Backtracking :-

It is the one of the most general technique which is used to solve many problems which deals with searching for a set of solution or optimal solution which satisfy some constraints.

Using exhaustic search we collects all the possible solutions and see which one produces the optimum result. Backtracking technique is usually faster than exhaustic search.

- In many applications of backtrack method the desired solution is expressed as n-tuple $(x_1, x_2, \ldots x_n)$ where x_i are chosen from some finite set S_i. Then problem can be solved by finding one vector that that maximizes or minimizes, or satisfies a criterion function $P(x_1, x_2, \ldots x_n)$.

- For example sorting array of integers in $a[1:n]$ is a problem whose solution is express by n tuple, where x_i is the index in $a[]$. The criterian function P is $a[x_i] <= a[x_{i+1}]$ for $1 \leq i < n$

Date / /
Page

Set S_i is finite set which include integers from 1 to n.

- Major advantage of such algorithms is that we realize the fact that partial vector generated does not lead an optimal solution.

- Backtracking algorithm determine the solution by systematically searching the solution space for the given problem.

- Many problems we solve using backtracking require that all the solution satisfy a complex set of constraints. For any problem this constraints can be
- divided into explicit and implicit.

- Explicit constraints are the rules that restrict each x_i (vector element) to take on values from given set only. Explicit constraints are depend on perticular instance I of the problem. All the tuples that satisfy explicit constraints define a possible solution space for I.

- Implicit constraints are rules that determine which of tuples in solution space of I satisfy the criterion fun.

Date / /
Page

Set S_i is finite set which include integers from 1 to n.

- Major advantage of such algorithms is that we realize the fact that partial vector generated does not lead an optimal solution.

- Backtracking algorithm determine the solution by systematically searching the solution space for the given problem.

- Many problems we solve using backtracking require that all the solution satisfy a complex set of constraints. For any problem this constraints can be
- divided into explicit and implicit.

- Explicit constraints are the rules that restrict each x_i (vector element) to take on values from given set only. Explicit constraints are depend on perticular instance I of the problem. All the tuples that satisfy explicit constraints define a possible solution space for I.

- Implicit constraints are rules that determine which of tuples in solution space of I satisfy the criterion fun.

Thus implicit constraints describe the way in which the x_i must relate to each other.

example - 4 Queen problem

Consider 4 queen problem is solve using backtracking. We start with the root node as the only live node. This become E-node. We generate one child. Let us assume that children generated is ascending order.

Fig. (b) shows board configuration of 4-queen problem. The dots indicate placements of queen tried and rejected because other queen was attacking.

In (I) first queen is placed at [1,1] i.e. 1st row, 1st column. As the one queen is placed ~~on~~ in 1st row jump to the next row and try to place second queen

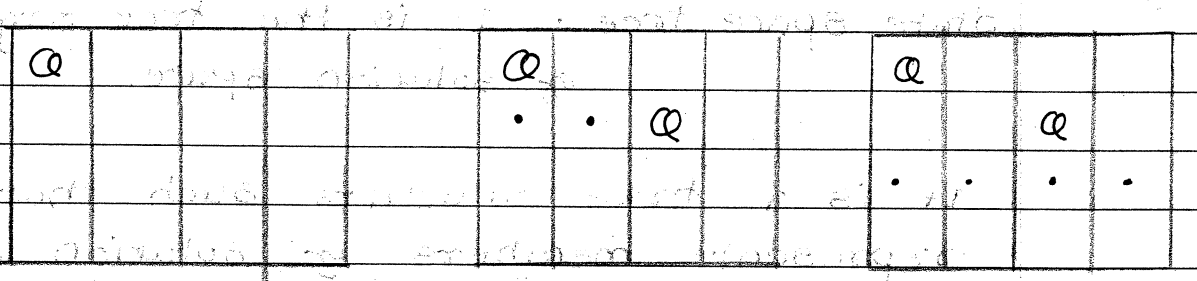
- In (II) dots shows tried placements and at last we place queen at 2nd row, 3rd column i.e [2,3]

- In [III] we tried to place queen but we can not cause at each column any one of the queen is attacking, so we can not place the queen hence we backtrack to 2nd queen

(4)

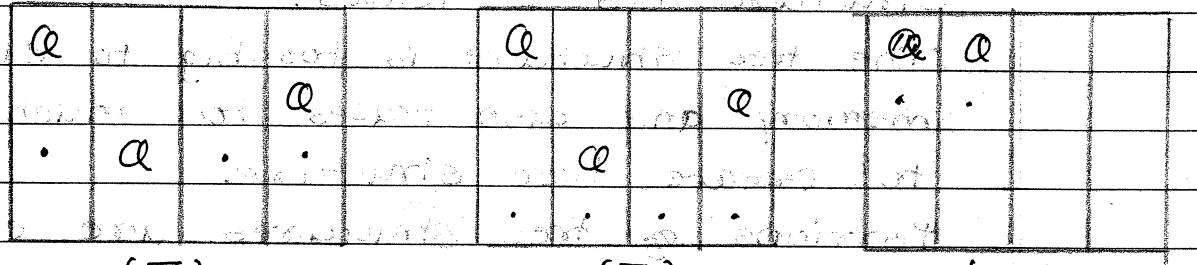
Date / /
Page

In (IV) we have one option left to place queen at [2,4] hence try that option. Then place 3rd queen at [3,2] as shown in (V) Then try for 4th queen again we failed to place queen so backtrack to 3rd queen but there are no more options left for 3rd queen so backtrack to 2nd here also the same condition then backtrack to 1st and go for next location as shown in (VI) and then follow same procedure so and we get the solution.

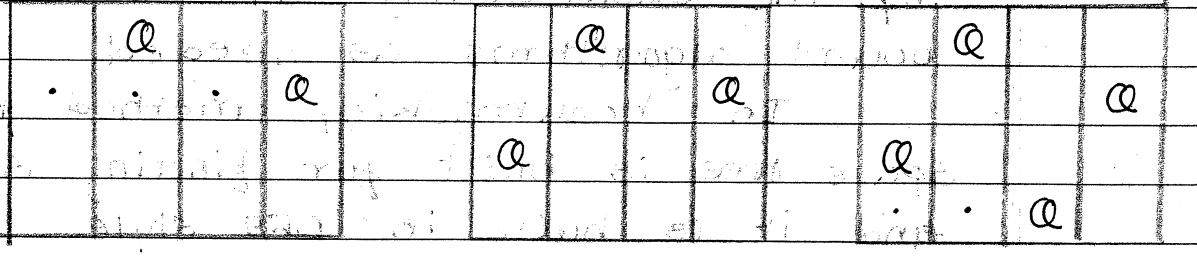


(I) (II) (III)

stuck! backtrack



(IV) (V) (VI)



(VII) (VIII) (IX)

We stuck at (III) so backtrack to 2nd queen or row go for next option and continue.

(5)

Date / /
Page

★ Tree organisation of solution space :-

- Each node in this tree defines a problem state.
- All the paths set up a tree structure such that leaves represent members of the solution space.

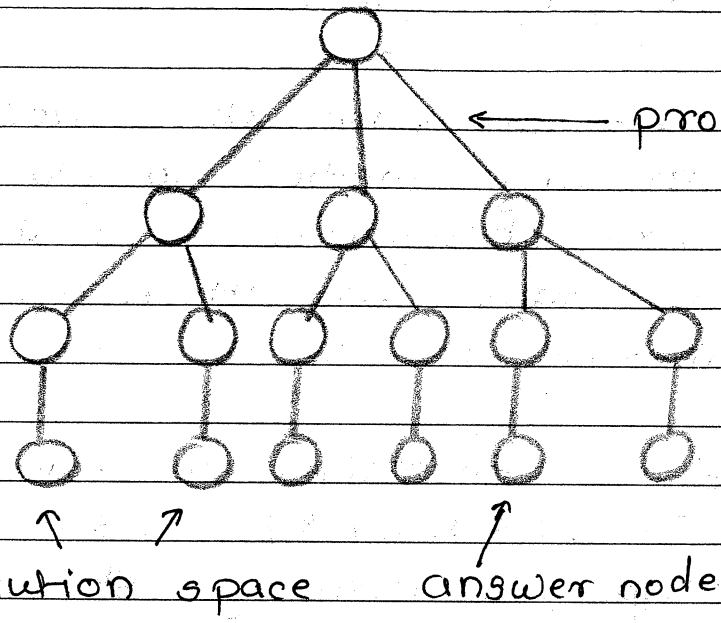
Defination -

State space tree - It is the tree organization of solution space.

