$. Similarly, $\lambda y = \frac{2}{3}$ from (iii) (and clearly x, y, $\lambda > 0$). Substituting these values for x and y in (iii), and noting that $\lambda > 0$,we ge

As $\lambda > 0$, we must have $\lambda = 1$ hence and $x = 1$ and we have a maximum at this point

UNIT-V

GEOMETRIC AND INTEGER PROGRAMMING

Geometric-Programming

Geometric programming origin at edin 1961 with Zener's discovery (1,2,3,4,5) of an inge- nious method for designing equipment at minimum total cost– a method that is applicable when the component capital costs and operating costs can be expressed in terms of the design variables via a certain type of generalized polynomial [one whose exponents need not be positive integers].

Unlike competing analytical methods, which require the solution of a system of non linear equations derived from the differential calculus, this method requires the solution of a system of linear equations derived from both the differential calculus and certain in genious transformations. Unlike competing numerical methods, which minimize the total cost by either "direct search" or steepest the irnumerous descendants], this method provides formulas that show how the minimum total cost and associated optimal design depend on the design parameters [such as the unit material costs and power costs, which are determined externally and hence can not be set by thedesigner].

In1962, Duffin(6,7) significantly enlarged the class of generalized polynomials that can be minimized with this method – by introducing an ingenious analogue of the "dual vari-ational principles" that characterize the "network duality" originating from the two "Kir- choff laws" and "Ohm's laws" (originally discovered in the context of electrical networks). In 1964, Duffin and Peterson (8, 9) extended this "geometric programming duality" and associated methodology to the minimization of generalized polynomials subject to in equality constraints on other generalized polynomials.In essence, that development provided a non linear generalization of "linear programming duality"– one that is frequently applicable to the optimal design of sophisticated equipment and complicated systems [such as motors, transformers, generators, heat exchangers, power plants and their associated systems].

In 1967, Duffin, Peterson and Zener (10) published the first book on geometric pro- gramming

design. In 1971, Zener (11) published a short introductory book to make geometric pro- gramming more accessible to design engineers. More recent developments and publications are discussed in later sections.

An elementary example: The optimal design of a power line

Suppose the capital cost is simply proportional to at the volume of the line, namely the product of its desired length L [a design parameter] and its cross-sectional area t [an independent design variable or decision variable]. In particular then, the capital cost is CL where C [a design parameter] is the cost per unit volume of the material making up the line. Also, suppose the operating cost is simply proportional to the power loss, which is known to be proportional to both L and the line resistivity R [a design parameter] as well as to the square of the carried current I [a design parameter] while being inversely proportional to In particular then, the operating cost is where the proportionality constant D[a design parameter] is determined from the predicted life time of the line as well as the present and future unit power costs [via standard accounting procedures for expressing the sum of all such costs as a "present value" determined by interest rates]. In summary, the problem is to find the cross-sectional area $t > 0$ that minimizes the total cost

$$P(t) = c_1 t^1 + c_2 t^{-1} \quad \text{for given coefficients } c_1 = CL \text{ and } c_2 = DLRI^2. \qquad (1)$$

Such an optimal cross-sectional area t^* exists, because the positivity of the coefficients c_1 and c_2 clearly implies that, for $t>0$, the continuous function $P(t) > 0$ and $P(t) \to +\infty$ as either $t \to 0^+$ or $t \to +\infty$.

Generalized polynomials

The "objective function" P (t) defined by equation (1) is an example of a "generalized

polynomial "P(t)= T_i–namely, a sum of terms $T_i = c_i t$ each of which is a given coefficient c_i [usually determined by design parameters] multiplied into a product t_j of the independen t design variables t_j raised to appropriate powers a_{ij}, termed exponents. In the "single-variable" generalized polynomial (1),

the independent design variable t is the single scalar variable t_1 while the exponents $a_{11} = 1$ and $a_{21} = -1$. In the "multi-variable" generalized polynomial

the independent design variable t is the vector variable (t_1, t_2) while the exponents $a_{11} = -1$, $a_{12} = 2$, $a_{21} = -1/2$ and $a_{22} = -3$. Since non-integer exponents a_{ij} are mathematically permissible and are, in fact, needed in many applications, the "natural domain" of a generalized polynomial P (t) is normally $t > 0$ [meaning that each component t_j of t is positive]

so that $t^{-1/2}$, for example, is defined and real-valued.

either because of the relevant geometric, physical or economic laws, or because the log of

$$m_{a_{ij}} m$$

a posymonomial $T_i = c_i t_j$ is a linear function $\ln c_i + \sum_j a_{ij} \ln t_j$ of the logs, $\ln t_j$, of its independent design variables t_j– which makes it relatively easy to use in analytically

approximating empirically determined relations (via "linear regression analysis"). Consequently, it is not surprising that many realistic optimization problems can be modelled accurately with generalized polynomials of one type or another.

Traditional calculus and numerical approaches

The differential-calculus approach to minimizing our power-line example P (t), given by equation (1), is to solve the "optimality condition" $dP/dt = 0$ for t; that is, solve

$$c_1 - c_2 t^{-2} = 0. \qquad (2)$$

The solution to this nonlinear equation, easily accomplished analytically in this simple case, gives the optimal design [or" optimal solution"]

—

which in turn provides the minimum total cost [or "minimum value" or "optimal value"]

$$P^* = P(t^*) = (c_1c_2)^{1/2} + (c_1c_2)^{1/2} = 2(c_1c_2)^{1/2} = 2(CLDLRI^2)^{1/2} = 2LI(CDR)^{1/2}.$$

However, more complicated "posynomial minimization problems" [with more terms T_i and/or more independent variables t_j] usually can not be solved analytically by solving the appropriate optimality condition – namely, $dP/dt = 0$ in the single-variable case, or its multi-variable version $\nabla P = 0$. Prior to the creation of geometric programming, such minimization problem had to be solved numerically, either via a type of Newton-Raphson method applied to $dP/dt=0$ [or $\nabla P=0$], or via a direct-search or descent method applied directly to P (t). Since all such numerical methods require specific values for the posynomial coefficients c_i, they provide only a specific optimal solution t^* and optimal value P^*, which are optimal only for the specific coefficient values c_i and hence a very limited range of design- parameter values.

INTEGER PROGRAMMING

The linear -programming models that have been discussed thus for all have been continuous, in the sense that decision variables are allowed to be fractional. Often this is a realistic assumption. For instance, we might easily produce $102\frac{3}{4}$ gallons of a divisible good such as wine. It also might be reasonable to accept a solution giving an hourly production of automobiles at $58\frac{1}{2}$ if the model were based upon a verage hourly production, and the production had the interpretation of production rates.

Integer programming models a ris e in practically every area of application of mathematical programming. To develop a preliminary appreciation for the importance of these models, we introduce, in this section, three areas where integer programming has played an important role in supporting managerial decisions. We do not provide the most intricate available formulations in each case, but rather give basic models and suggest possible extensions.

Capital Budgeting In a typical capital-budgeting problem, decisions involve the selection of a number of potential investments. The investment decisions might be to choose among possible plant locations, to select a configuration of capital equipment, or to settle upon a set of research-and-development projects. Often it makes no sense to consider partial investments in these activities, and so the problem becomes a go–no-go integer program, where the decision variables are taken to be x j 0 or 1, indicating that the j th investment is rejected or accepted. Assuming that c j is the contribution resulting from the j th investment and that ai j is the amount of resource i , such as cash

or manpower, used on the j th investment, we can state the problem formally as

The objective function incorporates transportation and variable warehousing costs, in addition to fixed costs for operating warehouses. The constraints (2) indicate that each customer's demand must be met. The summation over the shipment variables x_{ij} in the ith constraint of (3) is the amount of the good shipped from warehouse i. When the warehouse is not opened, $y_i 0$ and the constraint specifies that nothing can be shipped from the warehouse. On the other hand, when the warehouse is opened and $y_i 1$, the constraint simply states that the amount to be shipped from warehouse can be no larger than the total demand, which is always true. Consequently, constraints (3) imply restriction (ii) as proposed above.

Although over simplified, this model forms the core for sophisticated and realistic distribution models incorporating such features as:

1. multi-echelon distribution systems from plant to warehouse to customer;
2. capacity constraints on both plant production and warehouse through put;
3. economies of scale in transportation and operating costs;
4. service considerations such as maximum distribution time from warehouses to customers;
5. multiple products; or
6. conditions preventing splitting of orders (in the model above, the demand for any customer can be supplied from several warehouses).

