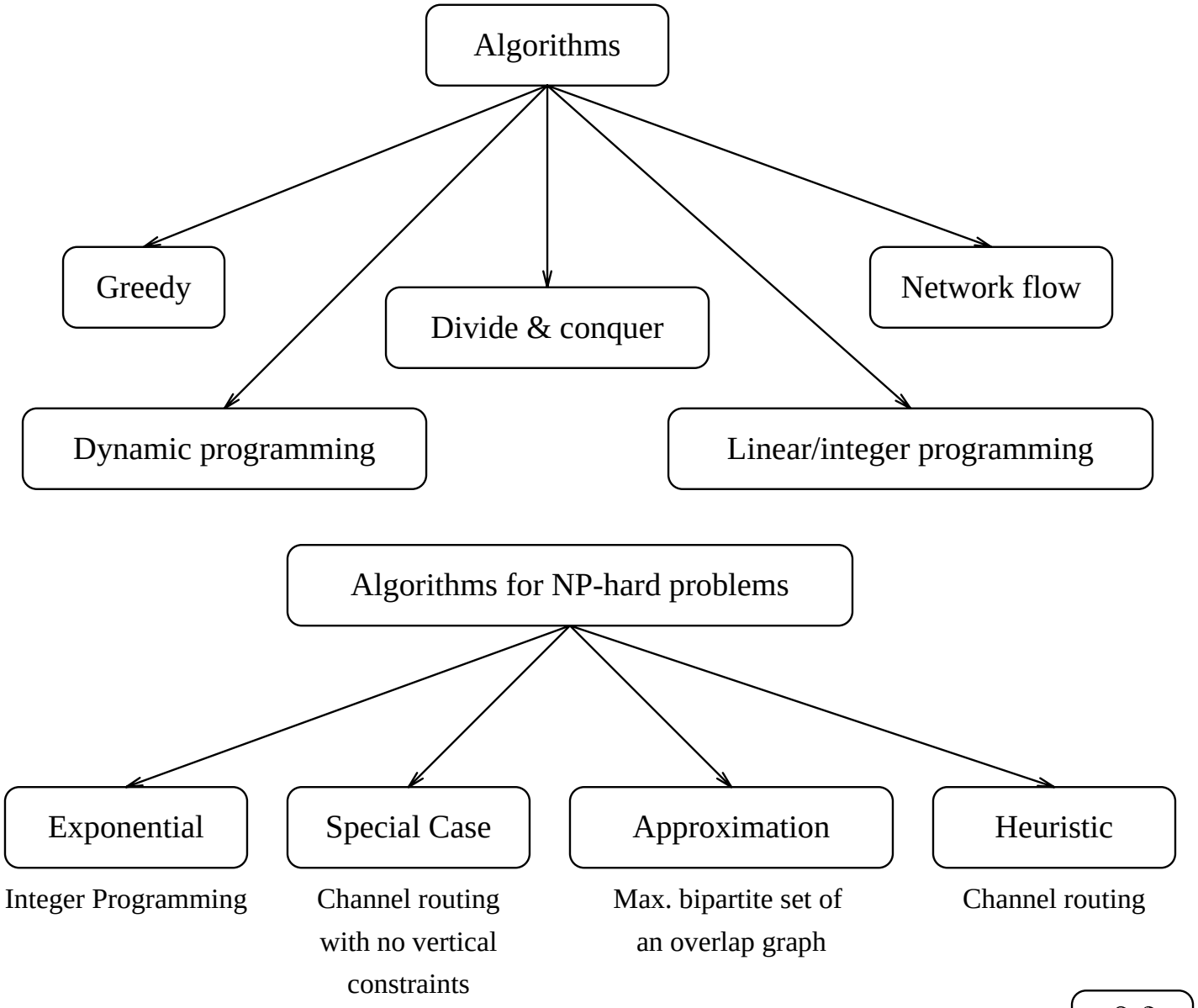
Data Structures and Basic Algorithms

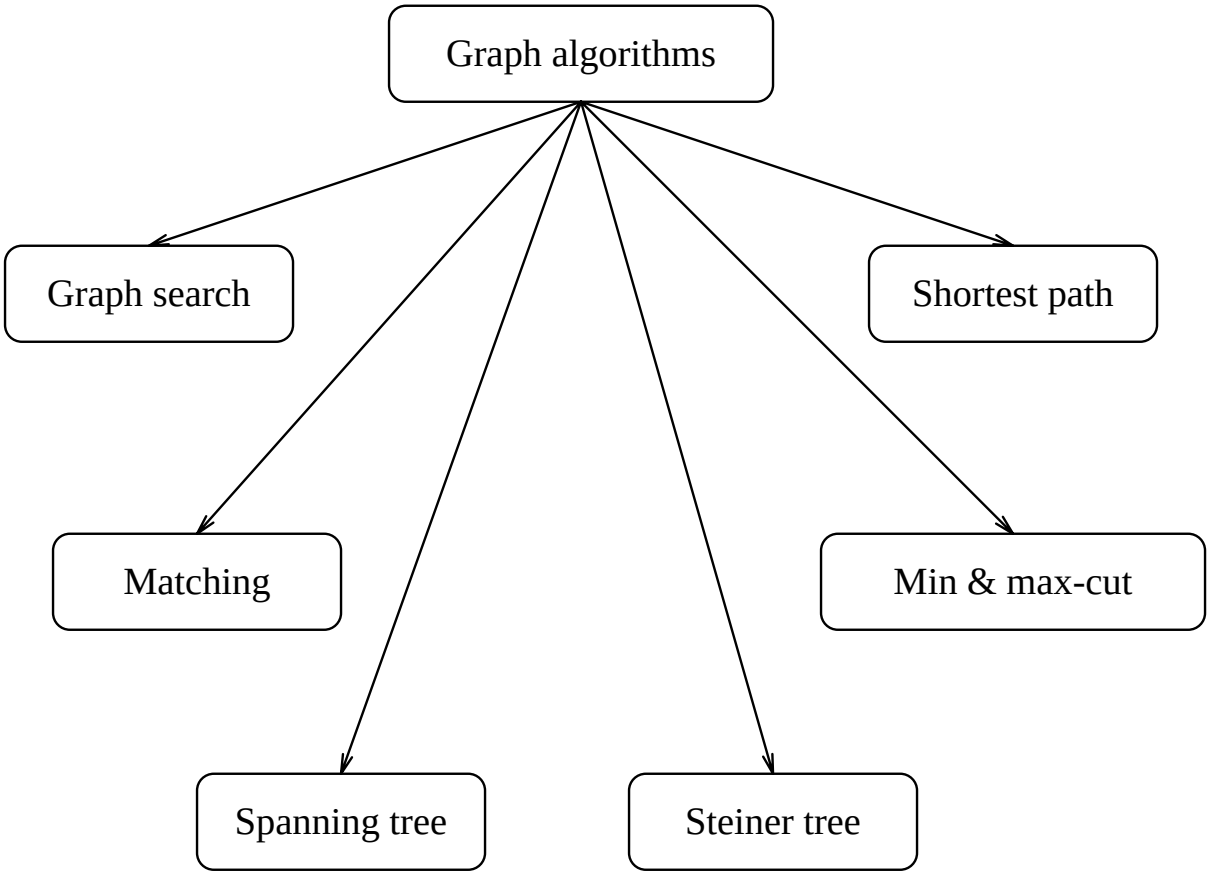
Objectives:

- To review basic algorithms.
- To understand how a layout is represented and manipulated.
- To study the types of graph algorithms that are of use in the VLSI design automation field.

Classifications of Algorithms



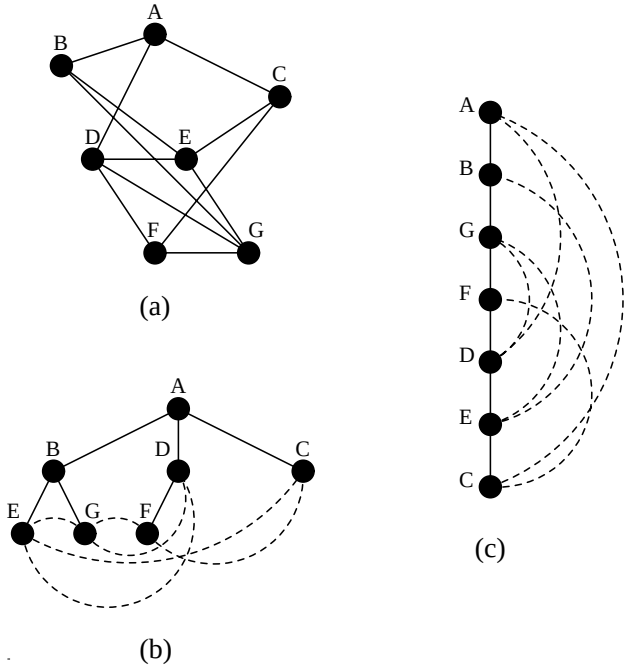
Graph Algorithms



Graph Search Algorithms

- Depth-First Search
- Breadth-First Search
- Topological Search

Examples of graph search algorithms



(a) A graph (b) Breadth-First Search (c) Depth-First Search

Algorithm DEPTH-FIRST-SEARCH

Algorithm DEPTH-FIRST-SEARCH(v)
begin
    MARKED(v) = 1;
    for each vertex u, such that $(u, v) \in E$ **do**
        if MARKED(u) = 0 **then**
            DEPTH-FIRST-SEARCH(u);
end.

Spanning Tree

Problem Formulation:

Given a graph $G = (V, E)$, select a subset $V' \subseteq V$, such that V' has property $\mathcal{P}$.

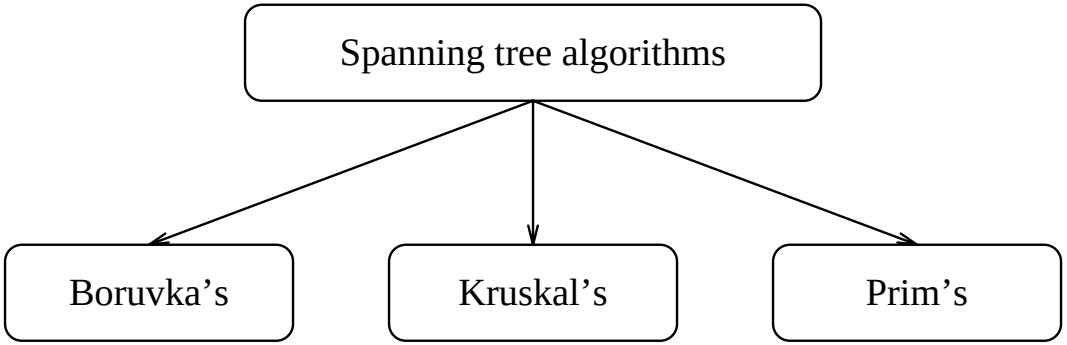
Minimum Spanning Tree

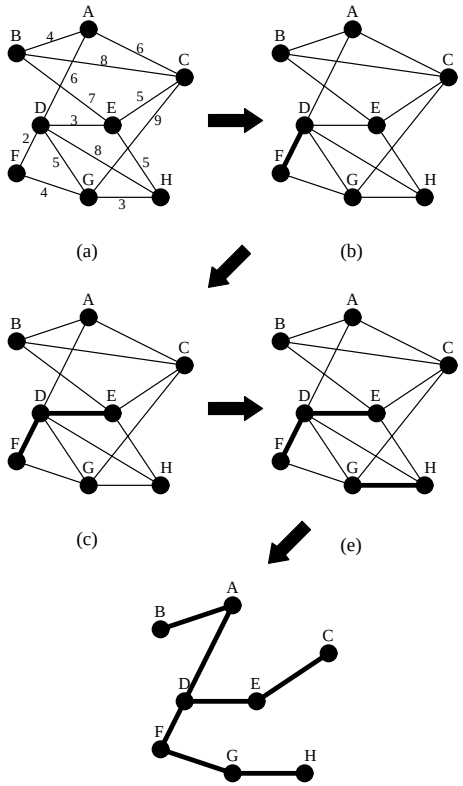
Problem Formulation:

Given an edge-weighted graph $G = (V, E)$, select a subset of edges $E' \subseteq E$ such that E' induces a tree and the total cost of edges $\Sigma_{e_i \in E'} wt(e_i)$, is minimum over all such trees, where $wt(e_i)$ is the cost or weight of the edge e_i.

– Used in routing applications.