- It is a tree structure such that leaves represent members of solution space.
- For a size n subset problem, this tree structure has 2^n leaves.
- The tree structure is too big to store in memory and also takes too much time to create tree structure.
- Portions of tree structures are created by the backtracking and branch and bound algorithms as needed.

In backtracking method the state space tree is built for finding solution and it is built in DFS style.

In fig. (a) a simple state space tree is shown having problem state, solution state etc.



Terminology -

- Problem state - It is each node in DFS tree
- State space - It is the set of all paths from root node to other node.
- Solution states - These are the problem states s for which the path from the root node to s defines a tuple in the solution space.
- Answer states - These are those solution states s for which the path from root node to s defines a tuple that is a member of set of solutions.
- Live node - It is a generated node for which all of the ~~node~~ children have not been generated yet.
- E-node - It is a live node whose childrens are currently being generated or explored.

(7)

Date / /
Page

* The 8-queen problem :-

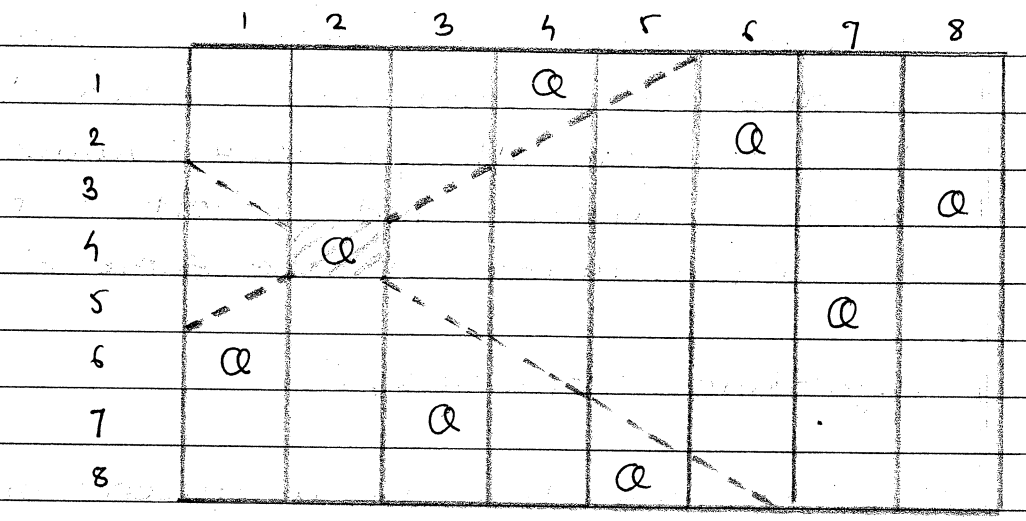
We can solve this problem using backtracking method to place 8 queens on 8×8 chessboard so that no two queen attack each other. Queens are numbered 1 to 8.

- Explicit constraints to this problem $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$ $1 \le i \le 8$ therefore sol^n space consists of 8^8 tuples.

- Implicit constraints to this problem are that no two x_i can be the same. (i.e all queens must be on different column and no two queens can be on same diagonal.)

- If we imagine that chessboard squares being numbered as the indices of the two dimensional array $a[1:n, 1:n]$, the we observed that every element on same diagonal that runs from upper left to lower right has same (row - column) value.

- Consider queen Q at [4,2] the square that are diagonal to this queen are [3,1], [5,3], [6,4], [7,5] & [8,6] all these square have same (row - column) value i.e 2.



- Also every element on diagonal that goes from upper right to lower left has same (row + column) value. For same Q [4,2] values are [5,1], [3,3], [2,4] [1,5] has same (row + col) value i.e 6.

- Suppose two queens are placed at position (i, j) and (k, l) then they are on the same diagonal if

$$i - j = k - l \quad \text{or} \quad i + j = k + l$$

since

- Therefore two queens lie on the same diagonal if and only if

$$|j - l| = |i - k|$$

- We use place (k, i) a boolean function that gives true values if it is possible to place k^{th} queen at i^{th}

Date / /
Page

column. otherwise returns false.
It also test both we whether 'i' is distinct from all previous values x[1] --- x[k-1] and whether there is no other queen on the same diagonal.

Algorithm Place(k, i)

// It Returns true if a queen can be
// placed at kth row & ith column otherwise
// returns false. x[] is global array
// whose first (k-1) values have been
// set. Abs(r) returns absolute value of r

```
Repeat
{
    for j ← 1 to k-1 do
    if ((x[j] = i) or // check for same column
     or (Abs(x[j] - i) = abs(j-k))) // check for
     then return false          same diagonal
    else
        return true
}
```

Algorithm N-Queens(k, n)

// Using backtracking this process prints
// all possible placements of n queens
// on nxn chessboard.

```
for i ← 1 to n do
{
    if place (k, i) then
    {
        x[k] ← i
        if (k = n) then print (x[1:n])
        else NQueens (k+1, n)
    }
}
```

Hamiltonian Cycle :-

- Let G(V, E) be connected graph with n vertices. A hamiltonian cycle is a round trip path along n edges of G that visits every vertex and returning to its starting position.

- If hamiltonian cycle begins at some vertex $v_1 \in G$ & the vertices of G are visited in the order $v_1, v_2, \ldots v_{n+1}$ then the edges (v_i, v_{i+1}) are in E, $1 \le i \le n$ the v_i are distinct except for v_1 & v_{n+1} which are equal.

- The graph G_1 from fig. (a) contain hamiltonian cycle 1, 2, 3, 4, 5, 6, 7, 8, 1 But graph G_2 for fig (b) does not contain hamiltonian cycle.

Sum of Subset –

Suppose we are given *n* distinct positive numbers (usually called weights) and we desire to find all combinations of these numbers whose sum is *M.* This is called the *sum of subsets* problem. We will consider a backtracking solution using the fixed tuple size strategy. In this case the element *X(i)* of the solution vector is either one or zero depending upon whether the weight *W(i)* is included or not.

Example 7.2 (Sum of subsets) Given $n + 1$ positive numbers: w_i, $1 \leq i \leq n$ and M, this problem calls for finding all subsets of the w_i whose sum is M. For example, if $n = 4$, $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$ and $M = 31$ then the desired subsets are (11, 13, 7) and (24, 7). Rather than represent the solution vector by the w_i which sum to M, we could represent the solution vector by giving the indices of these w_i. Now the two solutions are described by the vectors (1, 2, 4) and (3, 4). In general, all solutions are k-tuples $(x_1, x_2, \ldots, x_k)$, $1 \leq k \leq n$ and different solutions may have different size tuples. The explicit constraints require $x_i \in \{j | j$ is an integer and $1 \leq j \leq n\}$. The implicit constraints require that no two be the same and that the sum of the corresponding w_i be M. Since we wish to avoid generating multiple instances of the same subset (e.g. (1, 2, 4) and (1, 4, 2) represent the same subset), another implicit constraint which is imposed is that $x_i < x_{i+1}$, $1 \leq i < n$.

A simple choice for the bounding functions is $B_k(X(1), \ldots, X(k)) =$ true iff

$$\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \geq M$$

Clearly $X(1), \ldots, X(k)$ cannot lead to an answer node if this condition is not satisfied. The bounding functions may be strengthened if we assume the $W(i)$s are initially in nondecreasing order. In this case $X(1), \ldots, X(k)$ cannot lead to an answer node if

$$\sum_{i=1}^{k} W(i)X(i) + W(k + 1) > M$$

Example 7.6 Figure 7.9 shows the portion of the state space tree generated by procedure SUMOFSUB while working on the instance $n = 6$, $M = 30$ and $W(1:6) = (5, 10, 12, 13, 15, 18)$. The rectangular nodes list the values of s, k, r on each of the calls to SUMOFSUB. Circular nodes represent points at which a subset with sum M is printed out. At nodes A, B and C the output is respectively (1, 1, 0, 0, 1), (1, 0, 1, 1) and (0, 0, 1, 0, 0, 1). Note that the tree of Figure 7.9 contains only 23 rectangular nodes. The full state space tree for $n = 6$ contains $2^6 - 1 = 63$ nodes from which calls could be made (this count excludes the 64 leaf nodes as no call need be made from a leaf). □