These features can be included in the model by changing it in several ways. For example, warehouse capacities are incorporated by replacing the term involving y_i in constraint (3) with $y_i K_i$, where K_i is the through put capacity of ware house i; multi-echel on distribution may require triple- subscripted variables x_{ijk} denoting the amount to be shipped, from plant it ustomer k through warehouse j . Further examples of how the simple warehousing model described here can be modified to incorporate the remaining features mentioned in this list are given in the exercises at the end of the chapter.

9.1 FORMULATING INTEGERPROGRAMS

The illustrations in the previous section not only have indicated specific integer-programming applications, but also have suggested how integer variables can be used to provide broad modeling capabilities beyond those available in linear programming. In many applications, integrality restrictions reflect natural indivisibilities of the problem under study. For example, when deciding how many nuclear aircraft carriers to have in th U.S. Navy, fractional solutions clearly are meaningless, since the optimal number is on the order of one or two. In these situations, the decision variables are inherently integral by the nature of the decision-making problem.

This is not necessarily the case in every integer-programming application, as illustrated by the capital- budgeting and the warehouse-location models from the last section. In these models, integer variables arise from (i) logical conditions, such as if a new product is developed, then a new plant must be constructed, and from (ii) non-linearities such as fixed costs for opening a warehouse. Considerations of this nature are so important for modeling that we devote this section to analyzing and consolidating specific integer- programming formulation techniques, which can be used as tools for a broad range of applications. When formulating LP's we often found that, strictly, certain variables should have been regarded as taking integer values but, for the sake of convenience, we let them take fractional values reasoning that the variables were likely to be so large that any fractional part could be neglected. Whilst this is acceptable in some situations, in many cases it is not, and in such cases we must find a numeric solution in which the variables take integer values.

Problems in which this is the case are called integer programs (IP's) and the subject of solving such programs is called integer programming (also referred to by the initials IP).

IP's occur frequently because many decisions are essentially discrete (such as yes/no, go/no-go) in that one (or more) options must be chosen from a finite set of alternatives.

Note here that problems in which some variables can take only integer values and some variables can take fractional values are called mixed-integer programs (MIP's).

As for formulating LP's the key to formulating IP's is practice. Although there are a number of standard "tricks" available to cope with situations that often arise in formulating IP's it is probably true to say that formulating IP's is a much harder task than formulating LP's.

We consider an example integer program below.

Capital budgeting

There are four possible projects, which each run for 3 years and have the following characteristics.

		Capital requirements (£m)		
Project	Return (£m)	Year 1	2	3
1	0.2	0.5	0.3	0.2
2	0.3	1.0	0.8	0.2
3	0.5	1.5	1.5	0.3
4	0.1	0.1	0.4	0.1
Available capital (£m)		3.1	2.5	0.4

We have a **decision problem** here: **Which projects would you choose in order to maximise the total return?**

Capital budgeting solution

We follow the same approach as we used for formulating LP's - namely:

- variables
- constraints
- objective.

We do this below and note here that the only significant change in formulating IP's as opposed to formulating LP's is in the definition of the variables.

Variables

Here we are trying to decide whether to undertake a project or not (a "go/no-go" decision). One "trick" in formulating IP's is to introduce variables which take the integer values 0 or 1 and represent binary decisions (e.g. do a project or not do a project) with typically:

- the positive decision (do something) being represented by the value 1; and
- the negative decision (do nothing) being represented by the value 0.

Such variables are often called zero-one or binary variables

To define the variables we use the verbal description of

x_j = 1 if we decide to do project j (j=1,...,4)
= 0 otherwise, i.e. not do project j (j=1,...,4)

Note here that, by definition, the x_j are integer variables which must take one of two possible values (zero or one).

Constraints

The constraints relating to the availability of capital funds each year are

$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 <= 3.1$ (year 1)
$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 <= 2.5$ (year 2)
$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 <= 0.4$ (year 3)

Objective

To maximise the total return - hence we have

maximise $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

This gives us the complete IP which we write as

maximise $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

subject to

$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 <= 3.1$
$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 <= 2.5$
$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 <= 0.4$
x_j = 0 or 1 j=1,...,4

Note:

- in writing down the complete IP we include the information that x_j = 0 or 1 (j=1,...,4) as a reminder that the variables are integers
- you see the usefulness of defining the variables to take zero/one values - e.g. in the objective the term $0.2x_1$ is zero if x_1=0 (as we want since no return from project 1 if we do not do it) and 0.2 if x_1=1 (again as we want since get a return of 0.2 if we do project 1). Hence effectively the zero-one nature of the decision variable means that we always capture in the single term $0.2x_1$ what happens both when we do the project and when we do not do the project.
- you will note that the objective and constraints are linear (i.e. any term in the constraints/objective is either a constant or a constant multiplied by an unknown). In this course we deal only with linear integer programs (IP's with a linear objective and linear constraints). It is plain though that there do exist non-linear integer programs - these are, however, outside the scope of this course.
- whereas before in formulating LP's if we had integer variables we assumed that we could ignore any fractional parts it is clear that we cannot do so in this problem e.g. what would be the physical meaning of a numeric solution with x_1=0.4975 for example?

Extensions to this basic problem include:

- projects of different lengths
- projects with different start/end dates
- adding capital inflows from completed projects
- projects with staged returns
- carrying unused capital forward from year to year
- mutually exclusive projects (can have one or the other but not both)
- projects with a time window for the start time.

How to amend our basic IP to deal with such extensions is given here.

In fact note here that integer programming/quantitative modelling techniques are increasingly being used for financial problems.

Solving IP's

For solving LP's we have general purpose (independent of the LP being solved) and computationally effective (able to solve large LP's) algorithms (simplex or interior point).

For solving IP's no similar general purpose and computationally effective algorithms exist.

Indeed theory suggests that no general purpose computationally effective algorithms will ever be found. This area is known as computational complexity and concerns NP-completeness. It was developed from the early 1970's onward and basically is a theory concerning "how long it takes algorithms to run". This means that IP's are a lot harder to solve than LP's.

Solution methods for IP's can be categorised as:

- general purpose (will solve any IP) but potentially computationally ineffective (will only solve relatively small problems); or
- special purpose (designed for one particular type of IP problem) but potentially computationally more effective.

Solution methods for IP's can also be categorised as:

- optimal
- heuristic

An optimal algorithm is one which (mathematically) guarantees to find the optimal solution.

It may be that we are not interested in the optimal solution:

- because the size of problem that we want to solve is beyond the computational limit of known optimal algorithms within the computer time we have available; or
- we could solve optimally but feel that this is not worth the effort (time, money, etc) we would expend in finding the optimal solution.

In such cases we can use a heuristic algorithm - that is an algorithm that should hopefully find a feasible solution which, in objective function terms, is close to the optimal solution. In fact it is often the case that a well-designed heuristic algorithm can give good quality (near-optimal) results.

For example a heuristic for our capital budgeting problem would be:

- consider each project in turn
- decide to do the project if this is feasible in the light of previous decisions

Applying this heuristic we would choose to do just project 1 and project 2, giving a total return of 0.5, which may (or may not) be the optimal solution.

Hence we have four categories that we potentially need to consider:

- general purpose, optimal
- general purpose, heuristic
- special purpose, optimal
- special purpose, heuristic.

Note here that the methods presented below are suitable for solving both IP's (all variables integer) and MIP's (mixed-integer programs - some variables integer, some variables allowed to take fractional values).

General purpose optimal solution algorithms

We shall deal with just two general purpose (able to deal with any IP) optimal solution algorithms for IP's:

- enumeration (sometimes called complete enumeration)
- branch and bound (tree search).

We consider each of these in turn below. Note here that there does exist another general purpose solution algorithm based upon cutting planes but this is beyond the scope of this course.

Enumeration

Unlike LP (where variables took continuous values (>=0)) in IP's (where all variables are integers) each variable can only take a finite number of discrete (integer) values.

Hence the obvious solution approach is simply to enumerate all these possibilities - calculating the value of the objective function at each one and choosing the (feasible) one with the optimal value.