Spanning Tree Algorithms



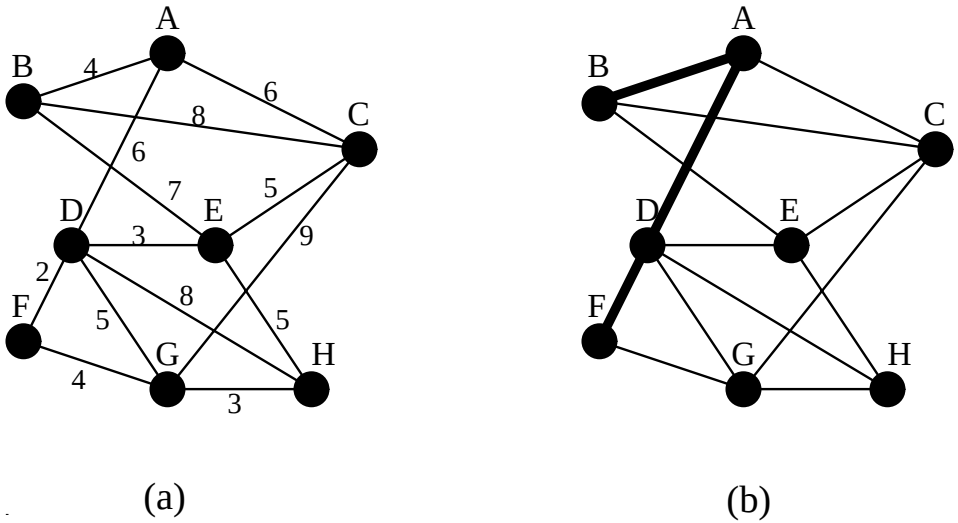


Dijsktra's Shortest-Path Algotithm

```
Algorithm SHORTEST-PATH(u)
begin
for i = 1 to n do
    if ((u, i) ∈ E) then D[i] = wt(u, i);
    else D[i] = +∞;
    P[i] = u;
V' = V − u; D[u] = 0;
while (|V'| > 0) do
    Select v such that D[v] = min_{w∈V'} D[w];
    V' = V' − v;
    for w ∈ V' do
        if (D[w] > D[v] + wt(v, w)) then
            D[w] = D[v] + wt(v, w);
            (* D[w] is the length of the shortest path from u to w. *)
            P[w] = v;
            (* P[w] is the parent of w. *)
for w ∈ V do
    (* print the shortest path from w to u. *)
    q = w;
    print q;
    while (q ≠ u) do
        q = P[q];
        print q;
    print q;
end.
```

Shortest Path Example

The Shortest Path between B and F



(a) (b)

- Used in global, detailed, and clock routing applications.

Min-Cut & Max-Cut Algorithms

Min-cut and Max-cut algorithms partition the vertex set of a graph.

1. **Min-cut:**

   Given a graph $G = (V, E)$, partition V into subsets V_1 and V_2 of equal sizes, such that the number of edges $E' \subseteq E$, and $E' = \{(u, v) | u \in V_1, v \in V_2\}$ is minimized.

   - Used in partitioning and placement applications.

2. **Max-cut:**

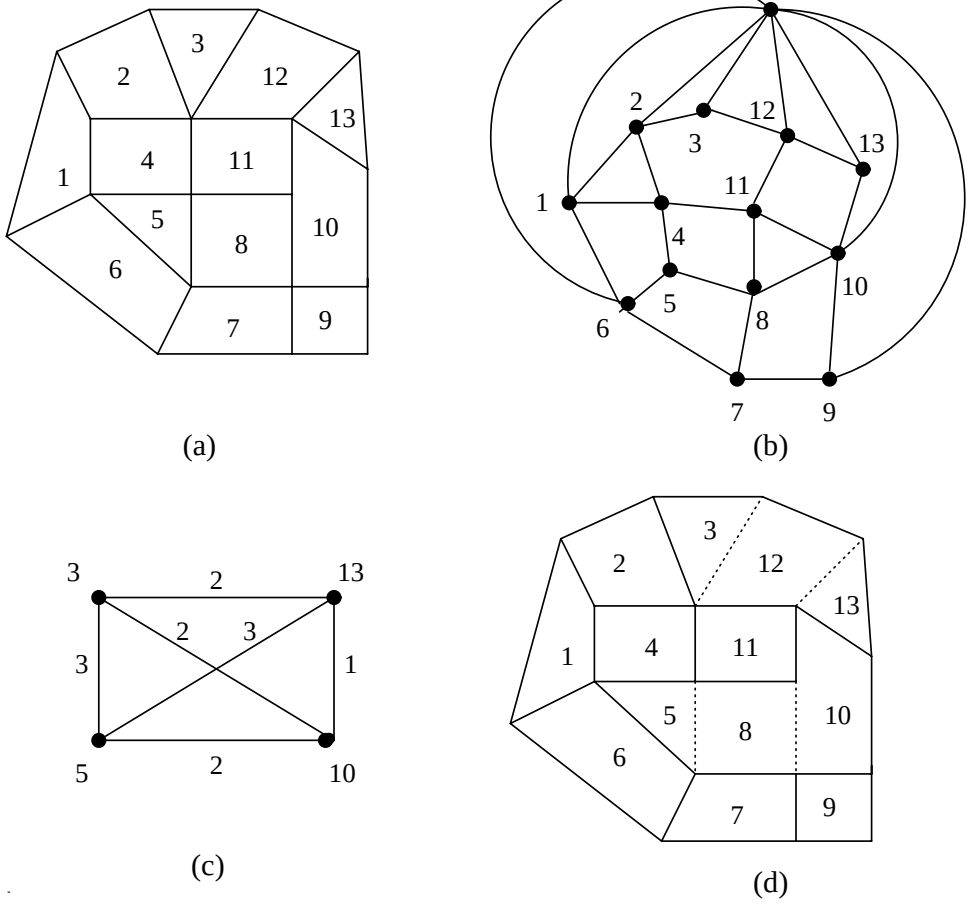
   Given a graph $G = (V, E)$, find the maximum bipartite graph of G. Let $G' = (V, E')$ be the maximum bipartite of G, which is obtained by deleting K edges of G, then G has a max-cut of size $|E| - K$.

   - Used in compaction applications.

Algorithm MAXCUT

Algorithm MAXCUT
begin
F = PLANAR-EMBED(G);
Let $f_1, f_2, \ldots, f_k$ be the faces of the F;
G_F = CONSTRUCT-DUAL(F);
Let R be the set of vertices of odd degree in G_F and let
Q be the set of all pairs consisting of vertices in R;
G_W = CONSTRUCT-WT-GRAPH(R, Q);
M = MIN-WT-MATCHING(G_W);
Using M find a set of paths, one for each of the matched pairs
in G_F;
Each path determines a set of edges whose deletion leaves the
graph G bipartite;
end.

