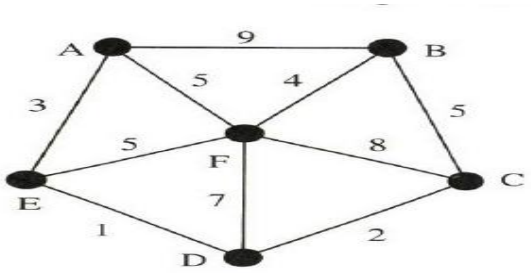
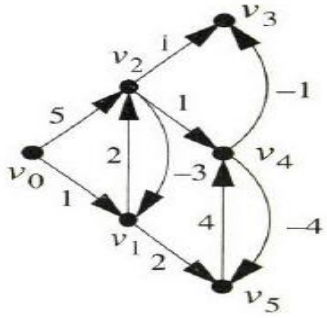
USN | | | | | | | | | | |

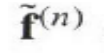
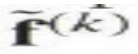
Internal Assessment Test 2– November 2020

Sub:	CAD of Digital Systems					Sub Code:	18EVE31	Branch:	M.Tech
Date:	12/11/2020	Duration:	90 min's	Max Marks:	50	Sem / Sec:	III, VLSI		OBE

Answer any FIVE FULL Questions

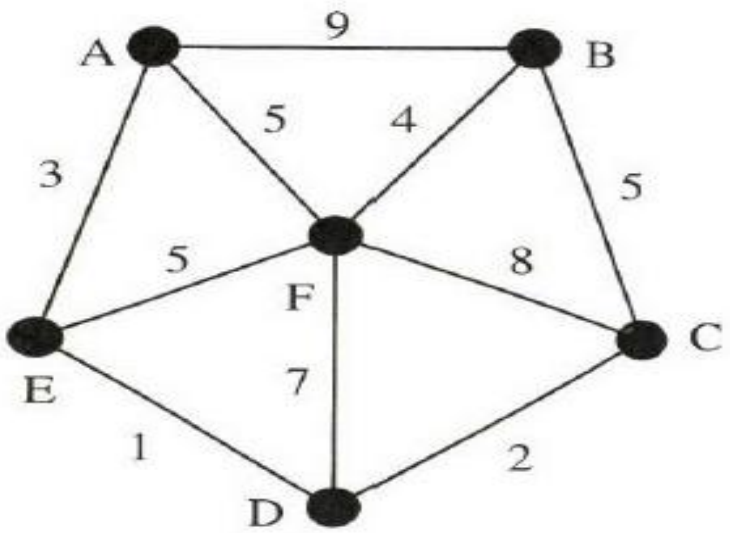
	MARKS	CO	RBT
1. Calculate the least cost solution for the travelling salesman problem shown in figure below using **backtracking algorithm**. Show the least – cost path. Draw the search tree and explain. 	[10]	CO2	L3
2. Describe briefly VLSI design problem formulation with respect to compaction, informal and graph theoretical formulation and maximum distance constraints with diagrams.	[10]	CO2	L1
3. (a)Write short notes on minimum distance rules.	[06]	CO2	L2
(b)Discuss briefly about applications of compaction	[04]	CO2	L2
4. Explain how a genetic algorithm is used for search and solution of a combinatorial Optimization problem.	[10]	CO2	L3
5. Explain the applications of the Bellman Ford algorithm with pseudo-code to the directed graph shown below in figure 1:  Figure 1 – Directed Cyclic Graph	[10]	CO2	L3
6. Write and explain the pseudo-code for simulated annealing. Detail the application in VLSI design with pros and cons.	[10]	CO2	L2
7. Explain the principle of **Branch and bound algorithm** for finding the optimal solution of a combinatorial optimization problem.	[10]	CO2	L1

CMR Institute of Technology
Department of ECE
M.Tech III Sem VLSI Design & Embedded Systems
IAT -2 Scheme and Solution
18EVE31- CAD of Digital Systems

1. **Calculate the least cost solution for the travelling salesman problem shown in figure below using backtracking algorithm. Show the least – cost path. Draw the search tree and explain.**
   - Principle of using backtracking for an exhaustive search of the solution space- start with an initial partial solution in which as many variables as possible are left unspecified
   - Then systematically assign values to the unspecified variables until either a single point in the search space is identified or an implicit constraint makes it impossible to process more unspecified variables
   - Both cases- the algorithm continues by going back to a partial solution generated earlier and then assigning a next value to an unspecified variable (hence the name "backtracking").
   - Pseudo-Code
   - Assumption- all variables f_i have type ***solution-element***.
   - The partial solutions are generated in
   - such a way that the variables f_i are specified for $1 \leq i \leq k$ and are unspecified for $i > k$
   - Partial solutions having this structure will be denoted by 
   -  corresponds to a fully-specified solution
   - The global array ***val*** corresponds to the vector 
   - The value of f_k, is stored in ***va1[k – 1].***
   - The procedure ***cost(val)*** is supposed to compute the cost of a feasible solution using the cost function ***c***. It is obviously only called when ***k= n***, i.e. when a solution is fully specified.
   - Procedure ***allowed(val, k)*** returns a set of values allowed by the explicit and implicit constraints for the variable f_{k+1}, given 
   - The best solution found is stored in the global array ***best-solution*** and it is reported at the end of the search process.
   - Consider the graph version of the traveling salesman problem (TSP)
   - The number of vertices in a graph G(V, E) equals ***n – 1***
   - A solution can be specified by a sequence of ***n***, vertices such that an edge between subsequent vertices exists, the first vertex in the sequence equals the last one and the first ***n - 1*** vertices are distinct.
   - The sequence can be characterized by ***n*** variables f_i ($1 \leq i \leq n$), one for each position in the sequence
   - The value of f_i is the vertex at position ***i*** in the sequence