For example for the capital budgeting problem considered above there are 2^4=16 possible solutions. These are:

x_1 x_2 x_3 x_4
0 0 0 0 do no projects

0 0 0 1 do one project
0 0 1 0
0 1 0 0
1 0 0 0

0 0 1 1 do two projects

0 1 0 1
1 0 0 1
0 1 1 0
1 0 1 0
1 1 0 0

1 1 1 0 do three projects
1 1 0 1
1 0 1 1
0 1 1 1

1 1 1 1 do four projects

Hence for our example we merely have to examine 16 possibilities before we know precisely what the best possible solution is. This example illustrates a general truth about integer programming:

What makes solving the problem easy when it is small is precisely what makes it become very hard very quickly as the problem size increases

This is simply illustrated: suppose we have 100 integer variables each with 2 possible integer values then there are 2x2x2x ... x2 = 2^{100} (approximately 10^{30}) possibilities which we have to enumerate (obviously many of these possibilities will be infeasible, but until we generate one we cannot check it against the constraints to see if it is feasible or not).

This number is plainly too many for this approach to solving IP's to be computationally practicable. To see this consider the fact that the universe is around 10^{10} years old so we would need to have considered 10^{20} possibilities per year, approximately $4x10^{12}$ possibilities per second, to have solved such a problem by now if we started at the beginning of the universe.

Be clear here - **conceptually** there is not a problem - simply enumerate all possibilities and choose the best one. But **computationally** (numerically) this is just impossible.

IP nowadays is often called "combinatorial optimisation" indicating that we are dealing with optimisation problems with an extremely large (combinatorial) increase in the number of possible solutions as the problem size increases.

Branch and bound (tree search)

The most effective general purpose optimal algorithm is LP-based tree search (tree search also being called branch and bound). This is a way of systematically enumerating feasible solutions such that the optimal integer solution is found.

Where this method differs from the enumeration method is that not all the feasible solutions are enumerated but only a fraction (hopefully a small fraction) of them. However we can still guarantee that we will find the optimal integer solution. The method was first put forward in the early 1960's by Land and Doig.

Consider our example capital budgeting problem. What made this problem difficult was the fact that the variables were restricted to be integers (zero or one). If the variables had been allowed to be fractional (takes all values between zero and one for example) then we would have had an LP which we could easily solve. Suppose that we were to solve this LP relaxation of the problem [replace $x_j = 0$

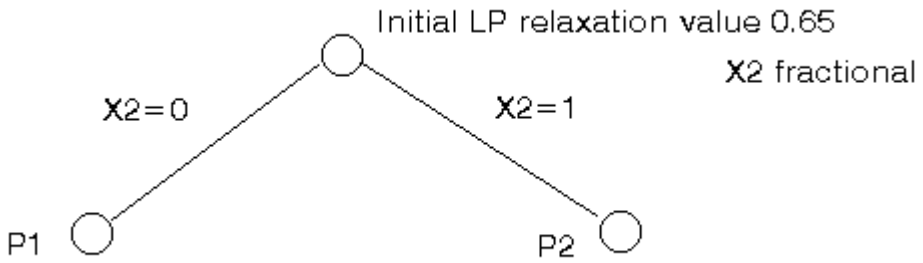
or 1 j=1,...,4 by $0 <= x_j <= 1$ j=1,...,4]. Then using the package we get $x_2=0.5$, $x_3=1$, $x_1=x_4=0$ of value 0.65 (i.e. the objective function value of the optimal linear programming solution is 0.65).

As a result of this we now know something about the optimal integer solution, namely that it is <= 0.65, i.e. this value of 0.65 is a (upper) bound on the optimal integer solution. This is because when we relax the integrality constraint we (as we are maximising) end up with a solution value at least that of the optimal integer solution (and maybe better).

Consider this LP relaxation solution. We have a variable x_2 which is fractional when we need it to be integer. **How can we rid ourselves of this troublesome fractional value?** To remove this troublesome fractional value we can generate two new problems:

- original LP relaxation plus $x_2=0$
- original LP relaxation plus $x_2=1$

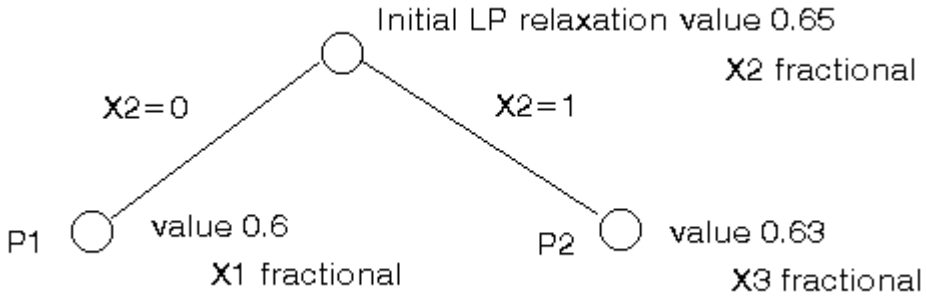
then we will claim that the optimal integer solution to the original problem is contained in one of these two new problems. This process of taking a fractional variable (a variable which takes a fractional value in the LP relaxation) and explicitly constraining it to each of its integer values is known as **branching**. It can be represented diagrammatically as below (in a tree diagram, which is how the name **tree search** arises).



We now have two new LP relaxations to solve. If we do this we get:

- P1 - original LP relaxation plus $x_2=0$, solution $x_1=0.5$, $x_3=1$, $x_2=x_4=0$ of value 0.6
- P2 - original LP relaxation plus $x_2=1$, solution $x_2=1$, $x_3=0.67$, $x_1=x_4=0$ of value 0.63

This can be represented diagrammatically as below.

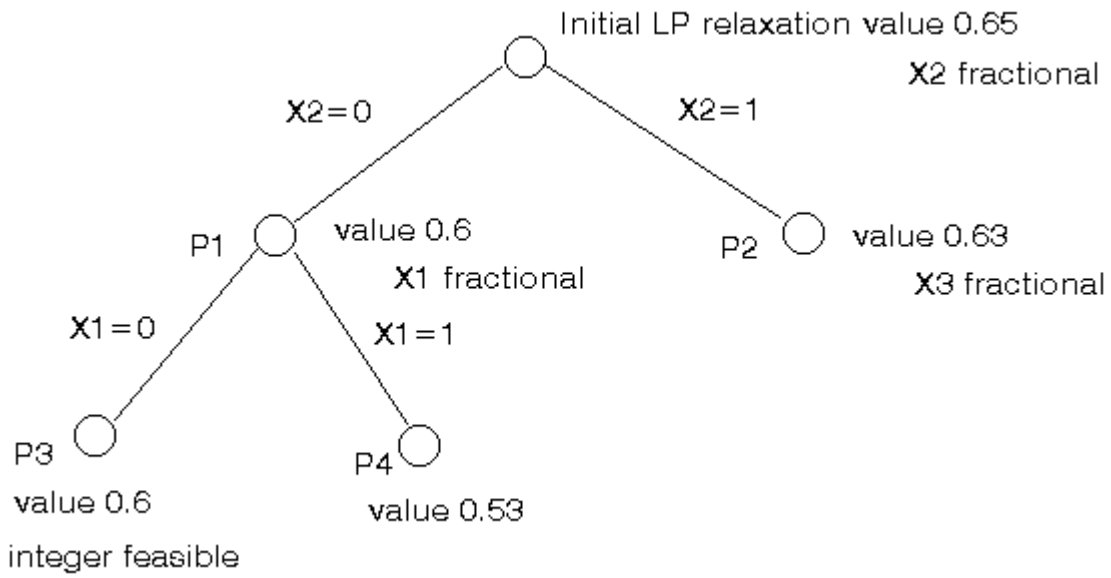


To find the optimal integer solution we just repeat the process, choosing one of these two problems, choosing one fractional variable and generating two new problems to solve.

Choosing problem P1 we branch on x_1 to get our list of LP relaxations as:

- P3 - original LP relaxation plus $x_2=0$ (P1) plus $x_1=0$, solution $x_3=x_4=1$, $x_1=x_2=0$ of value 0.6
- P4 - original LP relaxation plus $x_2=0$ (P1) plus $x_1=1$, solution $x_1=1$, $x_3=0.67$, $x_2=x_4=0$ of value 0.53
- P2 - original LP relaxation plus $x_2=1$, solution $x_2=1$, $x_3=0.67$, $x_1=x_4=0$ of value 0.63

This can again be represented diagrammatically as below.