An example of finding a max-cut



(a) A graph (b) Its dual (c) A complete weighted graph by using only vertices corresponding to odd faces (d) The resultant bipartite graph

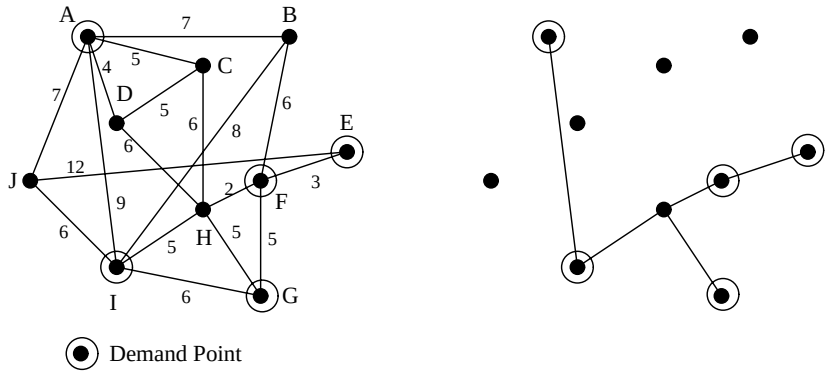
Steiner Trees

1. **Problem formulation:**

   Given an edge weighted graph $G = (V, E)$ and a subset $D \subseteq V$, select a subset $V' \subseteq V$, such that $D \subseteq V'$ and V' induces a tree of minimum cost over all such trees.

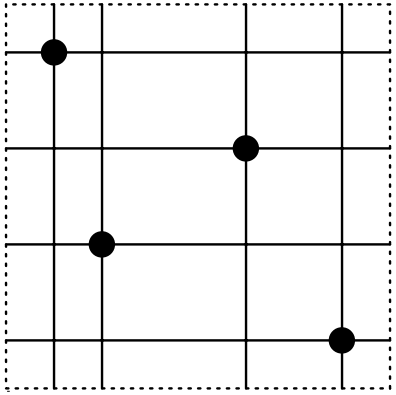
   The set D is referred to as the set of *demand points* and the set $V' - D$ is referred to as *Steiner points*.

- Used in the global routing of multi-terminal nets.

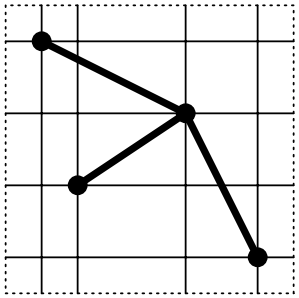


Underlying Grid Graph

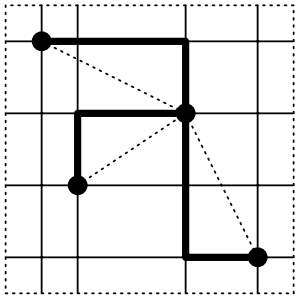
The underlying grid graph is defined by the intersections of the horizontal and vertical lines drawn through the demand points.



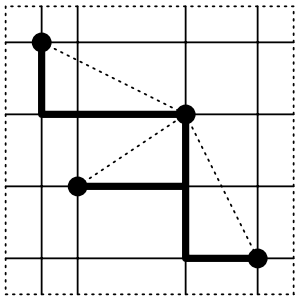
Different Steiner trees constructed from a MST



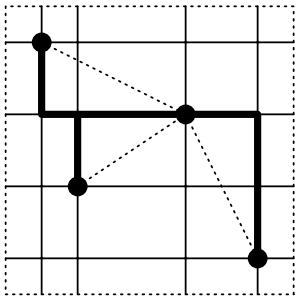
(a)



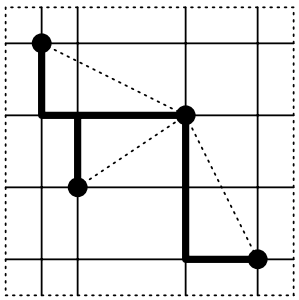
(b)



(c)



(d)



(e)

Line Sweep Method

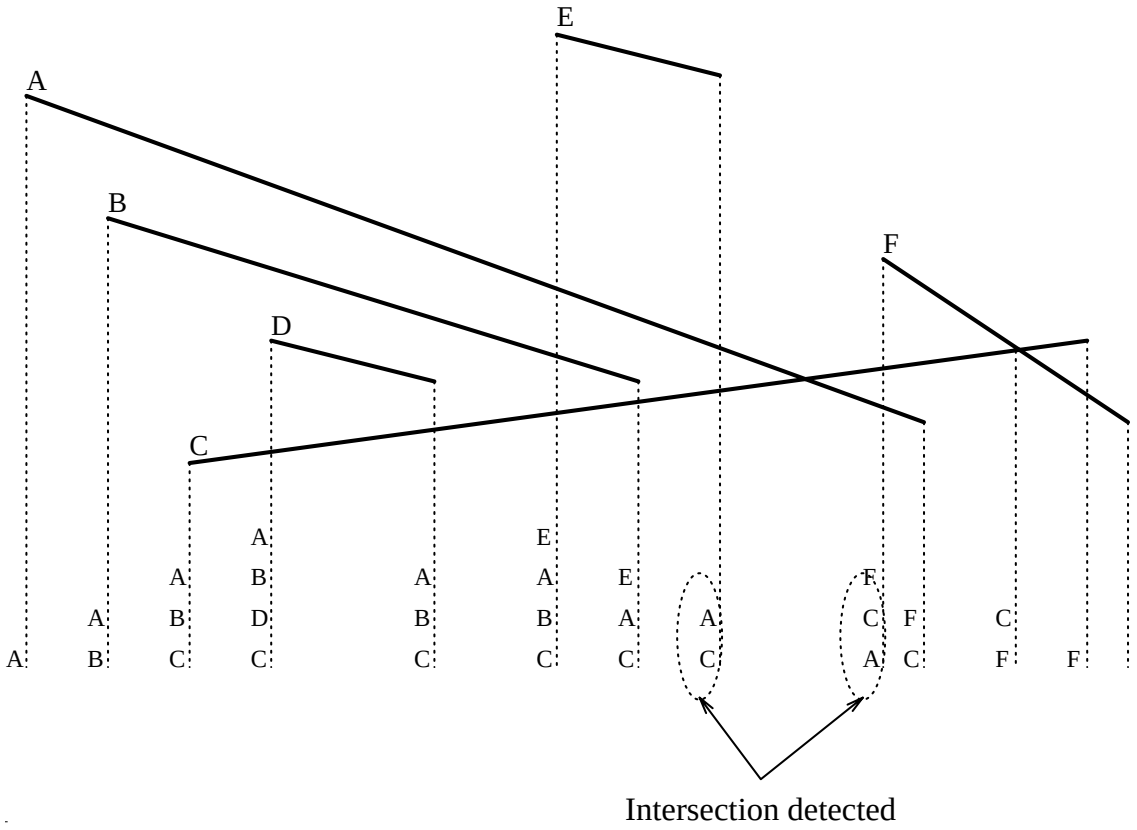
1. Sorts n line segments represented by their $2n$ endpoints by increasing x-coordinate values.
2. A virtual vertical line sweeps the endpoint set from left to right.
3. At each x-coordinate, a list of endpoints is established, sorted by their y-coordinate.
4. If two endpoints change order in one of these lists, those two lines intersect.