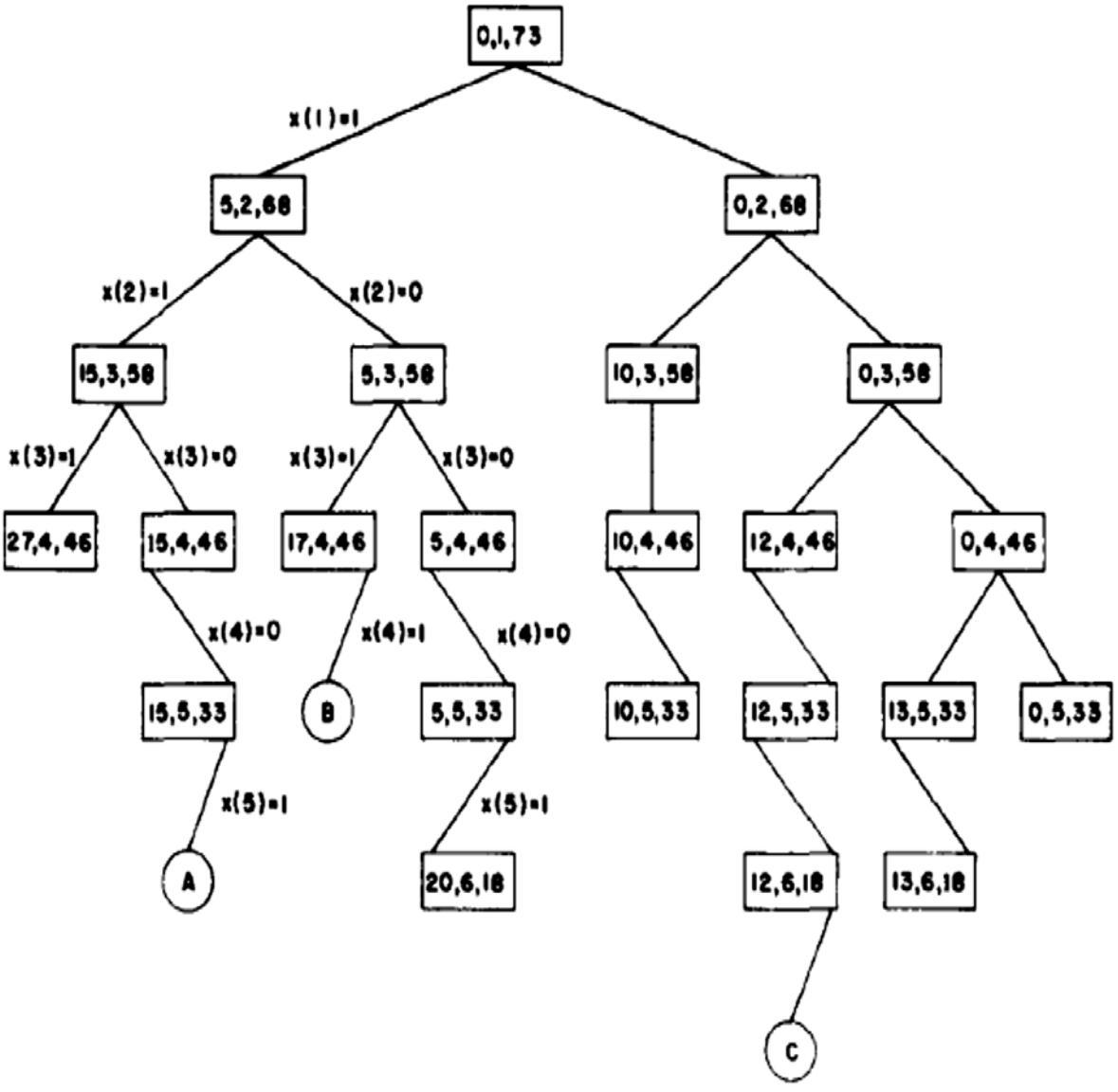


Figure 7.9 Portion of state space tree generated by SUMOFSUB

Date / /
Page

* Graph Coloring Problem :-

- Let G be the graph & m be the positive integer. We wanto color the graph G such that no two adjacent nodes have the same color.
- It is also called as m-coloring or m-colorability decision problem.
- If d is the degree of graph G then it can be color with maximum d+1 color
- The number of colors which are required color a graph is called as chromatic number.
- Consider the following graph can be colored with Red (R), Blue (B), Green (G) i.e 3 colors hence it's chromatic number is 3.

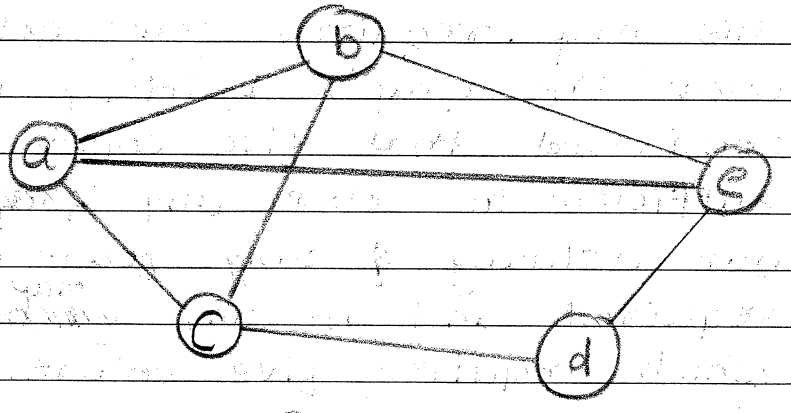
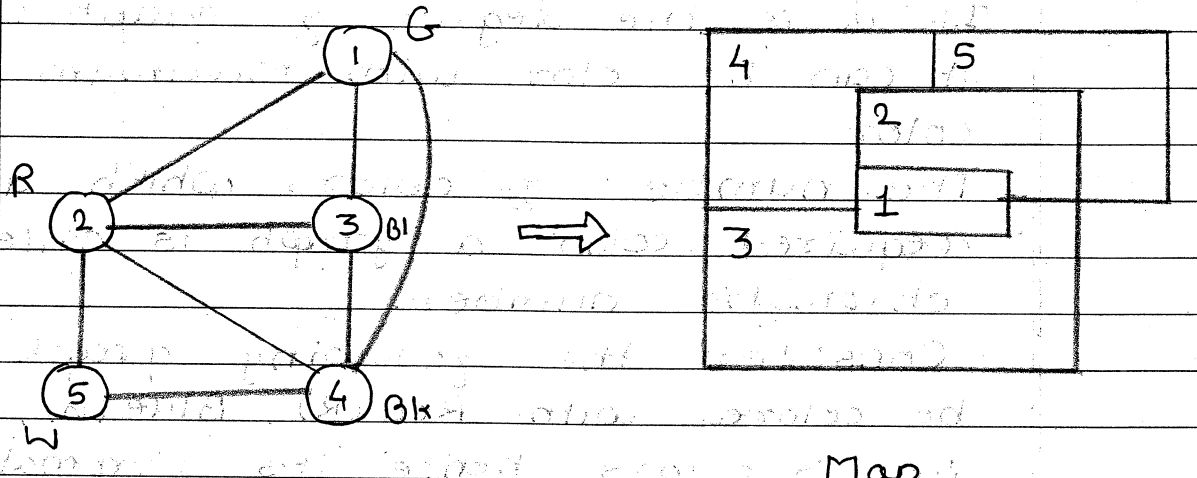


fig (a)

A graph is said to be planner iff it can be drawn in a plane in such a way that no two edges cross

each other. Such a planner graph can be transform to map. Each region of map becomes nodes if two regions are adjacent then corresponding nodes are join by an edge. Fig (b) shows a map with five regions and it's corresponding graph.



Graph (Planner) Map

- This map requires four colors to color the graph. Initially it is considered that five colors are sufficient to color any graph map
- But actually if only four colors are required and no any map is found which requires five colors.

- Suppose we represent a graph with adjacency matrix G[1:n, 1:n], where G[i,j] = 1 if (i,j) is an angle edge of G and G[i,j] = 0 otherwise.

(13)

Date / /
Page

- The colors represented by an integers 1, 2, ---m. & solution are given by n-tuples $(x_1, x_2, ---x_n)$ where x_i is color & node i.
- The function m-coloring is begun by first assigning the graph to its adjacency matrix.