At this stage we have identified a integer feasible solution of value 0.6 at P3. There are no fractional variables so no branching is necessary and P3 can be dropped from our list of LP relaxations.

Hence we now have new information about our optimal (best) integer solution, namely that it lies between 0.6 and 0.65 (inclusive).

Consider P4, it has value 0.53 and has a fractional variable (x_3). However if we were to branch on x_3 any objective function solution values we get after branching can never be better (higher) than 0.53. As we already have an integer feasible solution of value 0.6 P4 can be dropped from our list of LP relaxations since branching from it could never find an improved feasible solution. This is known as **bounding** - using a known feasible solution to identify that some relaxations are not of any interest and can be discarded.

Hence we are just left with:

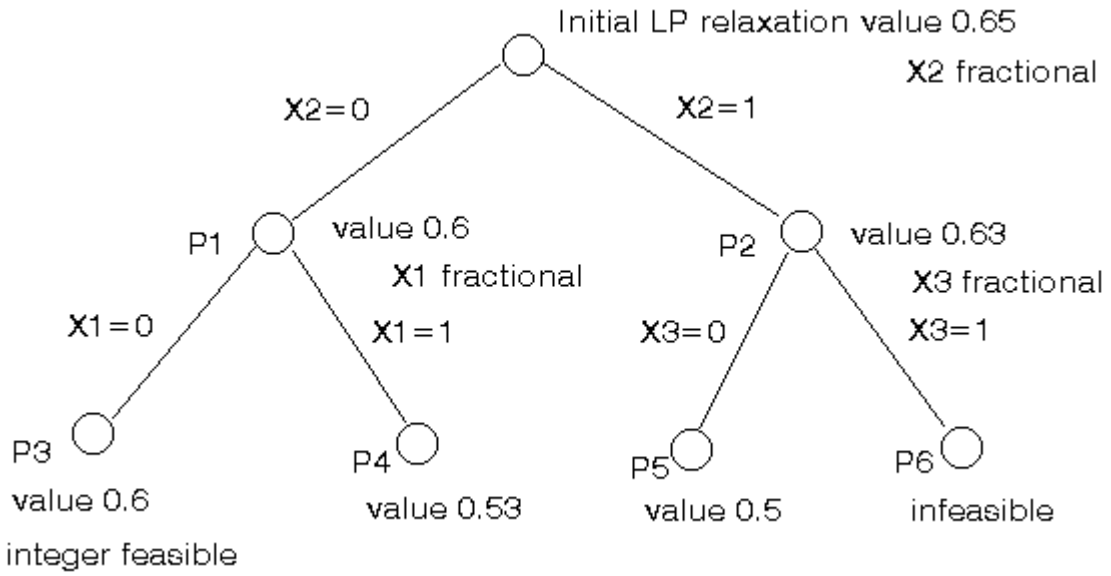
- P2 - original LP relaxation plus $x_2=1$, solution $x_2=1$, $x_3=0.67$, $x_1=x_4=0$ of value 0.63

Branching on x_3 we get

- P5 - original LP relaxation plus $x_2=1$ (P2) plus $x_3=0$, solution $x_1=x_2=1$, $x_3=x_4=0$ of value 0.5
- P6 - original LP relaxation plus $x_2=1$ (P2) plus $x_3=1$, problem infeasible

Neither of P5 or P6 lead to further branching so we are done, we have discovered the optimal integer solution of value 0.6 corresponding to $x_3=x_4=1$, $x_1=x_2=0$.

The entire process we have gone through to discover this optimal solution (and to prove that it is optimal) is shown graphically below.



You should be clear as to why 0.6 is the optimal integer solution for this problem, simply put if there were a better integer solution the above tree search process would (logically) have found it.

Note here that this method, like complete enumeration, also involves powers of two as we progress down the (binary) tree. However also note that we did not enumerate all possible integer solutions (of which there are 16). Instead here we solved 7 LP's. This is an important point, and indeed why tree search works at all. We do not need to examine as many LP's as there are possible solutions. Whilst the computational efficiency of tree search differs for different problems it is this basic fact that enables us to solve problems that would be completely beyond us were we to try complete enumeration.

You may have noticed that in the example above we never had more than one fractional variable in the LP solution at any tree node. This arises due to the fact that in constructing the above example I decided to make the situation as simple as possible. In general we might well have more than one fractional variable at a tree node and so we face a decision as to which variable to choose to branch on. A simple rule for deciding might be to take the fractional variable which is closest in value to 0.5, on the basis that the two branches (setting this variable to zero and one respectively) may well perturb the situation significantly.

Good computer packages (solvers) exist for finding optimal solutions to IP's/MIP's via LP-based tree search. Many of the computational advances in IP optimal solution methods (e.g. constraint aggregation, coefficient reduction, problem reduction, automatic generation of valid inequalities) are included in these packages. Often the key to making successful use of such packages for any particular problem is to put effort into a good formulation of the problem in terms of the variables and constraints. By this we mean that for any particular IP there may be a number of valid formulations. Deciding which formulation to adopt in a solution algorithm is often a combination of experience and trial and error.

Constraint logic programming (CLP), also called constraint programming, which is essentially branch and bound but without the bound, can be of use here if:

- the problem cannot be easily expressed in linear mathematics

- the gap between the LP relaxation solution and the IP optimal solution is so large as to render LP-based tree search impracticable.

General purpose heuristic solution algorithms

Essentially the only effective approach here is to run a general purpose optimal algorithm and terminate early (e.g. after a specified computer time).

Special purpose optimal solution algorithms

If we are dealing with one specific type of IP then we might well be able to develop a special purpose solution algorithm (designed for just this one type of IP) that is more effective computationally than the general purpose branch and bound method given earlier. These are typically tree search approaches based upon generating bounds via:

- dual ascent
- lagrangean relaxation and:
  - subgradientoptimisation; or
  - multiplier adjustment.

Such algorithms, being different for different problems, are really beyond the scope of this course but suffice to say:

- such algorithms draw upon the concepts, such as branch and bound, outlined previously
- such algorithms often use linear programming via LP relaxation
- such algorithms take advantage of the structure of the constraints of the IP they are solving.
- different methods have different computational performance and their general behaviour is that each method is computationally effective up to a certain size of problem and then becomes computationally ineffective (as the effort needed to obtain an optimal solution begins to increase exponentially in terms of the size of problem considered).

A large amount of academic effort in this field is devoted to generating methods that out-perform previous methods for the same problem.

On a personal note this is an area with which I am familiar and special purpose algorithms can be very effective (e.g. a problem dealing with the location of warehouses and involving some 500,000 continuous variables, 500 zero-one variables and 500,000 constraints solved in some 20 minutes). They can be very successful compared with general purpose optimal algorithms (perhaps an order of magnitude or more in terms of the size of problem that can be solved).

You should be clear though why a special purpose optimal algorithm can be so computationally more effective than a general purpose optimal algorithm, it is because you have to put intellectual effort into designing the algorithm!

Special purpose heuristic solution algorithms

With regard to heuristics we have a number of generic approaches in the literature, for example:

- greedy

- interchange
- bound based heuristics (e.g. lagrangean heuristics)
- tabu search
- simulated annealing
- population heuristics (e.g. genetic algorithms)

By generic here we mean that there is a general framework/approach from which to build an algorithm. All of these generic approaches however must be tailored for the particular IP we are considering. In addition we can design heuristics purely for the particular problem we are considering (problem-specific heuristics).

Structured problems

To illustrate the concept of structure in a mathematical program consider the following LP:

maximise $c_1x_1 + c_2x_2 + c_3x_3 + c_4 x_4 + c_5x_5 + c_6x_6$
subject to
$x_1+x_2<= 1$
$x_3+x_4<= 1$
$x_5+x_6<= 1$
$x_i>= 0$ i=1,...,6

where the $c_i>=0$ i=1,...,6 are known constants.

Whilst it is plain that we could solve this LP (via a computer package using simplex or interior point) it is obvious that the constraints of this LP have a special structure - namely

- each variable appears in just one constraint; and
- each constraint involves only two variables.