- Used in channel and switchbox routing aplications.

Algorithm LINE-SWEEP

Algorithm LINE-SWEEP
begin
Sort the endpoints lexicographically on
x-coordinates so that Point[1] is leftmost and
Point[2n] is rightmost;
for $i = 1$ to $2n$ **do**
(* Let S be the segment of which P is an endpoint *)
P = Point[i];
if P is the left endpoint of S **then**
INSERT (S, T);
A = ABOVE(S, T);
B = BELOW(S, T);
if A intersects S **then** return (A, S);
if B intersects S **then** return (B, S);
else (* P is the right endpoint of S *)
A = ABOVE(S, T);
B = BELOW(S, T);
if A intersects B **then** return (A, B);
Delete(S, T);
end.

Line Sweep Example



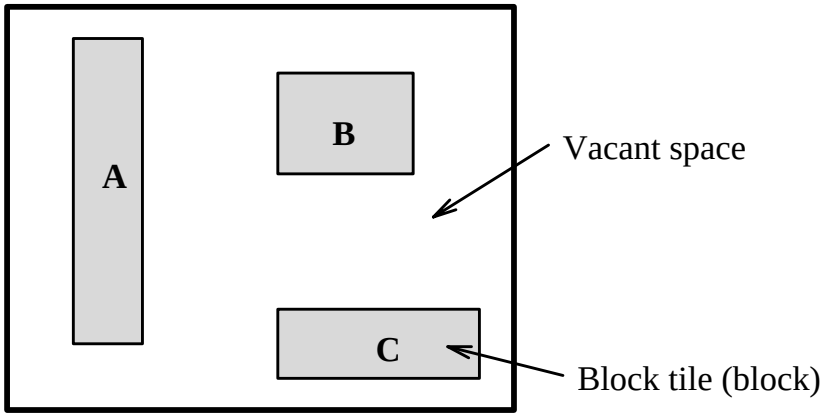
Intersection detected

Layout Representation

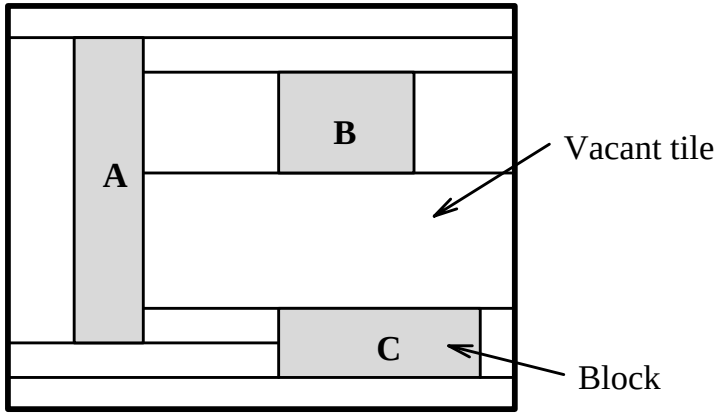
A layout is a collection of *tiles*.

- A tile is a rectangular section within a single layer.
- Tiles are not allowed to overlap within a layer.
- The elements of a layout are refered to as *block tiles*.
- Block tiles are used represent to reperesent p-diffusion, n-diffusion, poly segment, etc.
- The area of a layout that does not contain a block is refered to as *vacant space.*
- Vacant space can be partitioned into a series of *vacant tiles*.

Layout Representation Examples



Block tile (block) representation in a layout.



Vacant space partitioned into vacant tiles.

Atomic Operations for Layout Editors

The basic set of operations that give a designer the freedom to fully manipulate a layout, is referred to as the *Atomic Operations*

- Point Finding
- Neighbor Finding
- Block Visibility
- Area Searching
- Directed Area Enumeration
- Block Insertion
- Block Deletion
- Plowing
- Compaction
- Channel Generation

Atomic Operations (cont.)

- **Point Finding:**

  Given the coordinate of a point $p = (x, y)$, determine whether p lies within a block and, if so, identify that block.

- **Neighbor Finding:**

  This operation is used to determine all blocks touching a given block B.

- **Block Visibility:**

  This operation is used to determine all blocks visible, in the x and y direction, from a given block B.

Atomic Operations (cont.)

- **Area Search:**

  Given a fixed area A, defined by its upper left corner (x, y), the length l, and the width w, determine whether A intersects with any blocks B.

- **Directed Area Enumeration:**

  Given a fixed area A, defined by its upper left corner (x, y), length l, and width w, visit each block intersecting A exactly once in sorted order according to its distance from a given side (top, bottom, left, or right) of A.

Atomic Operations (cont.)

- **Block Insertion:**

  Block insertion refers to the act of inserting a new block B into the layout such that B does not intersect with any existing block.

- **Block Deletion:**

  This operation is used to remove an existing block B from the layout. In an iterative approach to placement, blocks are moved from one location to another. This is done by inserting the block into a new location and then deleting the block at its previous location.

Atomic Operations (cont.)

- **Plowing:**

  Given an area A and direction d, remove all blocks B_i from A by shifting each B_i in direction d while preserving ther order.