Algorithm mcoloring(k) :-

```
// This is recurssive backtracking in which
// 1, 2, ... m are the integers (colors)
// which are assign to each vertex. k
// is index & next vertex.
{
  repeat
  {
    Nextvalue(k);
    if (x[k]=0) then return; // No new color
                                possible
    if (k=n) then // atmost m colors have
                     been used to color n verh
     write (x[1:n]);
     else mcoloring(k+1);
  } until(false);
}
```

(14)

Date / /
Page

bilt 10/10 Student Notebooks

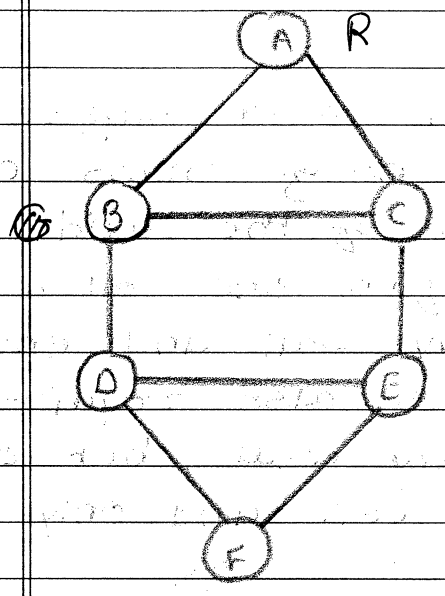
• Algorithm nextvalue(k)

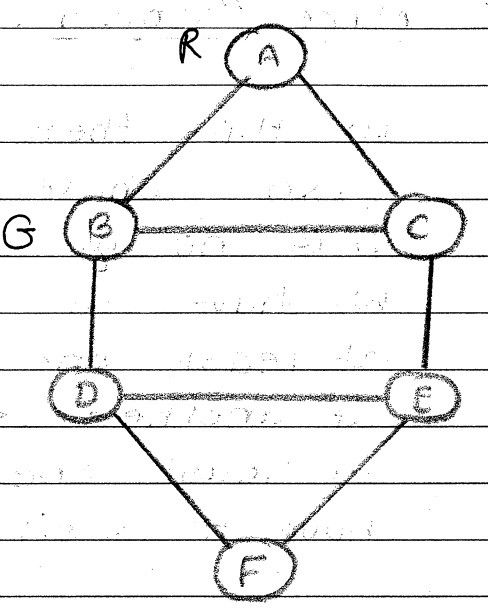
```
// x[1],...x[k-1] have been assigned integers
// in the range [1..m] such that adjacent
// vertex have distinct integers. A value
// for x[k] is determined in the range
// [0..m]. x[k] is assigned the next
// highest numbered color while maintaining
// distinctness from the adjacent vertices
// of vertex k. If no such color then x[k]=0
  {
    repeat
     {
      x[k]:=(x[k]+1) mod (m+1); // next color
      if(x[k]=0) then return; // all colors used
      for j:=1 to n do
       {
      //check distinctness of color
      if (G[k,j]≠0) and (x[k]=x[j])
      // if (k,j) is an edge & adj. vertices
      // have same color
        then break;
      }
      if (j=n+1) then return; // New color found
    } until (false); // otherwise try to find
                     // another node color.
}
```

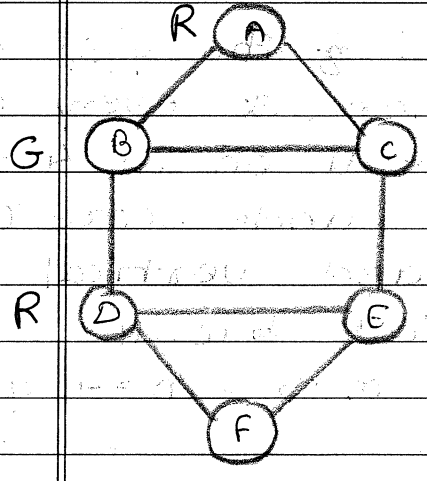
15

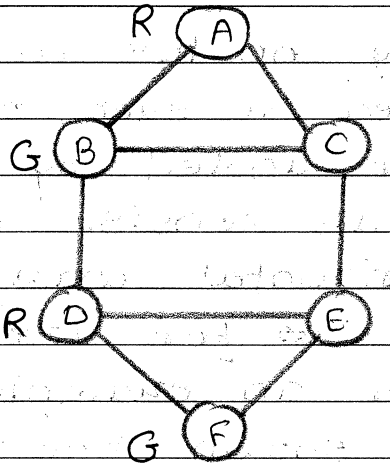
Date / /

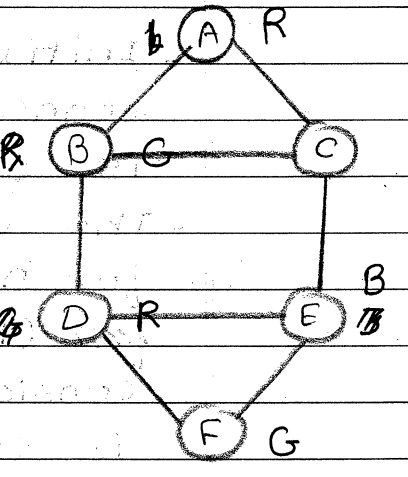
Page

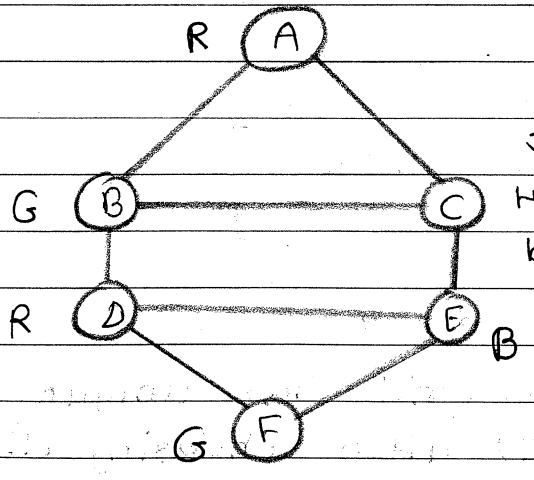




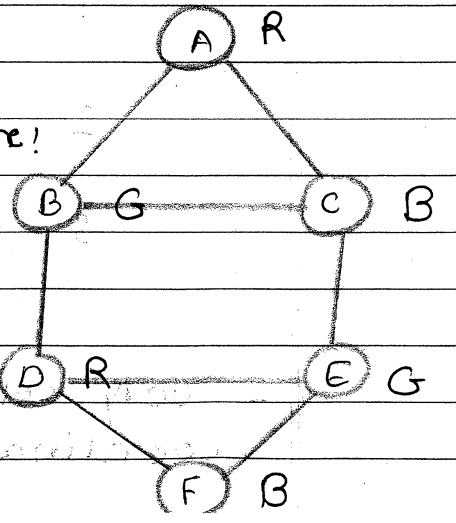








Stuck here!
Hence backtrack



Linear Programming

Optimization problems are common in the real world, with many applications in business and science that involve maximizing or minimizing some goal subject to a set of given constraints. For example, we might like to maximize a profit given a certain initial investment, or we might wish to minimize the error produced by a computer simulation using a fixed number of CPUs. Thus, in such contexts, there are in general two components to such optimization problems:

- a set of constraints that must be satisfied
- an objective function to maximize or minimize subject to the given constraints.

In a business application, for instance, the constraints might include the amount of risk allowed in an investment portfolio, and in a scientific application, the constraints might be determined by the number of CPUs available to run a simulation. In either case, often the objective is to maximize profit or minimize cost.

Example - suppose that a web server company wants to buy new servers to replace outdated ones and has two options to choose from. There is a standard model which costs $400, uses 300W of power, takes up two shelves of a server rack, and can handle 1000 hits/min. There is also a cutting-edge model, which costs $1600, uses 500W of power, but takes up only one shelf, and can handle 2000 hits/min. With a budget of $36,800, 44 shelves of server space and 12,200W of power, how many units of each model should the company purchase in order to maximize the number of hits it can serve every minute?

Let us introduce some variables, say $x1$ and $x2$, to represent the number of servers for each model. Then the number of hits per minute that can be serviced by $x1$ standard servers and $x2$ cutting-edge servers is

$$1000x1 + 2000x2.$$

Our goal is to maximize this quantity.

The number of servers the company should get is limited by three factors: the budget, which translates into

$$400x1 + 1600x2 \leq 36800,$$

the number of shelves that can be taken up by these servers,

$$2x_1 + x_2 \leq 44,$$

and the amount of power these servers can use collectively,

$$300x_1 + 500x_2 \leq 12200.$$

Therefore, this optimization problem can be summarized as follows:

$$\text{maximize: } z = 1000x_1 + 2000x_2$$

$$\text{subject to: } 400x_1 + 1600x_2 \leq 36800$$

$$2x_1 + x_2 \leq 44$$

$$300x_1 + 500x_2 \leq 12200$$

$$x_1, x_2 \geq 0,$$

where the inequalities in the last line express the implicit requirement that the number of each server has to be nonnegative. Such inequalities are necessary to prevent a nonsensical solution, and they are distinguished from the other constraints in that they determine the sign of the variables. Solving optimization problems that have the above general form is known as *linear programming*.