Using this information it is clear that the optimal solution of this LP can be obtained by inspection (technically in polynomial time) and is given by

$x_1 = 1$ if $c_1>=c_2$
  $= 0$ otherwise
$x_2 = 1-x_1$
$x_3 = 1$ if $c_3>=c_4$
  $= 0$ otherwise
$x_4 = 1-x_3$
$x_5 = 1$ if $c_5>=c_6$
  $= 0$ otherwise
$x_6 = 1-x_5$

with the optimal objective function value being given by

$\max[c_1,c_2] + \max[c_3,c_4] + \max[c_5,c_6]$

i.e. we have obtained the optimal solution without using a complicated algorithm (and, in computer terms, at considerably less computational cost).

In essence structured problems are problems which can be formulated as LP's/IP's but which, because of the structure of the constraints, can be solved much more efficiently by algorithms which take

advantage of this structure (as compared with solving them by using the general LP (simplex/interior point) or IP (tree search) algorithms).

Examples of structured problems considered in this course are:

- network analysis (PERT/CPM)
- network flow (i.e. assignment, transportation, transhipment and maximum flow)
- shortest path
- spanning tree .

LP relaxation

For any IP we can generate an LP (called the LP relaxation) from the IP by taking the same objective function and same constraints but with the requirement that variables are integer replaced by appropriate continuous constraints

e.g. $x_i = 0$ or 1 can be replaced by the two continuous constraints $x_i >= 0$ and $x_i <= 1$

We can then solve this LP relaxation of the original IP.

For example consider the IP for the capital budgeting problem considered before.

The LP relaxation of this IP is given by:

maximise $0.2x_1 + 0.3x_2 + 0.5x_3 + 0.1x_4$

subject to

$0.5x_1 + 1.0x_2 + 1.5x_3 + 0.1x_4 <= 3.1$

$0.3x_1 + 0.8x_2 + 1.5x_3 + 0.4x_4 <= 2.5$

$0.2x_1 + 0.2x_2 + 0.3x_3 + 0.1x_4 <= 0.4$

$x_j <= 1$ j=1,...,4

$x_j >= 0$ j=1,...,4

If we solve this LP then:

- the LP solution might turn out to have all variables taking integer values at the LP optimal solution - if so we have found the optimal integer solution (in such a case we have been lucky and the LP is said to be "naturally integer").

However for the LP relaxation of the capital budgeting problem the LP solution is $x_1=0$, $x_2=0.5$, $x_3=1$, $x_4=0$, so we have been unlucky.

If we have variables taking fractional values at the LP optimal solution then we can round these to the nearest integer value. If we do this then:

- this may lead to certain constraints being violated (i.e. we have an infeasible solution) - this may, or may not, be important.

For example if we round the LP solution given above to $x_1=0$, $x_2=1$, $x_3=1$, $x_4=0$ then the constraints relating to the amount of available capital in year 3 is violated.

- the solution may be feasible (all constraints satisfied) but the solution may not be the optimal integer solution.

For example if we round the LP solution given above to $x_1=0$, $x_2=0$, $x_3=1$, $x_4=0$ then all the constraints are satisfied and we have a feasible solution of value 0.5.

In fact for this particular (very small) problem this is not the optimal solution. In general we can say that the rounded LP relaxation solution is almost certainly non-optimal and may be very non-optimal (in the sense that the objective function value of the rounded LP relaxation solution is very far away from the objective function value of the optimal integer solution).

In general the solution process used in many LP based IP packages is

- specify the IP: objective, constraints, integer variables (typically x_j = integer between 0 and n).
- automatically generate the LP relaxation: same objective as IP, same constraints as IP with the addition of $0<=x_j<=n$, variables as before but no longer required to be integer
- use the tree search procedure to generate the optimal solution to the IP.

Integer programming formulation examples

Capital budgeting extension

For the integer programming problem given before related to capital budgeting suppose now that we have the additional condition that either project 1 or project 2 must be chosen (i.e. projects 1 and 2 are mutually exclusive). To cope with this condition we enlarge the IP given above in the following manner.

This condition is an example of an either/or condition "either project 1 or project 2 must be chosen". In terms of the variables we have already defined this condition is "either $x_1 = 1$ or $x_2 = 1$". The standard trick for dealing with either/or conditions relating to two zero-one variables is to add to the IP the constraint

- $x_1 + x_2 = 1$

To verify that this mathematical description is equivalent to the verbal description "either $x_1 = 1$ or $x_2 = 1$" we use the fact that x_1 and x_2 are both zero-one variables.

Choose any one of these two variables (x_1, say) and tabulate, for each possible value of the chosen variable, the meaning of the corresponding mathematical and verbal descriptions as below.

Variable value	Mathematical description	Verbal description
	$x_1 + x_2 = 1$	"either $x_1 = 1$ or $x_2 = 1$"
$x_1 = 0$	$x_2 = 1$	$x_2 = 1$

$x_1 = 1$ $1 + x_2 = 1$ $x_2 = 0$
i.e. $x_2 = 0$

It is clear from this tabulation that the mathematical and verbal descriptions are equivalent with the proviso that we have interpreted the condition "either $x_1 = 1$ or $x_2 = 1$" as meaning that the case $x_1 = 1$ and $x_2 = 1$ (both projects 1 and 2 chosen) cannot occur. If this is not so (i.e. both projects 1 and 2 can be chosen) then we replace the constraint $x_1 + x_2 = 1$ by the constraint $x_1 + x_2 >= 1$ (i.e. at least one of projects 1 and 2 must be chosen).

Note here that, in general, we can regard formulating a problem as translating a verbal description of the problem into an equivalent mathematical description. What we have presented above is a similar process - translating a specific "either ... or ..." verbal description into an equivalent mathematical description. Whilst it is not usual to go to the lengths of a complete tabulation (as shown above) to verify that the verbal and mathematical descriptions are equivalent a complete tabulation is helpful in clarifying thought and detecting incorrect mathematical descriptions.

Other problems

The main areas in which IP is used in practice include:

- imposition of logical conditions in LP problems (such as the either/or condition dealt with above)
- blending with a limited number of ingredients
- depot location
- job shop scheduling
- assembly line balancing
- airline crew scheduling
- timetabling

Integer programming example

Recall the blending problem dealt with before under linear programming. To remind you of it we reproduce it below.

Blending problem

Consider the example of a manufacturer of animal feed who is producing feed mix for dairy cattle. In our simple example the feed mix contains two active ingredients and a filler to provide bulk. One kg of feed mix must contain a minimum quantity of each of four nutrients as below:

Nutrient	A	B	C	D
gram	90	50	20	2

The ingredients have the following nutrient values and cost

	A	B	C	D	Cost/kg
Ingredient 1 (gram/kg)	100	80	40	10	40

Ingredient 2 (gram/kg) 200 150 20 - 60

What should be the amounts of active ingredients and filler in one kg of feed mix?

Blending problem solution

Variables

In order to solve this problem it is best to think in terms of one kilogram of feed mix. That kilogram is made up of three parts - ingredient 1, ingredient 2 and filler so: let
x_1 = amount (kg) of ingredient 1 in one kg of feed mix
x_2 = amount (kg) of ingredient 2 in one kg of feed mix
x_3 = amount (kg) of filler in one kg of feed mix
where $x_1 >= 0$, $x_2 >= 0$ and $x_3 >= 0$.

Constraints

- balancing constraint (an implicit constraint due to the definition of the variables)

$x_1 + x_2 + x_3 = 1$

- nutrient constraints

$100x_1 + 200x_2 >= 90$ (nutrient A)
$80x_1 + 150x_2 >= 50$ (nutrient B)
$40x_1 + 20x_2 >= 20$ (nutrient C)
$10x_1 >= 2$ (nutrient D)

Note the use of an inequality rather than an equality in these constraints, following the rule we put forward in the Two Mines example, where we assume that the nutrient levels we want are lower limits on the amount of nutrient in one kg of feed mix.

Objective

Presumably to minimise cost, i.e.

minimise $40x_1 + 60x_2$

which gives us our complete LP model for the blending problem.

Suppose now we have the additional conditions:

- if we use any of ingredient 2 we incur a fixed cost of 15
- we need not satisfy all four nutrient constraints but need only satisfy three of them (i.e. whereas before the optimal solution required all four nutrient constraints to be satisfied now the optimal solution could (if it is worthwhile to do so) only have three (any three) of these nutrient constraints satisfied and the fourth violated.