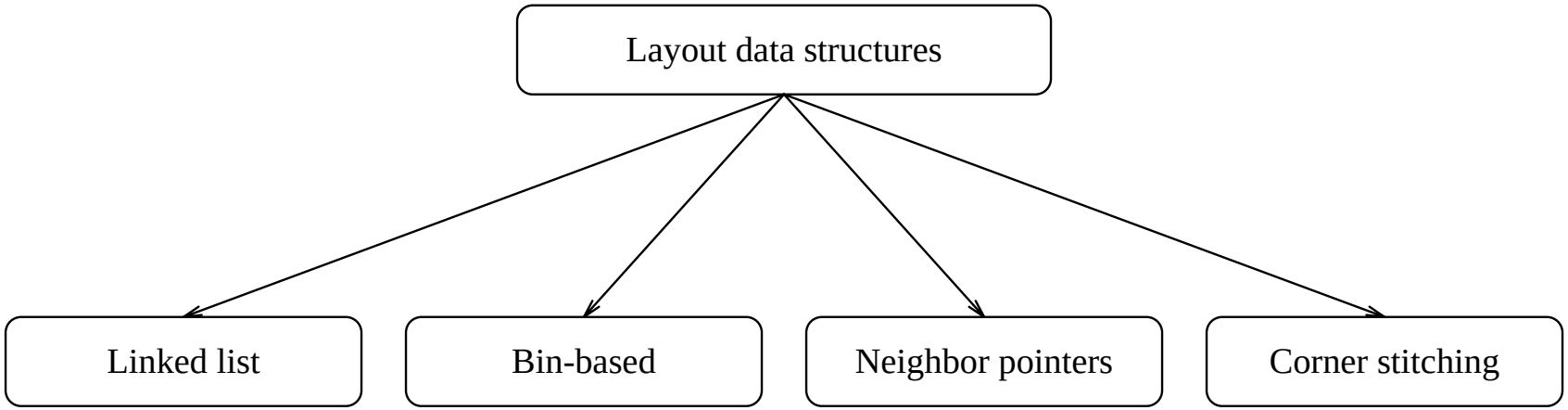
- **Compaction:**

  Compaction refers to plowing or "compressing" of the entire layout. If compaction is along the x-axis or y-axis then the compaction is called *1-dimensional compaction*. If compaction is carried out in both x- and y-direction then it is called *2-dimensional compaction*.

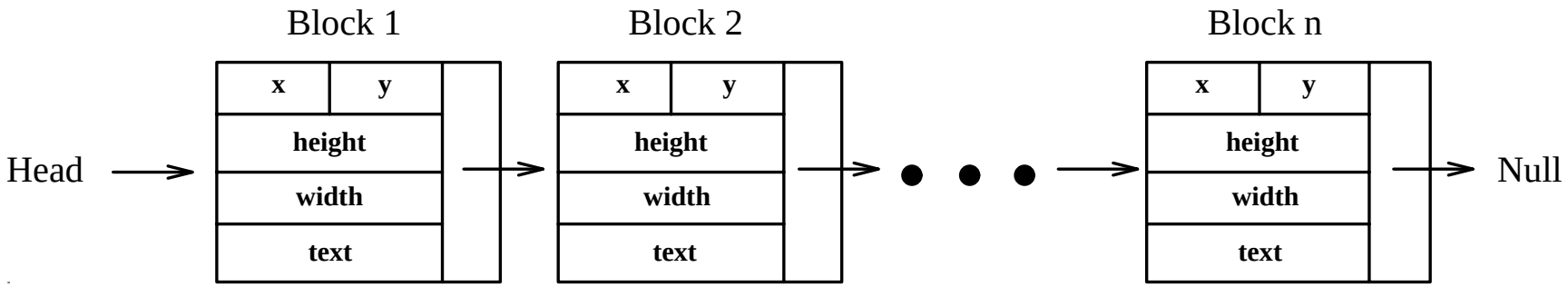
- **Channel Generation:**

  This operation refers to determining the vacant space in the layout and partitioning it into tiles.

Data Structures Used to Represent Layouts



Linked List Representation

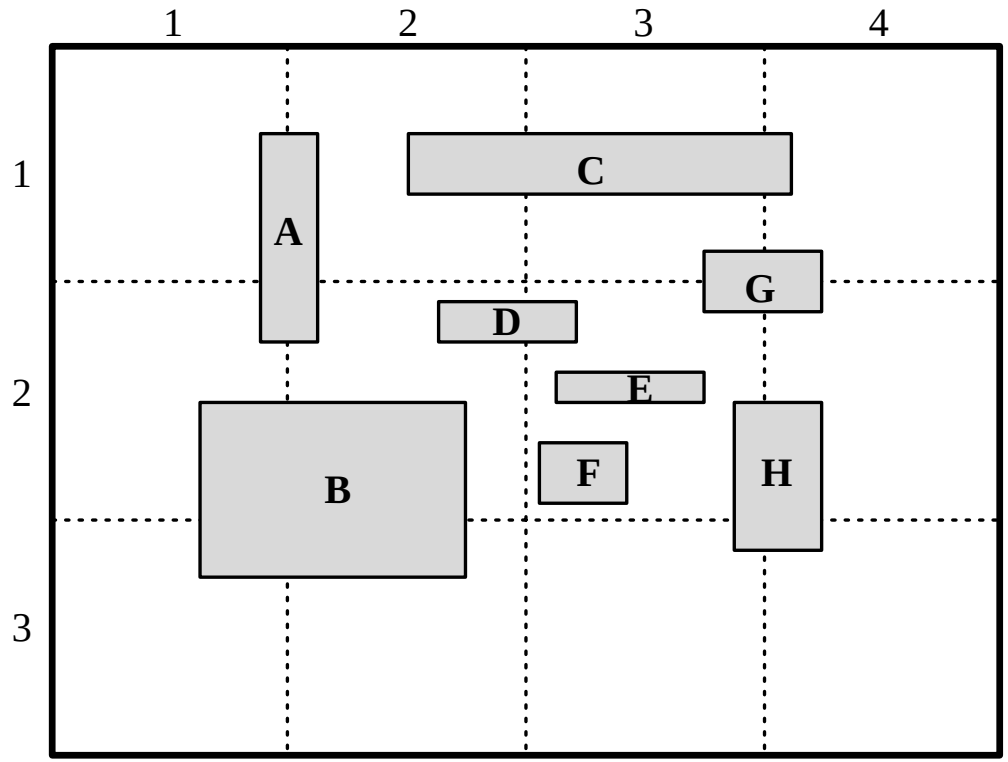


Algorithm NEIGHBOR-FIND1

This algorithm finds the neighbors on the right side of the given block B by using linked list data structure.

Algorithm NEIGHBOR-FIND1(B, L)
begin
    $neighbor\text{-}list = \phi$;
    $x1 = B.x + B.width$;
    $y11 = B.y$;
    $y12 = B.y + B.height$;
    for each block $R \in L$ such that $R \neq B$ **do**
        $y21 = R.y$;
        $y22 = R.y + R.height$;
        if ($x1 = R.x$ **and** ($y11 \leq y21 \leq y12$ **or**
                $y11 \leq y22 \leq y12$ **or**
                $y21 < y11 < y12 < y22$)) **then**
            INSERT(R, $neighbor\text{-}list$);
    return $neighbor\text{-}list$;
end.