Linear Programming - Many problems take the form of maximizing or minimizing an objective, given limited resources and competing constraints. If we can specify the objective as a linear function of certain variables, and if we can specify the constraints on resources as equalities or inequalities on those variables, then we have a linear programming problem. Formally, a *linear-programming problem* is the problem of either minimizing or maximizing a linear function subject to a finite set of linear constraints. If we are to minimize, then we call the linear program a *minimization linear program*, and if we are to maximize, then we call the linear program a *maximization linear program*.

Translating Problems into Linear Programs

In general, there are three steps for turning an optimization problem into a linear program, assuming such a formulation is possible:

1. Determining the variables of the problem.

2. Finding the quantity to optimize, and write it in terms of the variables.

3. Finding all the constraints on the variables and writing equations or inequalities to express these constraints.

In addition, in defining the constraints, we need to be sure to include any implicit constraints describing the range of values the variables can take and to make sure all the equations are linear.

Standard Form

Recall that a function, f, is a ***linear function*** in the variables, $x_1, x_2, \ldots, x_n$, if it has the following form:

$$f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n = \sum_{i=1}^{n} a_ix_i,$$

for some real numbers, $a_1, a_2, \ldots, a_n$, which are called ***coefficients*** or ***weights***.

A ***linear program*** in ***standard form*** is an optimization problem with the following form:

$$\begin{aligned} &\text{maximize:} && z = \sum_{i \in V} c_i x_i \\ &\text{subject to:} && \sum_{j \in V} a_{ij} x_j \leq b_i \text{ for } i \in C \\ & && x_i \geq 0 \text{ for } i \in V \end{aligned}$$

where *V* indexes over the set of variables and *C* indexes over the set of constraints. The Xi 's are variables, whereas all other symbols represent fixed real numbers. The function to maximize is called the *objective function*, and the inequalities are called *constraints*. In particular, the inequalities *xi* ≥ 0 are called *nonnegativity constraints*.

For more details refer the links –

1. Graphical Method - https://www.youtube.com/watch?v=ku1KSgBfzs4
2. Simplex Method -1. https://www.youtube.com/watch?v=0PAOSZhwsnc
   2. https://www.youtube.com/watch?v=0cZKvD05i9U
3. Duality - https://www.youtube.com/watch?v=c1V6zbjAc5I
4. LP formulation - https://www.youtube.com/watch?v=fgoaGxzwLqI

The Geometry of Linear Programs

To understand a linear program, it helps to look at the problem from a geometric point of view. For simplicity, let us restrict ourselves to the two-dimensional case for the time being. In this case, each inequality constraint describes a half-plane, expressed as an infinite region to one side of a line. So a point that is inside all of these half-planes is a point that satisfies all the constraints. We call such a point a ***feasible solution***. Now intersections of half-planes have the shape of a convex polygon (Section 22.2). So the set of all feasible solutions is a convex polygon. We call this set the ***feasible region***. For example, Figure 26.1 shows the feasible region of the linear program from the web server example.

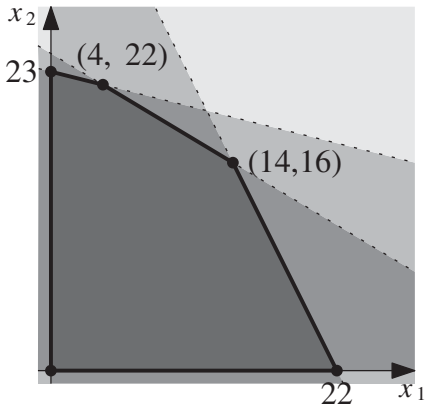


Figure 26.1: A feasible region is the intersection of half-spaces.

A region is ***convex*** if any two points in the region can be joined by a line segment entirely in the region (see Figure 26.2). Intuitively, if a two-dimensional shape is convex, then a person walking on its boundary, assuming it has one, would always be making left turns, or always right turns. (See, also, Section 22.2.)

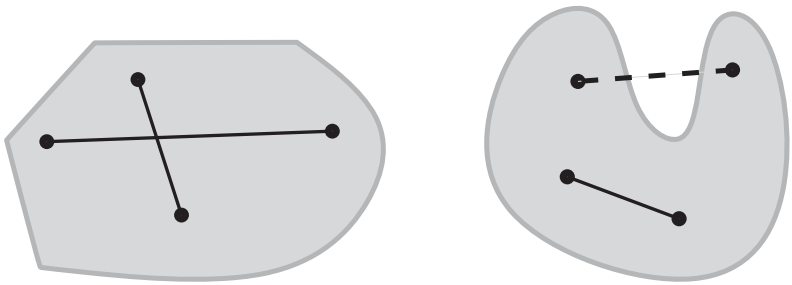


Figure 26.2: (a) In a convex set, any segment line joining two points inside the set is also inside the set. (b) In a non-convex set, there are segments joining two points in the set that are not entirely in the set.

In general, linear programs have a geometry in d dimensions where d is the number of variables. The geometric intuition is the same as in the plane but the shapes are more complex and the terminology is different. Inequality constraints in d dimensions are represented by half-spaces instead of half-planes, and their intersection forms a convex polytope instead of a convex polygon. For instance, a three-dimensional linear program could have a feasible region in the shape of a cube, pyramid, soccer ball, or any other three-dimensional convex shape with flat sides defining its boundary.

It is not enough to find just any feasible solution, of course. We are interested in one that optimizes the objective function. We refer to this feasible solution an ***optimal solution***. In the web server example, for instance, the set of points that produce a particular value, c, of the objective function is given by the equation

$$c = 1000x_1 + 2000x_2,$$

which is represented by a line. Such lines with varying values of c are all parallel, and we are interested in the one that maximizes c while still containing a feasible solution somewhere on the line. (See Figure 26.3.)

Referring to Figure 26.3, note that the slope of a line for the above objective function is always $-1/2$, regardless of the value of c. So we can imagine that, as c increases, the line sweeps the plane from the bottom left to the top right, staying parallel to itself. An optimal solution must be contained within the feasible region, which is represented by the gray region, and it must have the highest possible objective value. So the optimal solution is the point in the feasible region which is last hit by the sweeping line as it sweeps up and to the right. In this case, this point is the intersection of the lines representing the budget and power constraints,

$$400x_1 + 1600x_2 = 36800$$
$$300x_1 + 500x_2 = 12200.$$

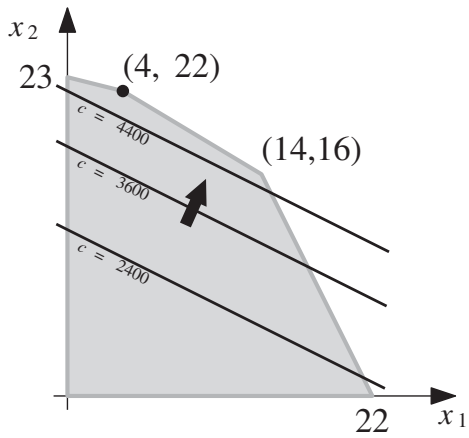


Figure 26.3: An example two-dimensional feasible region and objective function, $c = 1000x_1 + 2000x_2$. Values of the objective are represented by parallel lines.

By solving the above system of two equations in two variables, we see that the optimal solution occurs when $x_1 = 4$ and $x_2 = 22$. This example illustrates a principle that applies even to linear programs in higher dimensions—namely, that, because the feasible region is a convex polytope, an optimal solution, if one exists, always occurs on the boundary.

An optimal solution may not always be unique, however, such as when we modify the coefficients of the objective function in our example to

$$\text{maximize} \quad 1500x_1 + 2500x_2.$$

(See Figure 26.4(a).) It is also possible that the solution does not exist at all. For example, this case occurs when the feasible region is unbounded and the objective function tends to $+\infty$ as we move along a ray contained in the feasible region. (See Figure 26.4(b).)

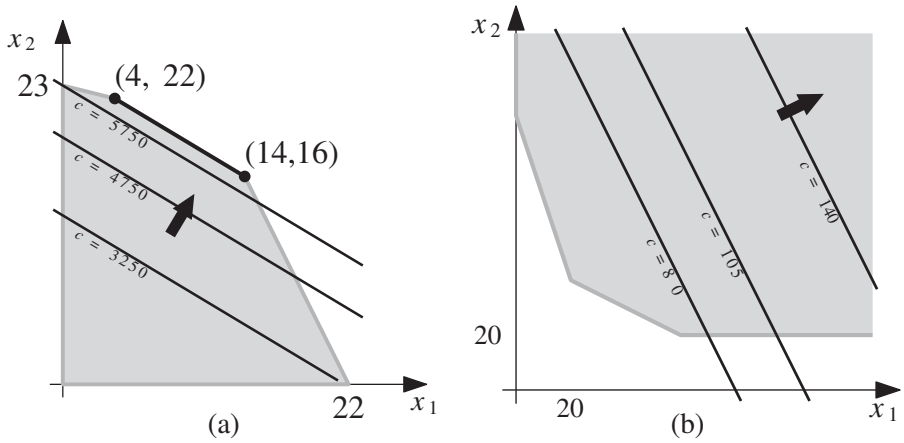


Figure 26.4: A linear program can have (a) many optimal solutions or (b) no optimal solution at all.