9.2 Give the complete MIP formulation of the problem with these two new conditions added.

Solution

To cope with the condition that if x_2>=0 we have a fixed cost of 15 incurred we have the standard trick of introducing a zero-one variable y defined by

y = 1 if x_2>=0
= 0 otherwise

and

- add a term +15y to the objective function

and add the additional constraint

- x_2 <= [largest value x_2 can take]y

In this case it is easy to see that x_2 can never be greater than one and hence our additional constraint is x_2<= y.

To cope with condition that need only satisfy three of the four nutrient constraints we introduce four zero-one variables z_i (i=1,2,3,4) where

z_i = 1 if nutrient constraint i (i=1,2,3,4) is satisfied
= 0 otherwise

and add the constraint

- $z_1+z_2+z_3+z_4$ >= 3 (at least 3 constraints satisfied)

and alter the nutrient constraints to be

- $100x_1 + 200x_2 >= 90z_1$
- $80x_1 + 150x_2 >= 50z_2$
- $40x_1 + 20x_2 >= 20z_3$
- $10x_1 >= 2z_4$

The logic behind this change is that if a z_i=1 then the constraint becomes the original nutrient constraint which needs to be satisfied. However if a z_i=0 then the original nutrient constraint becomes

- same left-hand side >= zero

which (for the four left-hand sides dealt with above) is always true and so can be neglected - meaning the original nutrient constraint need not be satisfied. Hence the complete (MIP) formulation of the problem is given by

minimise $40x_1 + 60x_2 + 15y$
subject to

$x_1 + x_2 + x_3 = 1$
$100x_1 + 200x_2 >= 90z_1$
$80x_1 + 150x_2 >= 50z_2$
$40x_1 + 20x_2 >= 20z_3$
$10x_1 >= 2z_4$
$z_1 + z_2 + z_3 + z_4 >= 3$
$x_2 <= y$
$z_i = 0$ or 1 i=1,2,3,4
$y = 0$ or 1
$x_i >= 0$ i=1,2,3

Integer programming example

In the planning of the monthly production for the next six months a company must, in each month, operate either a normal shift or an extended shift (if it produces at all). A normal shift costs £100,000 per month and can produce up to 5,000 units per month. An extended shift costs £180,000 per month and can produce up to 7,500 units per month. Note here that, for either type of shift, the cost incurred is fixed by a union guarantee agreement and so is independent of the amount produced.

It is estimated that changing from a normal shift in one month to an extended shift in the next month costs an extra £15,000. No extra cost is incurred in changing from an extended shift in one month to a normal shift in the next month.

The cost of holding stock is estimated to be £2 per unit per month (based on the stock held at the end of each month) and the initial stock is 3,000 units (produced by a normal shift). The amount in stock at the end of month 6 should be at least 2,000 units. The demand for the company's product in each of the next six months is estimated to be as shown below:

Month	1	2	3	4	5	6
Demand	6,000	6,500	7,500	7,000	6,000	6,000

Production constraints are such that if the company produces anything in a particular month it must produce at least 2,000 units. If the company wants a production plan for the next six months that avoids stockouts, formulate their problem as an integer program.

Hint: first formulate the problem allowing non-linear constraints and then attempt to make all the constraints linear.

Solution

Variables

The decisions that have to be made relate to:

- whether to operate a normal shift or an extended shift in each month; and
- how much to produce each month.

Hence let:

$x_t = 1$ if we operate a normal shift in month t (t=1,2,...,6)

= 0 otherwise
y_t = 1 if we operate an extended shift in month t (t=1,2,...,6)
= 0 otherwise
P_t (>= 0) be the amount produced in month t (t=1,2,...,6)

In fact, for this problem, we can ease the formulation by defining three additional variables - namely let:

z_t = 1 if we switch from a normal shift in month t-1 to an extended shift in month t (t=1,2,...,6)
= 0 otherwise
I_t be the closing inventory (amount of stock left) at the end of month t (t=1,2,...,6)

w_t = 1 if we produce in month t, and hence from the production constraints P_t >= 2000 (t=1,2,...,6)
= 0 otherwise (i.e. P_t = 0)

The motivation behind introducing the first two of these variables (z_t, I_t) is that in the objective function we will need terms relating to shift change cost and inventory holding cost. The motivation behind introducing the third of these variables (w_t) is the production constraint "either P_t = 0 or P_t >= 2000", which needs a zero-one variable so that it can be dealt with using the standard trick for "either/or" constraints.

In any event formulating an IP tends to be an iterative process and if we have made a mistake in defining variables we will encounter difficulties in formulating the constraints/objective. At that point we can redefine our variables and reformulate.

Constraints

We first express each constraint in words and then in terms of the variables defined above.

- only operate (at most) one shift each month

$x_t + y_t <= 1 \quad t=1,2,...,6$

Note here that we could not have made do with just one variable (x_t say) and defined that variable to be one for a normal shift and zero for an extended shift (since in that case what if we decide not to produce in a particular month?).

Although we could have introduced a variable indicating no shift (normal or extended) operated in a particular month this is not necessary as such a variable is equivalent to $1-x_t-y_t$.

- production limits not exceeded

$P_t <= 5000x_t + 7500y_t \quad t=1,2,...,6$

Note here the use of addition in the right-hand side of the above equation where we are making use of the fact that at most one of x_t and y_t can be one and the other must be zero.

- no stockouts

$I_t >= 0 \quad t=1,2,...,6$

- we have an inventory continuity equation of the form

closing stock = opening stock + production - demand

where $I_0 = 3000$. Hence letting D_t = demand in month t (t=1,2,...,6) (a known constant) and assuming

- that opening stock in period t = closing stock in period t-1 and
- that production in period t is available to meet demand in period t

we have that

$I_t = I_{t-1} + P_t - D_t \quad t=1,2,...,6$

As noted above this equation assumes that we can meet demand in the current month from goods produced that month. Any time lag between goods being produced and becoming available to meet demand is easily incorporated into the above equation. For example for a 2 month time lag we replace P_t in the above equation by P_{t-2} and interpret I_t as the number of goods in stock at the end of month t which are available to meet demand i.e. goods are not regarded as being in stock until they are available to meet demand. Inventory continuity equations of the type shown are common in production planning problems.

- the amount in stock at the end of month 6 should be at least 2000 units $I_6 >= 2000$

- production constraints of the form "either $P_t = 0$ or $P_t >= 2{,}000$".

Here we make use of the standard trick we presented for "either/or" constraints. We have already defined appropriate zero-one variables w_t (t=1,2,...,6) and so we merely need the constraints

$P_t <= Mw_t \quad t=1,2,...,6$
$P_t >= 2000w_t \quad t=1,2,...,6$

Here M is a positive constant and represents the most we can produce in any period t (t=1,2,...,6). A convenient value for M for this example is M = 7500 (the most we can produce irrespective of the shift operated).

- we also need to relate the shift change variable z_t to the shifts being operated

The obvious constraint is

$z_t = x_{t-1}y_t \quad t=1,2,...,6$

since as both x_{t-1} and y_t are zero-one variables z_t can only take the value one if both x_{t-1} and y_t are one (i.e. we operate a normal shift in period t-1 and an extended shift in period t). Looking back to the verbal description of z_t it is clear that the mathematical description given above is equivalent to that verbal description. (Note here that we define $x_0 = 1$ ($y_0 = 0$)).

This constraint is non-linear. However we are told that we can first formulate the problem with non-linear constraints and so we proceed. We shall see later how to linearise (generate equivalent linear constraints for) this equation.

Objective

We wish to minimise total cost and this is given by

SUM{t=1,...,6}($100000x_t + 180000y_t + 15000z_t + 2I_t$)

Hence our formulation is complete.

Note that, in practise, we would probably regard I_t and P_t as taking fractional values and round to get integer values (since they are both large this should be acceptable). Note too here that this is a non-linear integer program.

Comments

In practice a model of this kind would be used on a "rolling horizon" basis whereby every month or so it would be updated and resolved to give a new production plan.

The inventory continuity equation presented is quite flexible, being able to accommodate both time lags (as discussed previously) and wastage. For example if 2% of the stock is wasted each month due to deterioration/pilfering etc then the inventory continuity equation becomes $I_t = 0.98I_{t-1} + P_t - D_t$. Note that, if necessary, the objective function can include a term related to $0.02I_{t-1}$ to account for the loss in financial terms.