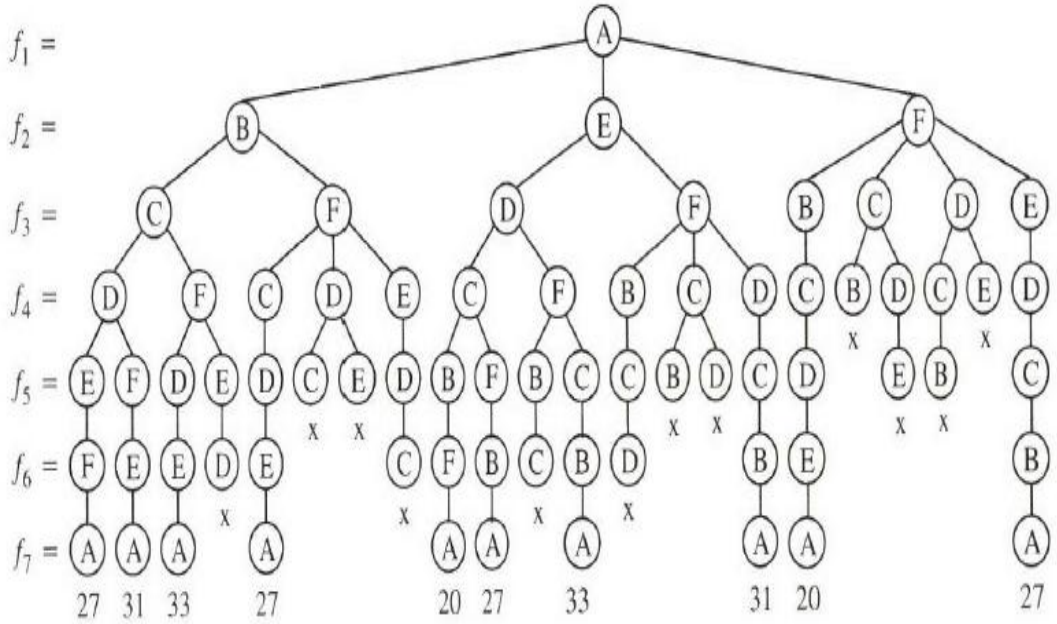
- one can select any of the vertices in the graph and state that f_1 and f_n can only have value v.
- These are the explicit constraints for f_1 and f_n
- The implicit constraints are that any sequence specified by the variables should form a path

```
float best_cost;
solution_element val[n], best_solution[n];

backtrack(int k)
{
  float new_cost;
  if (k = n) {
    new_cost := cost(val);
    if (new_cost < best_cost) {
      best_cost := new_cost;
      best_solution := copy(val);
    }
  }
  else
    for each (el ∈ allowed(val, k)) {
      val[k] := el;
      backtrack(k + 1);
    }
}

main ()
{
  best_cost := ∞;
  backtrack(0);
  report(best_solution);
}
```



Example- Travelling Salesman Problem (TSP)

The six vertices have been labeled A to F. The cost of traveling between two adjacent vertices is given by a weight along the edge connecting the two vertices. It will be assumed that $f_1 = f_7 = A$ and that the other variables take any of the values in the set $\{B, C, D, E, F\}$.



Search tree obtained by the exhaustive search backtracking algorithm for the TSP Problem

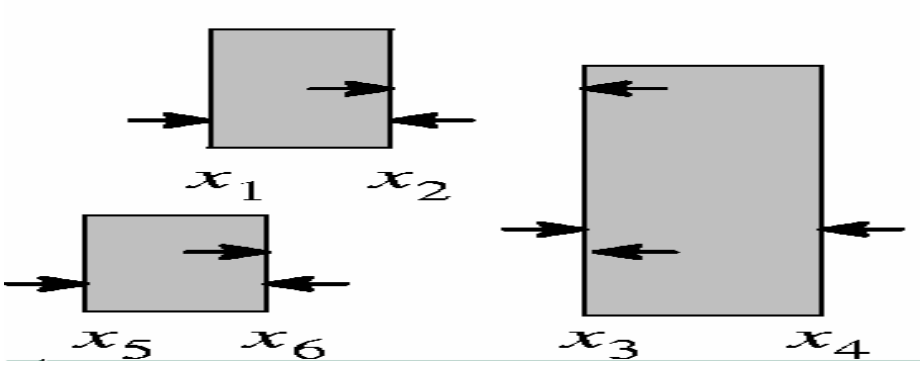
- Backtracking algorithm can be visualized by nodes in a tree, called the search tree
- The search tree for the given example is shown in figure
- A tree is a directed graph here
- It is normally drawn in such a way that all edges have an implicit direction from top to bottom.
- The single node at the highest level, Level 1, is called the root of the tree
- The nodes incident from the node (connected to it at the next lower level) are called its children
- Every level of the tree corresponds to a variable. The children at level ***k +1*** of a node at level ***k*** correspond to the partial solutions obtained by specifying a value for f_{k+1} ,
- So a node at level ***k*** represents a partial solution
- Nodes without children are called leaf nodes
- Each node except for the root has exactly one incoming edge. The other endpoint of this edge is called the node's parent.
- Removing this incoming edge for a node gives a subtree having as its root
- Each path in the tree starting at the root and ending in a leaf node corresponds to either a fully specified solution of which the cost can be computed (indicated with the cost value in the figure) or to a state in the backtracking procedure from which no new (partial) solution can be generated due to the fact that no legal assignment to the next variable is possible (indicated with an 'x' in the figure).
- The optimal tour visits the vertices in the order A, E, D, C, B, F, and A (or in the reverse order) and has a total length of 20