Bin-Based Representation

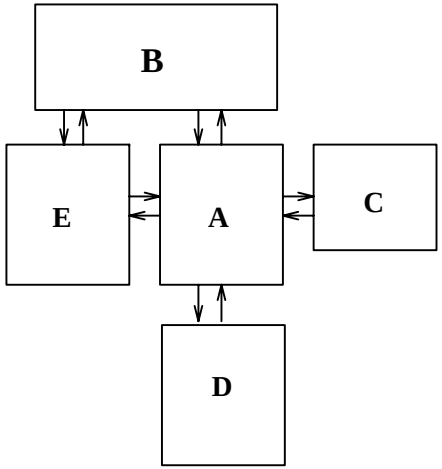


Algorithm NEIGHBOR-FIND2

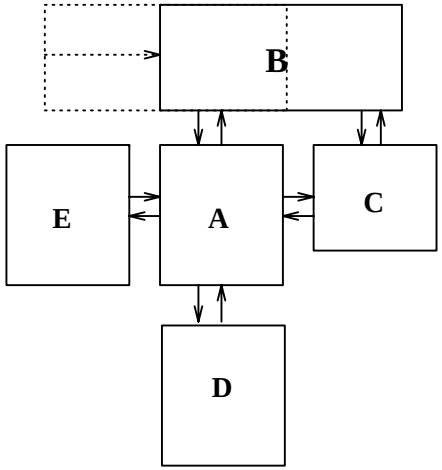
This algorithm finds the neighbors on the right side of block A by using bins.

Algorithm NEIGHBOR-FIND2(A, $\mathcal{B}$)
begin
  $neighbor\text{-}list = \phi$;
  $x1 = A.x + A.width$;
  $y11 = A.y$;
  $y12 = A.y + A.height$;
  let $B' \subseteq \mathcal{B}$ be set of bins which contain A;
  for all bins $X \in B'$ **do**
    for each block $R \in X$ **do**
      $y21 = R.y$;
      $y22 = R.y + R.height$;
      if ($x1 = R.x$ **and** ($y11 \leq y21 \leq y12$ **or**
        $y11 \leq y22 \leq y12$ **or**
        $y21 < y11 < y12 < y22$)) **then**
        INSERT($R, neighbor\text{-}list$);
  return $neighbor\text{-}list$;
end.

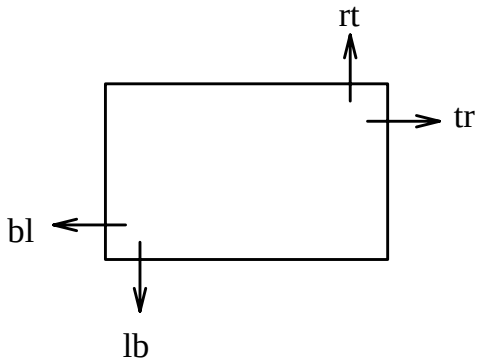
Neighbor Pointers



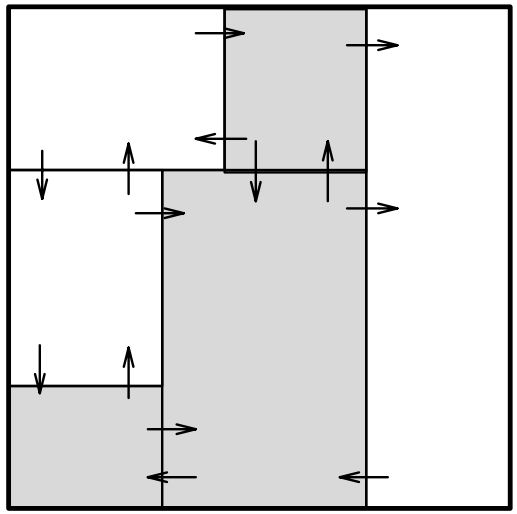
Update of Neighbor Pointers



Corner Stitches



An Example of the Corner Stitch Layout



Point Finding Using Corner Stitches

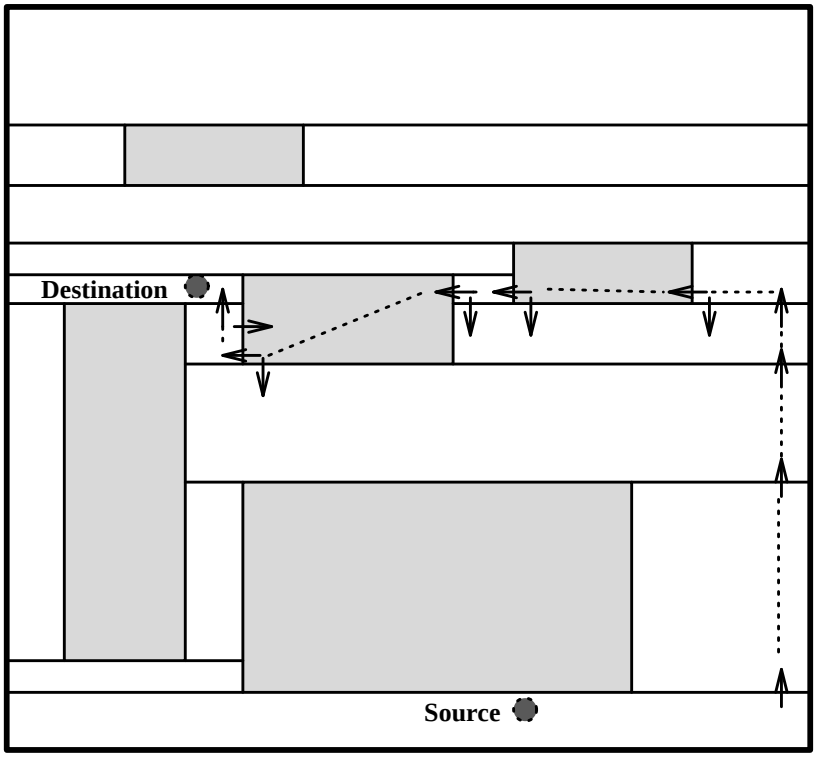
Point Finding: Given a point p_2, find the path through the corner stitches from the current point p_1 to p_2 traversing the minimum number of tiles.

(1) Move up or down, using **rt** and **lb** pointers until a tile is found whose vertical range contains the destination point.
(2) Move left or right using **tr** or **bl** pointers until a tile is found whose horizontal range contains the destination point.
(3) Whenever there is a misalignment (the search goes out of the horizontal range of the tile that contains the destination point) due to the above operations, steps 1 and 2 has to be iterated several times to locate the tile containing the point.