In order to linearise (generate equivalent linear constraints) for our non-linear constraint we again use a standard trick. Note that that equation is of the form

- A = BC

where A, B and C are zero-one variables. The standard trick is that a non-linear constraint of this type can be replaced by the two linear constraints

- A <= (B+C)/2 and
- A >= B+C-1

To see this we use the fact that as B and C take only zero-one values there are only four possible cases to consider:

B	C	A = BC	A <= (B+C)/2	A >= B+C-1
Becomes				
0	0	A = 0	A <= 0	A >= -1
0	1	A = 0	A <= 0.5	A >= 0
1	0	A = 0	A <= 0.5	A >= 0
1	1	A = 1	A <= 1	A >= 1

Then, recalling that A can also only take zero-one values, it is clear that in each of the four possible cases the two linear constraints (A <= (B+C)/2 and A >= B+C-1) are equivalent to the single non-linear constraint (A=BC).

Returning now to our original non-linear constraint

- $z_t = x_{t-1}y_t$

this involves the three zero-one variables z_t, x_{t-1} and y_t and so we can use our general rule and replace this non-linear constraint by the two linear constraints

$z_t <= (x_{t-1} + y_t)/2 \quad t=1,2,...,6$
and$z_t >= x_{t-1} + y_t - 1 \quad t=1,2,...,6$

Making this change transforms the non-linear integer program given before into an equivalent linear integer program.

Integer programming example 1996 MBA exam

A toy manufacturer is planning to produce new toys. The setup cost of the production facilities and the unit profit for each toy are given below. :

Toy	Setup cost (£)	Profit (£)
1	45000	12
2	76000	16

The company has two factories that are capable of producing these toys. In order to avoid doubling the setup cost only one factory could be used.

The production rates of each toy are given below (in units/hour):

	Toy 1	Toy 2
Factory 1	52	38
Factory 2	42	23

Factories 1 and 2, respectively, have 480 and 720 hours of production time available for the production of these toys. The manufacturer wants to know which of the new toys to produce, where and how many of each (if any) should be produced so as to maximise the total profit.

- Introducing 0-1 decision variables formulate the above problem as an integer program. (Do not try to solve it).
- Explain briefly how the above mathematical model can be used in production planning.

Solution

Variables

We need to decide whether to setup a factory to produce a toy or not so let $f_{ij} = 1$ if factory i (i=1,2) is setup to produce toys of type j (j=1,2), 0 otherwise

We need to decide how many of each toy should be produced in each factory so let x_{ij} be the number of toys of type j (j=1,2) produced in factory i (i=1,2) where $x_{ij} >= 0$ and integer.

Constraints

- at each factory cannot exceed the production time available

$x_{11}/52 + x_{12}/38 <= 480$
$x_{21}/42 + x_{22}/23 <= 720$

- cannot produce a toy unless we are setup to do so

$x_{11} <= 52(480)f_{11}$
$x_{12} <= 38(480)f_{12}$
$x_{21} <= 42(720)f_{21}$
$x_{22} <= 23(720)f_{22}$

Objective

The objective is to maximise total profit, i.e.

maximise $12(x_{11}+ x_{21}) + 16(x_{12}+ x_{22}) - 45000(f_{11} + f_{21}) - 76000(f_{12} + f_{22})$

Note here that the question says that in order to avoid doubling the setup costs only one factory could be used. This is not a constraint. We can argue that if it is only cost considerations that prevent us using more than one factory these cost considerations have already been incorporated into the model given above and the model can decide for us how many factories to use, rather than we artificially imposing a limit via an explicit constraint on the number of factories that can be used.

The above mathematical model could be used in production planning in the following way:

- enables us to maximise profit, rather than relying on an ad-hoc judgemental approach
- can be used for sensitivity analysis - for example to see how sensitive our production planning decision is to changes in the production rates
- enables us to see the effect upon production of (say) increasing the profit per unit on toy 1
- enables us to easily replan production in the event of a change in the system (e.g. a reduction in the available production hours at factory one due to increased work from other products made at that factory)
- can use on a rolling horizon basis to plan production as time passes (in which case we perhaps need to introduce a time subscript into the above model)

Integer programming

A project manager in a company is considering a portfolio of 10 large project investments. These investments differ in the estimated long-run profit (net present value) they will generate as well as in the amount of capital required.

Let P_j and C_j denote the estimated profit and capital required (both given in units of millions of £) for investment opportunity j (j=1,...,10) respectively. The total amount of capital available for these investments is Q (in units of millions of £)

Investment opportunities 3 and 4 are mutually exclusive and so are 5 and 6. Furthermore, neither 5 nor 6 can be undertaken unless either 3 or 4 is undertaken. At least two and at most four investment opportunities have to be undertaken from the set {1,2,7,8,9,10}.

The project manager wishes to select the combination of capital investments that will maximise the total estimated long-run profit subject to the restrictions described above.

Formulate this problem using an integer programming model and comment on the difficulties of solving this model. (Do not actually solve it).

What are the advantages and disadvantages of using this model for portfolio selection?

Solution

Variables

We need to decide whether to use an investment opportunity or not so let $x_j = 1$ if we use investment opportunity j (j=1,...,10), 0 otherwise

Constraints

- total amount of capital available for these investments is Q

SUM{j=1,...,10}C_jx_j <= Q

- investment opportunities 3 and 4 are mutually exclusive and so are 5 and 6

$x_3 + x_4 <= 1$
$x_5 + x_6 <= 1$

- neither 5 nor 6 can be undertaken unless either 3 or 4 is undertaken

$x_5 <= x_3 + x_4$
$x_6 <= x_3 + x_4$

- at least two and at most four investment opportunities have to be undertaken from the set {1,2,7,8,9,10}

$x_1 + x_2 + x_7 + x_8 + x_9 + x_{10} >= 2$
$x_1 + x_2 + x_7 + x_8 + x_9 + x_{10} <= 4$

Objective

The objective is to maximise the total estimated long-run profit i.e.

maximise SUM{j=1,...,10}P_jx_j

The model given above is a very small zero-one integer programming problem with just 10 variables and 7 constraints and should be very easy to solve. For example even by complete (total) enumeration there are just $2^{10} = 1024$ possible solutions to be examined.

The advantages and disadvantages of using this model for portfolio selection are:

- enables us to maximise profit, rather than relying on an ad-hoc judgemental approach
- can be easily extended to deal with a larger number of potential investment opportunities

- can be used for sensitivity analysis, for example to see how sensitive our portfolio selection decision is to changes in the data
- the model fails to take into account any statistical uncertainly (risk) in the data, it is a completely deterministic model, for example project j might have a (known or estimated) statistical distribution for its profit P_j and so we might need a model that takes this distribution into account

---

Integer programming example 1994 MBA exam

A food is manufactured by refining raw oils and blending them together. The raw oils come in two categories:

- Vegetable oil:
  - VEG1
  - VEG2
- Non-vegetable oil:
  - OIL1
  - OIL2
  - OIL3

The prices for buying each oil are given below (in £/tonne)

VEG1	VEG2	OIL1	OIL2	OIL3
115	128	132	109	114

The final product sells at £180 per tonne. Vegetable oils and non-vegetable oils require different production lines for refining. It is not possible to refine more than 210 tonnes of vegetable oils and more than 260 tonnes of non-vegetable oils. There is no loss of weight in the refining process and the cost of refining may be ignored.

There is a technical restriction relating to the hardness of the final product. In the units in which hardness is measured this must lie between 3.5 and 6.2. It is assumed that hardness blends linearly and that the hardness of the raw oils is:

VEG1	VEG2	OIL1	OIL2	OIL3
8.8	6.2	1.9	4.3	5.1

It is required to determine what to buy and how to blend the raw oils so that the company maximises its profit.

- Formulate the above problem as a linear program. (Do not actually solve it).
- What assumptions do you make in solving this problem by linear programming?

The following extra conditions are imposed on the food manufacture problem stated above as a result of the production process involved:

- the food may never be made up of more than 3 raw oils
- if an oil (vegetable or non-vegetable) is used, at least 30 tonnes of that oil must be used
- if either of VEG1 or VEG2 are used then OIL2 must also be used

Introducing 0-1 integer variables extend the linear programming model you have developed to encompass these new extra conditions.