Finally, a linear program may not have an optimal solution simply because it has no feasible solution. This situation occurs when the constraints are so restrictive that no assignment to the variables can satisfy every constraint. Geometrically, this is the situation in which the intersection of all the half-spaces is empty.

Therefore, one algorithm to solve a linear program that has at least one optimal solution is to find all the vertices of the feasible region, and evaluate the objective function at these points. The optimal solution will be the point or points with highest objective value. This method is not particularly efficient, however, because there can be exponentially many vertices to evaluate. A better approach consists of starting at one vertex and, over several iterations, moving to a neighboring vertex with an increasingly better objective value. Thus, we can find a path on the boundary of the feasible region that starts at any vertex and ends at an optimal one. This alternative algorithm is called the simplex method, which is the algorithm we discuss next.

26.2 The Simplex Method

In this section, we describe the ***simplex method***, which is an algorithm for solving linear programs that follows a path through the vertices of the feasible region that increases the objective function in a greedy fashion. Although the worst-case runtime for this algorithm is exponential, in practice the algorithm usually finishes quickly.

26.2.1 Slack Form

To solve a linear program using the simplex method, we must first rewrite the problem in a format known as ***slack form***. To convert a linear program from standard form into slack form, we rewrite each of the inequality constraints as an equivalent equality constraint. This is done with the introduction of new variables, called ***slack variables***, which are nonnegative and measure the difference in the original inequality. For example, to rewrite the inequality $2x - 5y \leq 28$ in slack form, we could introduce a slack variable, s, with the constraints, $s = 28 - (2x - 5y)$ and $s \geq 0$. Intuitively, variable s measures the "slack" in the inequality, that is, the amount between the lesser and greater quantities in the inequality. We perform this step for each inequality in the standard form, introducing a slack variable for each such inequality.

Formally, we say that linear program is in ***slack form*** if we seek to maximize a linear objective, z, subject to constraints that are either equality constraints involving a slack variable or are nonnegativity constraints, as follows:

$$\begin{aligned} \text{maximize:} \quad & z = c_* + \sum_{j \in F} c_j x_j \\ \text{subject to:} \quad & x_i = b_i - \sum_{j \in F} a_{ij} x_j, \quad \text{for } i \in B \\ & x_i \geq 0 \text{ for } 1 \leq i \leq m + n. \end{aligned}$$

The sets B and F partition the x_i variables into ***basic variables*** and ***free variables***, respectively. That is, each equality constraint has a basic (slack) variable on the left-hand side and only free variables on the right-hand side. Thus, free variables only appear on the left-hand side of nonnegativity constraints. Taken together, the a_{ij} coefficients form a $m \times n$ matrix, A, where $n = |F|$ is the number of variables in the standard form, and $m = |B|$ is the number of constraints in the standard form. Incidentally, the minus sign in the equality constraints is needed so that the matrix A is the same in the slack form and standard form.

Graphical Method

Solve the following LPP by Graphical Method

Minimize $Z = 20x_1 + 10x_2$

Subject to $x_1 + 2x_2 \leq 40$

$3x_2 + x_2 \geq 30$

$4x_1 + 3x_2 \geq 60$ $\quad x_1, x_2 \geq 0$

⇒ Replace all inequality constraints by equation

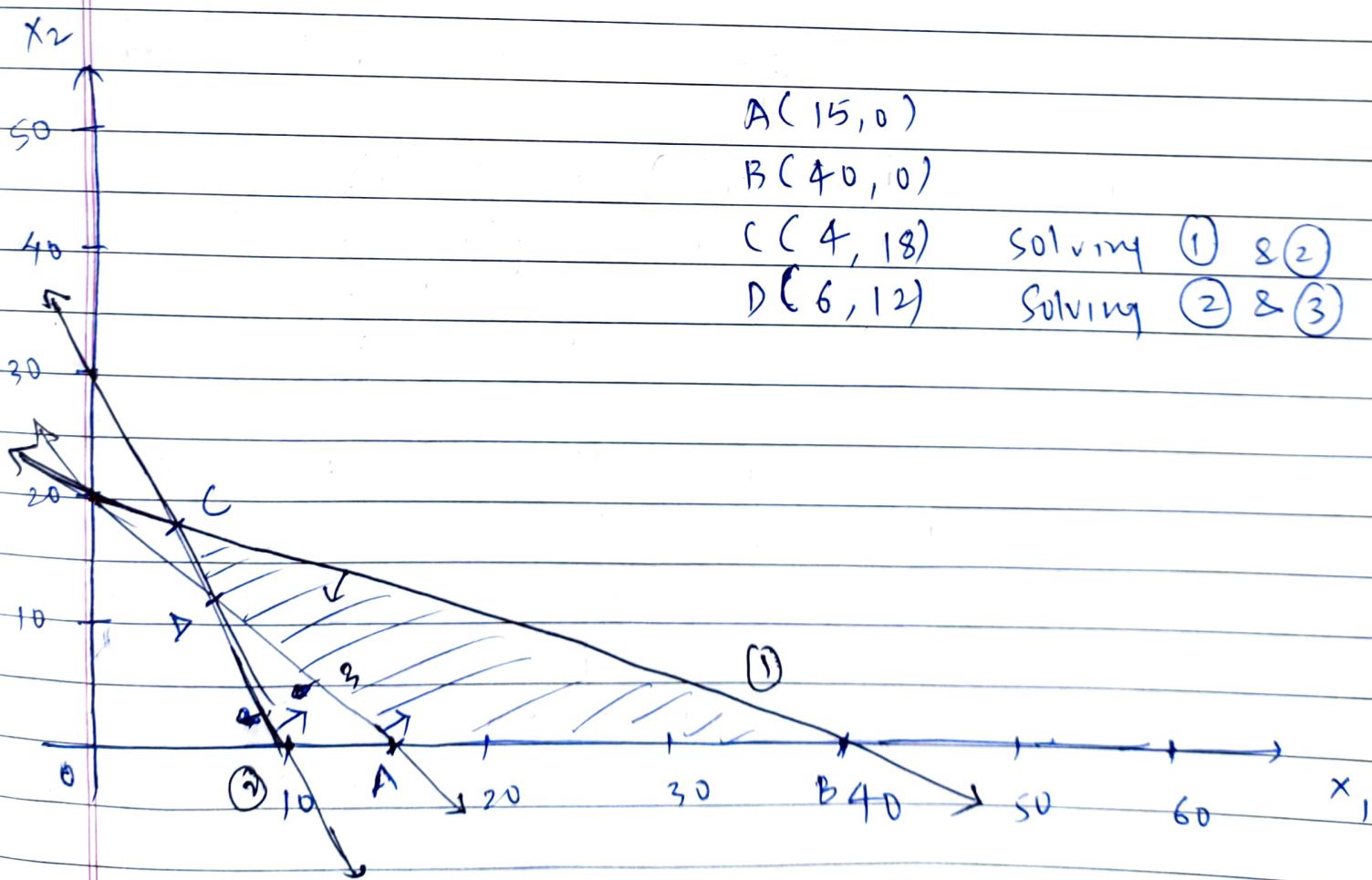
$x_1 + 2x_2 = 40 \quad -(1)$

$3x_1 + x_2 = 30 \quad -(2)$

$4x_1 + 3x_2 = 60 \quad -(3)$

$\div$ eq (1) by 40 $\quad \div$ eq 2 by 30 $\quad \div$ eqn 3 by 60

$\frac{x_1}{40} + \frac{x_2}{20} = 1 \quad \frac{3x_1}{10} + \frac{x_2}{30} = 1 \quad \frac{x_1}{15} + \frac{x_2}{20} = 1$



A(15, 0)

B(40, 0)

C(4, 18) Solving (1) & (2)

D(6, 12) Solving (2) & (3)