2. **Describe briefly VLSI design problem formulation with respect to compaction, informal and graph theoretical formulation and maximum distance constraints with diagrams**.

In one-dimensional, say horizontal, compaction a rigid rectangle can be represented by one x-coordinate (of its center, for example) and a stretchable one by two (one for each of the endpoints)

Assumption – **n** distinct ***x-coordinates indicated as*** $x_1, x_2, \ldots x_n$

A minimum-distance design rule between two rectangle edges can now be expressed as an inequality $x_j - x_i \geq d_{ij}$.



The pairs of variables associated with horizontally stretchable rectangles

- For example, if the minimum width is ***a*** *and* the minimum separation is ***b***, *then*

$$x_2 - x_1 \geq a$$
$$x_3 - x_2 \geq b$$
$$x_3 - x_6 \geq b$$

Constraint Graph

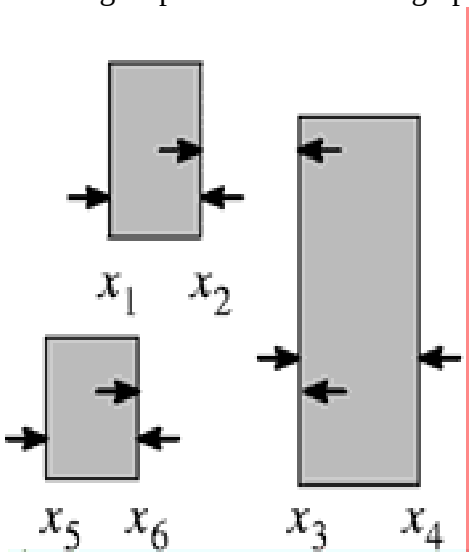
- The inequalities can be used to construct a constraint graph *G(V, E):*

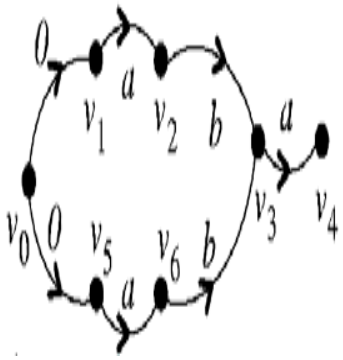
(i) There is a vertex v_i *for each variable* x_i *that occurs in an inequality*

(ii) For each inequality $x_j - x_i \geq d_{ij}$ *, there is an edge* (v_i, v_j) *with weight* d_{ij} .

(iii)There is an extra source vertex, v_0*; it is located at x = 0 ; all other* vertices are at its right. There are n +1 vertices in total ($v_0, v_1, \ldots v_n$)

- If all the inequalities express minimum-distance constraints, the graph is acyclic (DAG).
- The longest path in a constraint graph determines the layout dimension





Constraint Graph

Maximum-Distance Constraints

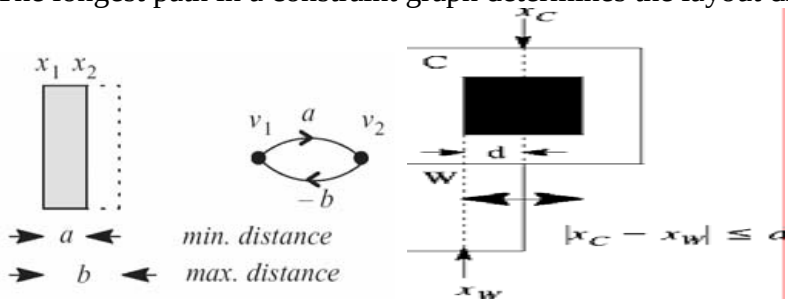
- Sometimes the distance of layout elements is bounded by a maximum, e.g., when the user wants a maximum wire width, maintains a wire connecting to a via, etc.