Solution

Variables

We need to decide how much of each oil to use so let x_i be the number of tonnes of oil of type i used (i=1,...,5) where i=1 corresponds to VEG1, i=2 corresponds to VEG2, i=3 corresponds to OIL1, i=4 corresponds to OIL2 and i=5 corresponds to OIL3 and where $x_i >= 0$ i=1,...,5

Constraints

- cannot refine more than a certain amount of oil

$x_1 + x_2 <= 210$
$x_3 + x_4 + x_5 <= 260$

- hardness of the final product must lie between 3.5 and 6.2

$(8.8x_1 + 6.2x_2 + 1.9x_3 + 4.3x_4 + 5.1x_5)/(x_1 + x_2 + x_3 + x_4 + x_5) >= 3.5$
$(8.8x_1 + 6.2x_2 + 1.9x_3 + 4.3x_4 + 5.1x_5)/(x_1 + x_2 + x_3 + x_4 + x_5) <= 6.2$

Objective

The objective is to maximise total profit, i.e.

maximise $180(x_1 + x_2 + x_3 + x_4 + x_5) - 115x_1 - 128x_2 - 132x_3 - 109x_4 - 114x_5$

The assumptions we make in solving this problem by linear programming are:

- all data/numbers are accurate
- hardness does indeed blend linearly
- no loss of weight in refining
- can sell all we produce

Integer program

Variables

In order to deal with the extra conditions we need to decide whether to use an oil or not so let $y_i = 1$ if we use any of oil i (i=1,...,5), 0 otherwise

Constraints

- must relate the amount used (x variables) to the integer variables (y) that specify whether any is used or not

$x_1 <= 210y_1$
$x_2 <= 210y_2$
$x_3 <= 260y_3$

$x_4 <= 260y_4$
$x_5 <= 260y_5$

- the food may never be made up of more than 3 raw oils

$y_1 + y_2 + y_3 + y_4 + y_5 <= 3$

- if an oil (vegetable or non-vegetable) is used, at least 30 tonnes of that oil must be used

$x_i >= 30y_i \quad i=1,...,5$

- if either of VEG1 or VEG2 are used then OIL2 must also be used

$y_4 >= y_1$
$y_4 >= y_2$

Objective

The objective is unchanged by the addition of these extra constraints and variables.

Integer programming example 1985 UG exam

A factory works a 24 hour day, 7 day week in producing four products. Since only one product can be produced at a time the factory operates a system where, throughout one day, the same product is produced (and then the next day either the same product is produced or the factory produces a different product). The rate of production is:

Product	1	2	3	4
No. of units produced per hour worked	100	250	190	150

The only complication is that in changing from producing product 1 one day to producing product 2 the next day five working hours are lost (from the 24 hours available to produce product 2 that day) due to the necessity of cleaning certain oil tanks.

To assist in planning the production for the next week the following data is available:

Product	Current stock (units)	Demand (units) for each day of the week 1	2	3	4	5	6	7
1	5000	1500	1700	1900	1000	2000	500	500
2	7000	4000	500	1000	3000	500	1000	2000
3	9000	2000	2000	3000	2000	2000	2000	500
4	8000	3000	2000	2000	1000	1000	500	500

Product 3 was produced on day 0. The factory is not allowed to be idle (i.e. one of the four products must be produced each day). Stockouts are not allowed. At the end of day 7 there must be (for each product) at least 1750 units in stock.

If the cost of holding stock is £1.50 per unit for products 1 and 2 but £2.50 per unit for products 3 and 4 (based on the stock held at the end of each day) formulate the problem of planning the production for the next week as an integer program in which all the constraints are linear.

Solution

Variables

The decisions that have to be made relate to the type of product to produce each day. Hence let:

- x_{it} = 1 if produce product i (i=1,2,3,4) on day t (t=1,2,3,4,5,6,7) = 0 otherwise

In fact, for this problem, we can ease the formulation by defining two additional variables - namely let:

- I_{it} be the closing inventory (amount of stock left) of product i (i=1,2,3,4) on day t (t=1,2,...,7)
- P_{it} be the number of units of product i (i=1,2,3,4) produced on day t (t=1,2,...,7)

Constraints

- only produce one product per day

$x_{1t} + x_{2t} + x_{3t} + x_{4t} = 1 \quad t=1,2,...,7$

- no stockouts

$I_{it} >= 0 \quad i=1,...,4 \; t=1,...,7$

- we have an inventory continuity equation of the form

closing stock = opening stock + production - demand

Letting D_{it} represent the demand for product i (i=1,2,3,4) on day t (t=1,2,...,7) we have

$I_{10} = 5000$
$I_{20} = 7000$
$I_{30} = 9000$
$I_{40} = 8000$

representing the initial stock situation and

$I_{it} = I_{it-1} + P_{it} - D_{it} i=1,...,4 \; t=1,...,7$

for the inventory continuity equation.

Note here that we assume that we can meet demand in month t from goods produced in month t and also that the opening stock in month t = the closing stock in month $_{t-1}$.

- production constraint

9.2

Let R_i represent the work rate (units/hour) for product i (i=1,2,3,4) then the production constraint is

$P_{it} = x_{it}[24R_i]$ i=1,3,4 t=1,...,7

which covers the production for all except product 2 and

$P_{2t} = [24R_2]x_{2t} - [5R_2]x_{2t}x_{1t-1}$ t=1,...,7

i.e. for product 2 we lose 5 hours production if we are producing product 2 in period t and we produced product 1 the previous period. Note here that we initialise by

$x_{10} = 0$

since we know we were not producing product 1 on day 0. Plainly the constraint involving P_{2t} is non-linear as it involves a term which is the product of two variables. However we can linearise it by using the trick that given three zero-one variables (A,B,C say) the non-linear constraint

- A = BC

can be replaced by the two linear constraints

- A <= (B + C)/2 and
- A >= B + C - 1

Hence introduce a new variable Z_t defined by the verbal description

Z_t = 1 if produce product 2 on day t and product 1 on day t-1
= 0 otherwise

Then

$Z_t = x_{2t}x_{1t-1}$ t=1,...,7

and our non-linear equation becomes

$P_{2t} = [24R_2]x_{2t} - [5R_2]Z_t$ t=1,...,7

and applying our trick the non-linear equation for Z_t can be replaced by the two linear equations

$Z_t <= (x_{2t} + x_{1t-1})/2$ t=1,...,7
$Z_t >= x_{2t} + x_{1t-1} - 1$ t=1,...,7

- closing stock

$I_{i7} >= 1750$ i=1,...,4

- all variables >= 0 and integer, (x_{it}) and (Z_t) zero-one variables

Note that, in practise, we would probably regard (I_{it}) and (P_{it}) as taking fractional values and round to get integer values (since they are both quite large this should be acceptable).

Objective

We wish to minimise total cost and this is given by

$SUM\{t=1,...,7\}(1.50I_{1t} + 1.50I_{2t} + 2.50_{I3t} + 2.50I_{4t})$

Note here that this program may not have a feasible solution, i.e. it may simply not be possible to satisfy all the constraints. This is irrelevant to the process of constructing the model however. Indeed one advantage of the model may be that it will tell us (once a computational solution technique is applied) that the problem is infeasible.

Integer programming example 1987 UG exam

A company is attempting to decide the mix of products which it should produce next week. It has seven products, each with a profit (£) per unit and a production time (man-hours) per unit as shown below:

Product	Profit (£ per unit)	Production time (man-hours per unit)
1	10	1.0
2	22	2.0
3	35	3.7
4	19	2.4
5	55	4.5
6	10	0.7
7	115	9.5

The company has 720 man-hours available next week.

- Formulate the problem of how many units (if any) of each product to produce next week as an integer program in which all the constraints are linear.

Incorporate the following additional restrictions into your integer program (retaining linear constraints and a linear objective):

- If any of product 7 are produced an additional fixed cost of £2000 is incurred.
- Each unit of product 2 that is produced over 100 units requires a production time of 3.0 man-hours instead of 2.0 man-hours (e.g. producing 101 units of product 2 requires 100(2.0) + 1(3.0) man-hours).
- If both product 3 and product 4 are produced 75 man-hours are needed for production line set-up and hence the (effective) number of man-hours available falls to 720 - 75 = 645.