Minimize $Z = 20x_1 + 10x_2$

$Z_A = 15 \times 20 + 10 \times 0 = 300$

$Z_B = 20 \times 40 + 10 \times 0 = 800$

$Z_C = 20 \times 4 + 10 \times 18 = 260$

$Z_D = 20 \times 6 + 10 \times 12 = 240$

Simplex Method

Max. $z = 3x_1 + 2x_2 + 5x_3$

Subject to

$$x_1 + 2x_2 + x_3 \leq 430$$
$$3x_1 + 2x_3 \leq 460$$
$$x_1 + 4x_2 \leq 420$$
$$x_1, x_2, x_3 \geq 0$$

General form

Inequality

We have 3 variables & 3 equations (

By Introducing slack variables s_1, s_2, s_3 convert the problem to standard form (convert inequalities to equalities)

(1)

Max. $z = 3x_1 + 2x_2 + 5x_3$

Subject to

$$x_1 + 2x_2 + x_3 + s_1 = 430$$
$$3x_2 + 2x_3 + s_2 = 460$$
$$x_1 + 4x_2 + s_3 = 420$$
$$x_1, x_2, x_3, s_1, s_2, s_3 \geq 0$$
$$z = 3x_1 + 2x_2 + 5x_3 + 0s_1 + 0s_2 + 0s_3$$

Standard form

(2) Initial basic feasible solution

$x_1 = x_2 = x_3 = 0$ $\therefore$ $s_1 = 430$ $s_2 = 460$, $s_3 = 420$

(3) Write the standard form LPP into matrix form.

$$AX = B$$

$$\begin{bmatrix} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 \\ 1 & 2 & 1 & 1 & 0 & 0 \\ 3 & 0 & 2 & 0 & 1 & 0 \\ 1 & 4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 430 \\ 460 \\ 420 \end{bmatrix}$$

(4) Construct Initial simplex table

Objective fun'

	← C_j		3	2	5	0	0	0
C_B	B (Basic variable)	X_B RHS	x_1	x_2	x_3	s_1	s_2	s_3
0	s_1	430	1	2	1	1	0	0
0	s_2	460	3	0	2	0	1	0
0	s_3	420	1	4	0	0	0	1

(5) Calculate $Z_j - C_j$

$$Z_j - C_j = C_B \cdot X_j - C_j$$

$$Z_1 - C_1 = C_B \cdot X_1 - C_1$$

$$= (0.1 + 0.3 + 0.1) - 3 = -3$$

$$Z_2 - C_2 = (0.2 + 0.0 + 0.4) - 2 = -2$$

$$Z_3 - C_3 = (0.1 + 0.2 + 0.0) - 5 = -5$$

Date
Page

incoming vector

Min. ratio	C_j		3	2	5	0	0	0
430/1 = 430	C_B	B	X_B	X_1	X_2	X_3	S_1	S_2	S_3
460/2 = 230	0	S_1	430	1	2	1	1	0	0
	0	S_2	460	3	0	2	0	1	0
	0	S_3	420	1	4	0	0	0	1
				-3	-2	-5			

Key element

Key column

Outgoing vector

① If all $(Z_j - C_j) \geq 0$ the optimal solution will be obtained

② If atleast one $(Z_j - C_j) < 0$ then indicate by an arrow and this column is called key column

③ If more than one $(Z_j - C_j) < 0$ then choose the most "-" ve value & that becomes the key column.

Basic variable represented by this column is called incoming vector. (X_3)

⑥ Calculate min. ratio

$$\text{Min. ratio} = \frac{X_B}{C_K} \qquad C_K = \text{Key column}, > 0$$

⑦ Construct new simplex table by entering incoming vector.

Replace outgoing vector by incoming vector

Min ratio x_B/x_2			C_j	3	2	5	0	0	0
	C_B	B	x_B	x_1	x_2	x_3	s_1	s_2	s_3
100	0	s_1	200	$-1/2$	2	0	1	$-1/2$	0
—	5	x_3	230	$3/2$	0	1	0	$1/2$	0
105	0	s_3	420	1	4	0	0	0	1
		$Z_j - C_j$		$9/2$	-2	0	0	$5/2$	0

outgoing vector; key element; incoming vector; key column

* The row which contains key element (2) row 2 ; divide the row with key element

(Matrix of Basic variables should be Identity matrix)

① $\frac{430 \times 2 - 460 \times 1}{2} = \frac{860 - 460}{2} = \frac{400}{2} = 200$

② $\frac{420 \times 2 - 0 \times 460}{2} = \frac{840}{2} = 420$

③ $\frac{1 \times 2 - 3 \times 1}{2} = \frac{2-3}{2} = -1/2$

④ $\frac{1 \times 2 - 3 \times 0}{2} = \frac{2-0}{2} = 1$

⑤ $\frac{2 \times 2 - 1 \times 0}{2} = \frac{4}{2} = 2$

⑥ $\frac{4 \times 2 - 0 \times 0}{2} = \frac{8}{2} = 4$

(7) $\frac{0 \times 2 - 1 \times 1}{2} = \frac{-1}{2}$

(8) $\frac{0 \times 2 - 0 \times 0}{2} = 0$

Calculating $Z_j - C_j = C_B \cdot X_j - C_j$

$$Z_1 - C_1 = \left(0 \times \frac{-1}{2} + 5 \times \frac{3}{2} + 0 \times 1\right) - 3$$

$$= \frac{15}{2} - 3 = \frac{9}{2}$$

$$Z_2 - C_2 = (0 \times 2 + 5 \times 0 + 0 \times 4) - 2$$

$$= -2$$

$$Z_3 - C_3 = (0 \times 0 + 5 \times 1 + 0 \times 0) - 5$$

$$= 0$$

		C_j	3	2	5	0	0	0
C_B	B	X_B	X_1	X_2	X_3	S_1	S_2	S_3
2	X_2	100	-1/4	1	0	1/2	-1/4	0
5	X_3	230	3/2	0	1	0	1/2	0
0	S_3	20 ← ①	2	0	0	-2	1	1
	$Z_j - C_j$		4 ②	0	0	1	2	0
$R_3' = R_3 - 4R_1$		R_3: 420	1	4		0	0	0	1
①		$4R_1$: 400	-1	4		0	2	-1	0
			20	2	0	0	-2	1	1

All values of $Z_j - C_j \geq 0$.
So solution has reached optimal solution

$\therefore x_2 = 100$
$x_3 = 230 \quad x_1 = 0$

Max $Z = C_B \cdot X_B = (2 \times 100 \times 5 \times 230 + 0 x_2$

$= 1350$

Use simplex method to solve LPP

Min $z = x_1 - 3x_2 + 2x_3$

Subject to

$$3x_1 - x_2 + 2x_3 \leq 7$$
$$-2x_1 + 4x_2 \leq 12$$
$$-4x_1 + 3x_2 + 8x_3 \leq 10$$
$$x_1, x_2, x_3 \geq 0$$

⇒ By introducing slack variables s_1, s_2, s_3 convert the problem in standard form

Multiplying z by (-1)

Max $z = -x_1 + 3x_2 - 2x_3 + 0s_1 + 0s_2 + 0s_3$

Subject to

$$3x_1 - x_2 + 2x_3 + s_1 = 7$$
$$-2x_1 + 4x_2 + s_2 = 12$$
$$-4x_1 + 3x_2 + 8x_3 + s_3 = 10$$

An initial basic feasible solution is

$x_1 = x_2 = x_3 = 0 \quad s_1 = 7 \quad s_2 = 12, \quad s_3 = 10$

Writing in Matrix form $AX = B$

$$\begin{matrix} x_1 & x_2 & x_3 & s_1 & s_2 & s_3 \end{matrix}$$
$$\begin{bmatrix} 3 & -1 & 2 & 1 & 0 & 0 \\ -2 & 4 & 0 & 0 & 1 & 0 \\ -4 & 3 & 8 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 12 \\ 10 \end{bmatrix}$$

Construct Initial Simplex table

outgoing vector ← | incoming vector ↓

	C_j	-1	3	-2	0	0	0	Min x_B/x_2
C_B	B	x_B	x_1	x_2	x_3	s_1	s_2	s_3	
0	s_1	7	3	-1	2	1	0	0	-
0	s_2	12	-2	4	0	0	1	0	3
0	s_3	10	-4	3	8	0	0	1	3.33
	$Z_j - C_j$		-1	-3	-2	0	0	0	

$Z_j - C_j$ → $C_B \cdot x_j - C_j$

Key element

outgoing vector ← | incoming vector ↓ | Key column

	C_j	-1	3	-2	0	0	0	Min
B	x_B	x_1	x_2	x_3	s_1	s_2	s_3	
s_1	10	5/2	0	2	1	1/4	0	4
x_2	3	-1/2	1	0	0	1/4	0	-6
s_3	1	-5/2	0	8	0	-3/4	1	-2/5
$Z_j - C_j$		① -1/2	0 ②	2 ③	0 ④	3/4 ⑤	0 ⑥	