(i)A maximum distance constraint gives an inequality of the form:

$$x_j - x_i \leq c_{ij} \text{ where } c_{ij} \geq 0$$
or
$$x_i - x_j \geq -c_{ij}$$

(ii) Consequence for the constraint graph: backward edge -→ (v_j, v_i) ***with weight*** $d_{ji} = -c_{ij}$***; the graph is not acyclic anymore***

- The longest path in a constraint graph determines the layout dimension



3. (a)Write short notes on minimum distance rules.

- Mask patterns that are used for the fabrication of an integrated circuit have to obey certain restrictions on their shapes and sizes
- **Design rules** -restrictions on the mask patterns to increase the probability of successful fabrication.
- Sticking to the design rules decreases the probability that the fabricated circuit will not work due to short circuits, disconnections in wires, parasitics, etc.
- Shape of the patterns- often restricted to rectilinear polygons, i.e polygons that are made of horizontal and vertical segments only (Figure 1)

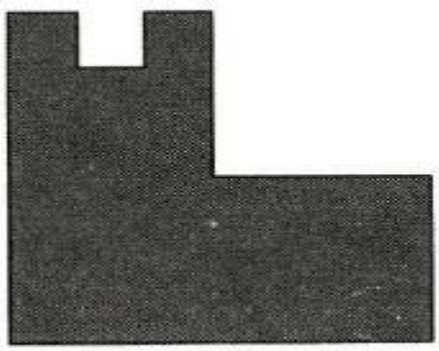


Figure 1- Rectilinear Polygon

- Some technologies allow 45-degree segments in polygons, segments that are parallel to the lines y = x or y = - x on an x-y plane(Figure 2)

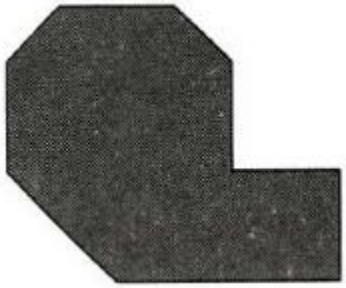


Figure 2 – 45- degree segments Polygon

- If patterns in two specific layers are constrained by one or more design rules, the layers are said to interact
- Example- polysilicon and diffusion are interacting layers as their overlapping creates a transistor, whereas polysilicon and metal form noninteracting layers
- Distances are often expressed in integer multiples (or small fractions) of a relative length unit, the **λ**, rather than absolute length units.
- All mask patterns are drawn along the lines of a so-called lambda grid (Figure)

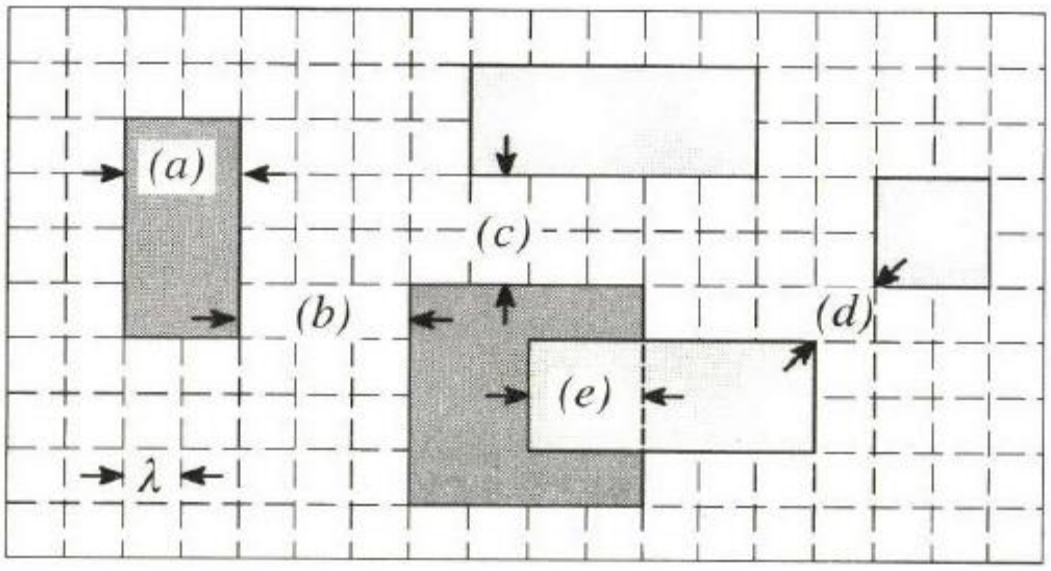


Fig (a) – minimum-width Fig (b, c, d) – minimum separation Fig(e) – minimum overlap

- Minimum width: a pattern in a certain layer cannot be narrower than a certain distance (Fig (a))
- Minimum separation: two patterns belonging to the same (Fig(b)) layer or to different but interacting layers (Fig (c)) cannot be positioned closer to each other than a certain distance; this is also true when the rectangles are diagonally separated (Fig(d)).
- Minimum overlap: a pattern in one layer located on top of a pattern in another interacting layer should have a minimal overlap (Fig (e)).

3. (b) Discuss briefly about applications of compaction

- A compaction program or compactor generates layout at the mask level. It attempts to make the layout as dense as possible.

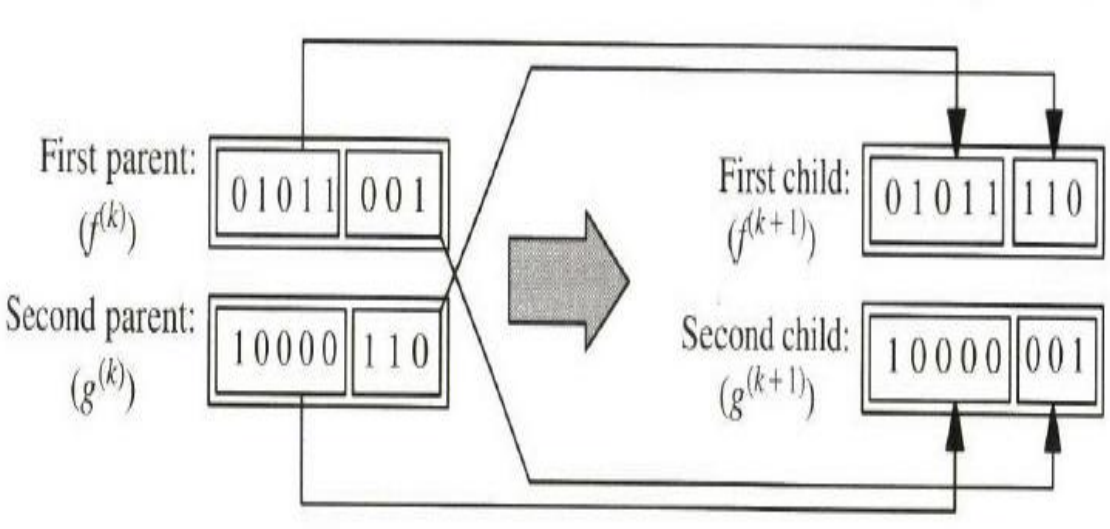
➢ Applications of compaction:

- Area minimization: remove redundant space in layout at the mask level.
- Layout compilation: generate mask-level layout from symbolic layout.
- Redesign: automatically remove design-rule violations.
- Rescaling: convert mask-level layout from one technology to another.

4. Explain how a genetic algorithm is used for search and solution of a combinatorial optimization problem.

- Works with fully specified solutions ***f*** included in the set of feasible solutions ***F***.
- The algorithm simultaneously keeps track of a set ***P*** of feasible solutions, called the ***population.***
- In an iterative search process, the current population $P^{(k)}$ is replaced by the next one $P^{(k+1)}$ using a procedure that is characteristic for genetic algorithms.
- To generate a feasible solution $f^{(k+1)} \in P^{(k+1)}$, two feasible solutions ,$f^{(k)}$ ***and*** $g^{(k)}$ called the parents of the child $f^{(k+1)}$ are first selected from $P^{(k)}$
- $f^{(k+1)}$ is generated in such a way that it inherits parts of its "properties" from one parent and the other part from the second parent by the application of an operation called crossover.

- First of all, this operation assumes that all feasible solutions $f \epsilon F$ can be encoded by a fixed length vector: $f = [f_1, f_2, \ldots f_n]^T$
- Genetic algorithms use bit strings to represent feasible solutions.
- This not only implies that the number of vector elements ***n*** is fixed, but that the number of bits to represent the value of each element f_i $(1 \leq i \leq n)$ is fixed as well.
- The string of bits that specifies a feasible solution in this way, is called a ***chromosome***
- A feasible solution (in biological terms, the phenotype) and its encoding as a chromosome (its genotype).
- Given two chromosomes, a crossover operator will use some of the bits of the first parents and some of the second parent to create a new bit string representing the child.
- Working of crossover Operator
- Generate a random number ***r*** between ***1 and the length l*** of the bit strings for the problem instance.
- Copy the bits ***1 through r - 1 from the first parent*** and the ***bits r through l from the second parent*** into the bit string for the child
- Sometimes, it is customary to generate a second child using the same ***r***, now reversing the roles of both parents when copying the bits.
- The generation of a pair of children $f^{(k+1)}$ and $g^{(k+1)}$ from a pair of parents $f^{(k)}$ ***and*** $g^{(k)}$ is shown in figure
- Also illustrates a complication that arises in genetic algorithms due to the encoding of the solution as a bit string.
- One solution to this complication is to use strings that consist of more sophisticated data structures than single bits
- Sometimes, it is customary to generate a second child using the same ***r***, now reversing the roles of both parents when copying the bits.
- The generation of a pair of children $f^{(k+1)}$ and $g^{(k+1)}$ from a pair of parents $f^{(k)}$ ***and*** $g^{(k)}$ is shown in figure
- Also illustrates a complication that arises in genetic algorithms due to the encoding of the solution as a bit string.
- One solution to this complication is to use strings that consist of more sophisticated data structures than single bits