$R_2' = 12_2/4$

$R_1' = R_1 + R_2'$

7	3	-1	2	1	0	0
3	-1/2	1	0	0	1/4	0
10	5/2	0	2	1	1/4	0

$R_3' = R_3 - 3R_2'$

10	-4	3	8	0	0	1
9	-3/2	3	0	0	3/4	0
1	-5/2	0	8	0	-3/4	1

C_j	B	x_B	x_1	x_2	x_3	s_1	s_2	s_3
-1	x_1	4	1	0	4/5	2/5	1/10	0
3	x_2	5	0	1	2/5	3/10	-3/10	0
0	s_3	11	0	0	10	1	-1/2	1
	$z_j - c_j$		0	0	12/5	1/5	4/5	0

$$R_2' = R_2 + \frac{1}{2} R_1'$$

R_2	3	$\frac{-1}{2}$	1	0	0	$\frac{1}{4}$	0
$\frac{1}{2}R_1'$	2	$\frac{1}{2}$	0	$\frac{2}{5}$	$\frac{2}{10}$	$\frac{1}{20}$	0
	5	0	1	$\frac{2}{5}$	$\frac{2}{10}$	$\frac{3}{10}$	0

$$R_3' = R_3 + \frac{5}{2} R_1$$

1	$\frac{-5}{2}$	0	8	0	$\frac{-3}{4}$	1
10	$\frac{5}{2}$	0	2	1	$\frac{1}{4}$	0
11	0	0	10	1	-1/2	1

Don't consider -'ve value

All $z_j - c_j \geq 0$ $\therefore$ we got optimal soln

$x_1 = 4$ $x_2 = 5$ $x_3 = 0$

Max $z' = -1x_4 + 3x_5 + 0x_1$

$= 11$

Min $z^* = -(\text{Max } z') = -11$

Min $Z = -\text{Max } Z' = -11$

$x_1 = 4 \quad x_2 = 5 \quad x_3 = 0$

⇓

DO not invert values of x_1, x_2, x_3

Primal to Dual Conversion

Max $Z = x_1 + 2x_2 + x_3$

Subject to

$$2x_1 + x_2 - x_3 \leq 2$$
$$-2x_1 + x_2 - 5x_3 \geq -6 \quad -(2)$$
$$4x_1 + x_2 + x_3 \leq 6$$
$$x_1, x_2, x_3 \geq 0$$

(1) First we convert the problem in canonical form

Min → Min $\geq$

Max → Max $\leq$

Canonical form.

Max $Z = x_1 + 2x_2 + x_3$

Subject to

$$2x_1 + x_2 - x_2 \leq 2$$

$-1 \times$ (2)

$$2x_1 - x_2 + 5x_3 \leq 6$$
$$4x_1 + x_2 + x_3 \leq 6$$
$$x_1, x_2, x_3 \geq 0$$

Date
Page

② Change the objective function of maximization in the primal into minimization in the dual & vice versa

Max z → Min z $\geq$

Min z → Max z $\leq$

~~Dual form~~

~~Min z~~ =

③ The number of variable in the primal will be the number of constraints in dual & vice versa

④ Cost coefficient in objective function of the primal will be RHS constant of the constraints in dual and vice versa

Dual form

Let w_1, w_2, w_3 be 3 variables

Min $z = 2w_1 + 6w_2 + 6w_3$

Subject to

$2w_1 + 2w_2 + 4w_3 \geq 1$

$w_1 - w_2 + w_3 \geq 2$

$-w_1 + 5w_2 + w_3 \geq 1$

$$\begin{bmatrix} 2 & 1 & -1 \\ 2 & -1 & 5 \\ 4 & 1 & 1 \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} 2 & 2 & 4 \\ 1 & -1 & 1 \\ -1 & 5 & 1 \end{bmatrix}$$

⑤ For formulating constraints we consider the transpose of the matrix

Primal	Canonical	Dual
Min	Min $\geq$	Max $\leq$
Max	Max $\leq$	Min $\geq$

(2) Max $z = x_1 - x_2 + 3x_3$

Subject to :

$$x_1 + x_2 + x_3 \leq 10$$
$$2x_1 - x_3 \leq 2$$
$$2x_1 - 2x_2 + 3x_3 \leq 6$$
$$x_1, x_2, x_3 \geq 0$$

(1) Canonical form

Max $z = x_1 - x_2 + 3x_3$

Subject to

$$x_1 + x_2 + x_3 \leq 10$$
$$2x_1 - x_3 \leq 2$$
$$2x_1 - 2x_2 + 3x_3 \leq 6$$
$$x_1, x_2, x_3 \geq 0$$

(2) Dual form

Min $z = 10w_1 + 2w_2 + 6w_3$

subject to

$$w_1 + 2w_2 + 2w_3 \geq 1$$
$$w_1 - w_2 + 3w_3 \geq 3$$
$$w_1 - 2w_3 \geq -1$$
$$w_1, w_2, w_3 \geq 0$$

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

$$\Downarrow$$

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

Min $z = 2x_2 + 5x_3$

subject to

$$x_1 + x_2 \geq 2$$
$$2x_1 + x_2 + 6x_3 \leq 6$$
$$x_1 - x_2 + 3x_3 = 4$$

x_1, x_2, x_3

$x_1 - x_2 + 3x_3 \leq$

$x_1 - x_2 + 3x_3 \geq$

⇒

① Canonical form

Min $z = 2x_2 + 5x_3$

Subject to

$$x_1 + x_2 \geq 2$$

$$-2x_1 - x_2 - 6x_3 \geq -6$$

$$x_1 - x_2 + 3x_3 \geq 4$$

$$-x_1 + x_2 - 3x_3 \geq -4$$

Dual form: Let w_1, w_2, w_3, w_4 be dual var

Max

$$z = 2w_1 - 6w_2 + 4w_3 - 4w_4$$

Subject to

$$w_1 + 2w_2 + w_3 - w_4 \leq 0$$
$$w_1 - w_2 - w_3 + w_4 \leq 2$$
$$-6w_2 + 3w_3 - 3w_4 \leq 5$$

$$w_1, w_2, w_3, w_4 \geq 0$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & -1 & -6 \\ 1 & -1 & 3 \\ -1 & 1 & -3 \end{bmatrix}$$

Solve by dual Simplex Method

Min $z = 5x_1 + 6x_2$

Subject to

$$x_1 + x_2 \geq 2$$
$$4x_1 + x_2 \geq 4$$
$$x_1, x_2 \geq 0$$

Max $z = -5x_1 - 6x_2$

Subject to

$$-x_1 - x_2 \leq -2$$
$$4x_1 - x_2 \leq -4$$
$$x_1, x_2 \geq 0$$

Converting inequalities to eqn

Max $z = -5x_1 - 6x_2$

Subject to

$$-x_1 - x_2 + s_1 = -2$$
$$-4x_1 - x_2 + s_2 = -4$$

An initial feasible solution is given by

$x_1, x_2 = 0$ $\quad s_1 = -2$ $\quad s_2 = -4$

		Cj	-5	-6	0	0
C_B	B	X_B	x_1	x_2	s_1	s_2
0	s_1	-2	-1	-1	1	0
0	s_2	-4	-4	-1	0	1
Zj-Cj			5	6	0	0
Max. ratio			-5/4	-6		

incoming vector

outgoing vector

key value

Max. ratio = $\frac{Z_j - C_j}{\text{key row}}$ (< 0)

Max ratio:

			0	-6.33		-5
			-5	-6	0	0
C_B	B	X_B	x_1	x_2	S_1	S_2
0	S_1	-1	0	-3/4	1	-1/4
-5	x_1	1	1	1/4	0	-1/4
		5	0	19/4	0	5/4

① $\div R_2$ by -4

② $R_1' = R_1 + R_2'$

			-2	-11	-1	1	0
			1	1	1/4	0	-1/4
			-1	0	-3/4	1	-1/4

S_2 → incoming vector S_1 → outgoing vector

			-5	-6	0	0
C_B	B	X_B	x_1	x_2	S_1	S_2
0	S_2	4	0	3	-4	1
-5	x_1	2	1	1	-1	0
	$Z_j - C_j$		0	1	5	0

$R_2' = R_2 + \frac{1}{4} R_1'$

$x_1 = 2 \quad x_2 = 0$

Max. $Z = 0 \times 4 + (-5) \times 2 = -10$

$\therefore$ [redacted] = [redacted]

Original $Z = -\text{Max} = \underline{10}$

$x_1 = 2, \; x_2 = 0$