- Sometimes, it is customary to generate a second child using the same ***r***, now reversing the roles of both parents when copying the bits.
- The generation of a pair of children $f^{(k+1)}$ and $g^{(k+1)}$ from a pair of parents $f^{(k)}$ ***and*** $g^{(k)}$ is shown in figure
- Also illustrates a complication that arises in genetic algorithms due to the encoding of the solution as a bit string.
- One solution to this complication is to use strings that consist of more sophisticated data structures than single bits
- A chromosome could then be represented as a vector of e.g. integers or characters.
- The combination of the chromosome representation and the crossover operator for generating new feasible solutions, leads, however, to more complications.
- This behavior can be avoided by using special-purpose crossover operators called order crossover.
- This operator copies the elements of the first parent chromosome until the point of the cut into the child chromosome
- The remaining part of the child is composed of the elements missing in the permutation in the order in which they appear in the second parent chromosome.
- One needs to favor good solutions above bad solutions in some way. This is done by giving a stronger preference to parents with a lower cost when selecting pairs of parents to be submitted to the crossover operator.
- So, the better the cost of some feasible solution, the higher the chances that it will be selected for reproduction.
- **Pseudo Code**
- In the main loop of the code, two parents at a time are selected and used to generate one new child in the new population
- The function ***select*** is responsible for the selection of feasible solutions from the current population favoring those that have a better cost
- The function ***crossover*** actually generates a new child from two parent chromosomes
- The function stop decides when to terminate the search, e.g. when there has been no improvement on the best solution in the population during the last ***m*** iterations, where m is a parameter of the algorithm.
- Mutation - main feature of genetic algorithm
- This is a phenomenon also encountered in nature, viz. that errors can be made during the copying of a chromosome from a parent to the child
- Mutation helps to avoid getting stuck in a local minimum

```
genetic()
{
  int pop_size;
  set of struct chromosome pop, newpop;
  struct chromosome parent1, parent2, child;

  pop ← ∅;
  for (i ← 1; i ≤ pop_size; i ← i + 1)
    pop ← pop ∪ {"chromosome of random feasible solution"};
  do {
     newpop ← ∅;
     for (i ← 1; i ≤ pop_size; i ← i + 1) {
       parent1 ← select(pop);
       parent2 ← select(pop);
       child ← crossover(parent1, parent2);
       newpop ← newpop ∪ {child};
     }
     pop ← newpop;
  } while (!stop());
  "report best solution";
}
```

5. Explain the applications of the Bellman Ford algorithm with pseudo-code to the directed graph shown below in figure 1:

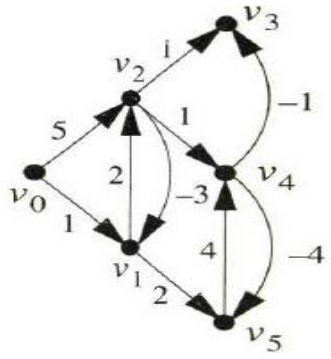
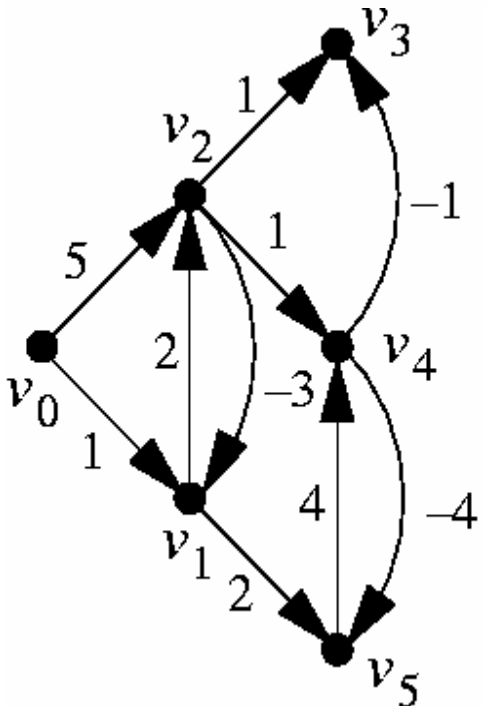


Figure 1 – Directed Cyclic Graph

- An alternative to the Liao-Wong algorithm
- Does not discriminate between forward and backward edges
- If there are more than ***n*** iterations, where ***n*** is the number of vertices in the graph ***G(V, E),***it can be concluded that the graph has positive cycles
- This can be seen as follows:
- ➢ After ***k*** iterations, the algorithm has computed the longest-path values for paths going through ***k - 1*** intermediate vertices.
- If there are no cycles, the algorithm should terminate after at most ***n*** iterations, as the longest path to a vertex can go through at most ***n - 1*** vertices.

- The time complexity of the Bellman-Ford algorithm is ***O(n x lE l)*** as each iteration visits all edges at most once and there are at most ***n*** iterations.

```
for (i ← 1; i ≤ n; i ← i + 1)
  x_i ← −∞;
x_0 ← 0;
count ← 0;
S_1 ← {v_0};
S_2 ← ∅;
while (count ≤ n && S_1 ≠ ∅) {
    for each v_i ∈ S_1
      for each v_j "such that" (v_i, v_j) ∈ E
        if (x_j < x_i + d_ij) {
          x_j ← x_i + d_ij;
          S_2 ← S_2 ∪ {v_j}
        }
    S_1 ← S_2;
    S_2 ← ∅;
    count ← count + 1;
}
if (count > n)
  error("positive cycle");
```



. Repeated "wave front propagation."

.S_1: *the current wave front.*

.x_i: *longest-path length from* v_0 *to* v_i.

. After k iterations, it computes the longest-path values for paths going through k-1 intermediate vertices.

S_1	x_1	x_2	x_3	x_4	x_5
"not initialized"	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
$\{v_0\}$	1	5	$-\infty$	$-\infty$	$-\infty$
$\{v_1, v_2\}$	2	5	6	6	3
$\{v_1, v_3, v_4, v_5\}$	2	5	6	7	4
$\{v_4, v_5\}$	2	5	6	8	4
$\{v_4\}$	2	5	7	8	4
$\{v_3\}$	2	5	7	8	4

- In each row, the first column gives the contents of the set s_1 at the beginning of the algorithm and the remaining entries of the row show the distance value after having processed all elements of s_1 .
- The final result is the same as the one obtained by the Liao-Wong algorithm

6. Write and explain the pseudo-code for simulated annealing. Detail the application in VLSI design with pros and cons.

Simulated annealing (sometimes also called statistical cooling) performs a computation that is analogous to a physical process.
A material is first heated up to a temperature that allows all its molecules to move freely around (the material becomes liquid), and is then cooled down very slowly.
The freedom of movement for the molecules decreases gradually until all the molecules take a fixed position.

- At the end of the process, the total energy of the material is minimal provided that the cooling is very slow.
- The analogy with the physical model has the following points of correspondence with a combinatorial optimization problem:
  - The energy corresponds to the cost function.
  - The movement of the molecules corresponds to a sequence of moves in the set of feasible solutions.
  - The temperature corresponds to a control parameter ***T*** which controls the acceptance probability for a move from ***f € F*** to ***g € N(f).***
  - Good move- ***c(g) ≤ c(f)*** – always accepted irrespective of the value of ***T***
  - Bad move- ***c(g) > c(f)*** - accepted with a probability where
- For high values of ***T*** nearly all bad moves are accepted, while hardly any bad move is accepted when ***T*** is low.
- The Boltzmann distribution in statistical mechanics states that the number of molecules ***N1*** with energy level ***ϵ1*** divided by the number of molecules ***N0*** with energy level ***ϵ0*** equals , where ***k*** is the Boltzmann constant and ***T*** is the absolute temperature.
- The algorithm consists of an outer loop in which the temperature is gradually lowered and an inner loop in which the configuration is randomly perturbed by moves that are either accepted or rejected

- The inner loop should be executed a number of times large enough to reach "thermal equilibrium" before going back to the outer loop.
- Accepting or rejecting a move is represented by the function ***accept***, in which the function ***random(k)*** generates a real-valued random number between ***0 and k*** with a uniform distribution.
- The function ***thermal-equilibrium*** should only return a value unequal to zero if the inner loop has been executed a "sufficient" number of times
- This number is normally a function of the problem instance size.
- The function ***new-temperature*** computes a new, lower, value for the temperature to be used for the next execution of the inner loop. In practice, this is often implemented by a multiplication of ***T*** by a constant between ***0 and 1.***
- The function ***stop***, finally, decides about the termination of the search.
- simulated annealing may visit an optimal solution and then move away from it
- So record the best solution in a separate variable and report its value at the end of the search instead of the final value of ***f***.
- The combination of the functions ***thermal-equilibrium***, ***new-temperature*** and ***stop*** realizes a strategy for simulated annealing, which is called the cooling schedule.

```
int accept(struct feasible_solution f, g)
{
  float Δc;

  Δc ← c(g) − c(f);
  if (Δc ≤ 0)
    return 1;
  else return (e^(−Δc/T) > random(1));
}

simulated_annealing()
{
  struct feasible_solution f, g;
  float T;

  f ← initial_solution();
  do {
     do {
        g ← "some element of N(f)";
        if (accept(f, g))
           f ← g
     while (!thermal_equilibrium());
     T ← new_temperature(T);
  while (!stop());
  "report f";
}
```

Figure 5.9 A pseudo-code description of simulated annealing.

7. Explain the principle of Branch and bound algorithm for finding the optimal solution of a combinatorial optimization problem.

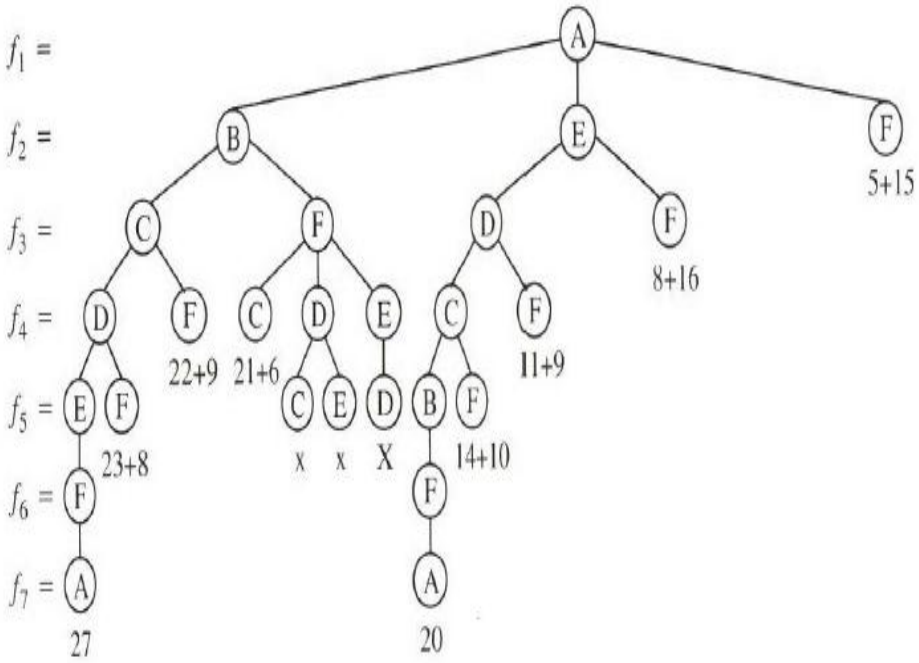
- It is not necessary to visit all (partial) solutions that the backtracking procedure generates.
- Let D($\tilde{\mathbf{f}}^{(k)}$) denote the set of fully- specified solutions in the subtree with root $\tilde{\mathbf{f}}^{(k)}$
- Information about a certain partial solution $1 \leq k < n$, at a certain level can indicate that any fully-specified solution $\tilde{\mathbf{f}}^{(k)}$ derived from it can never be the optimal solution
- This conclusion is based on the estimation of the lower bound of the cost of all solutions in D($\tilde{\mathbf{f}}^{(k)}$)
- The function that estimates this cost lower bound will be denoted by $\tilde{\mathbf{f}}^{(k)}$
- If inspection of $\tilde{\mathbf{f}}^{(k)}$ can guarantee that all of the solutions in D($\tilde{\mathbf{f}}^{(k)}$) have a higher cost than some solution already found earlier during the backtracking, none of the children of need any further investigation.
- We can say that the node in the tree corresponding to $\tilde{\mathbf{f}}^{(k)}$ can be killed.
- The modification of the backtracking algorithm that provides in killing partial solutions is called branch-and-bound
- One can ***prune*** the search tree by removing the subtree having $\tilde{\mathbf{f}}^{(k)}$ as its root.
- **Pseudo Code**
  - The main recursive procedure is called ***b-and_ b***
  - procedure ***lower-bound-cost*** is called to get a lower bound of the partial solution based on the function
- A next level of recursion is entered only if the node in the search tree cannot be killed

$$\tilde{c}(\tilde{\mathbf{f}}^{(k)}) = \tilde{g}(\tilde{\mathbf{f}}^{(k)}) + \tilde{h}(\tilde{\mathbf{f}}^{(k)})$$

where $\tilde{g}(\tilde{\mathbf{f}}^{(k)})$ is a function that is computed given the specified variables of $\tilde{\mathbf{f}}^{(k)}$ and $\tilde{h}(\tilde{\mathbf{f}}^{(k)})$ is a term that is based on the unspecified variables of $\tilde{\mathbf{f}}^{(k)}$. Taking the traveling salesman problem as an example, $\tilde{g}(\tilde{\mathbf{f}}^{(k)})$ can give the length of the path fixed by the specified variables and $\tilde{h}(\tilde{\mathbf{f}}^{(k)})$ can give a lower bound on the remaining tour length. One possibility is to define $\tilde{h}(\tilde{\mathbf{f}}^{(k)}) = 0$ for all partial solutions. This certainly satisfies the requirement that all final solutions in the set $D(\tilde{\mathbf{f}}^{(k)})$ have a cost higher than the estimated one. However, this definition does not help much to reduce the search

The branch-and-bound search tree for the TSP example of Figure 5.3 that is obtained using a function $\tilde{h}(\tilde{\mathbf{f}}^{(k)})$ based on the minimal spanning tree, is shown in Figure 5.6. In the figure, a label '$a + b$' means that the node is killed due to the fact that the sum exceeds the cost of the best known solution up to that moment. The term a is the value of $\tilde{g}(\tilde{\mathbf{f}}^{(k)})$ and b is the value of $\tilde{h}(\tilde{\mathbf{f}}^{(k)})$. A label 'X' means that the subgraph induced by the remaining nodes is not connected and that a spanning tree does not exist. Clearly, as a consequence, a solution to TSP cannot be found in the subtree of that node. Note that the number of nodes in the search tree is 72 for the exhaustive search version and 27 for the branch-and-bound version.

```
float best_cost;
solution_element val[n], best_solution[n];

b_and_b(int k)
{
  float new_cost;
  if (k = n) {
    new_cost := cost(val);
    if (new_cost < best_cost) {
      best_cost := new_cost;
      best_solution := copy(val);
    }
  }
  else if (lower_bound_cost(val,k) ≥ best_cost)
        /* No action, node is killed. */
  else
    for each (el ∈ allowed(val, k)) {
      val[k] := el;
      b_and_b(k + 1);
    }
}

main ()
{
   best_cost := ∞;
   b_and_b(0);
   report(best_solution);
}
```



Search Tree for the TSP Problem using Branch-and-Bound Algorithm