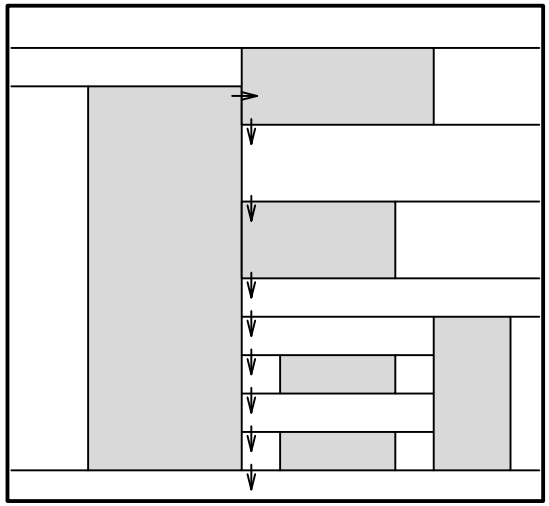
Point Finding Algorithm

Algorithm POINT-FIND(B, x, y)
begin
    do
        while $(y < B.y)$ **or** $(y > B.y + B.heigth)$ **do**
            if $y < B.y$ **then** $B = B.rt$; **else** $B = B.lb$;
        while $(x < B.x)$ **or** $(x > B.x + B.width)$ **do**
            if $x < B.x$ **then** $B = B.bl$; **else** $B = B.tr$;
    while $(y < B.y)$ **or** $(y > B.y + B.heigth)$;
end.

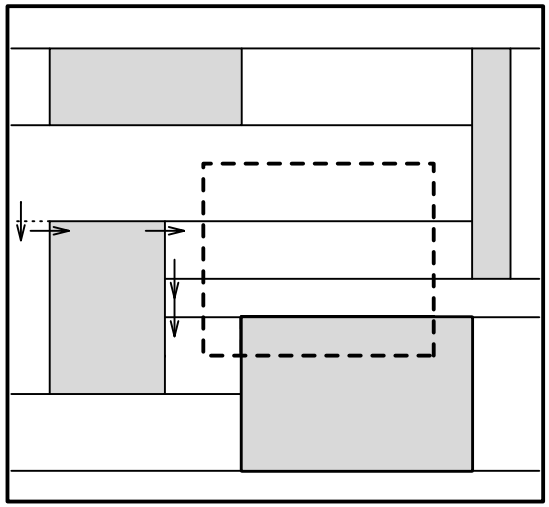
Point Finding Example



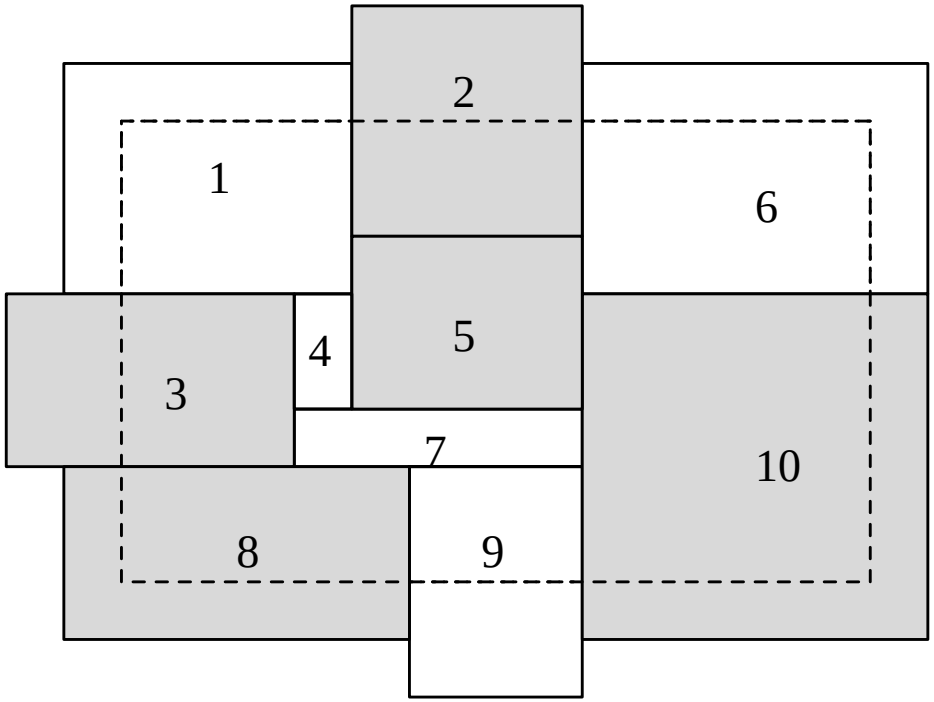
Neighbor Finding Using Corner Stitches



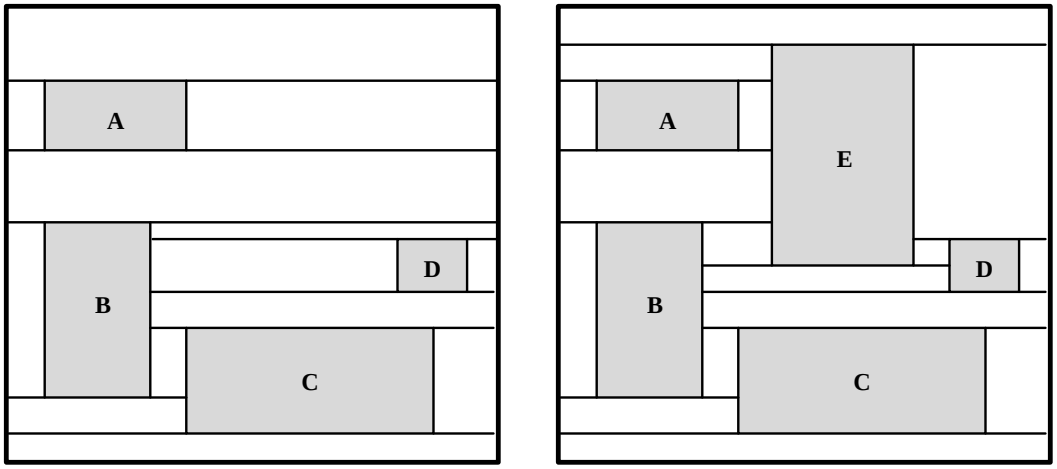
Area Search Using Corner Stitches



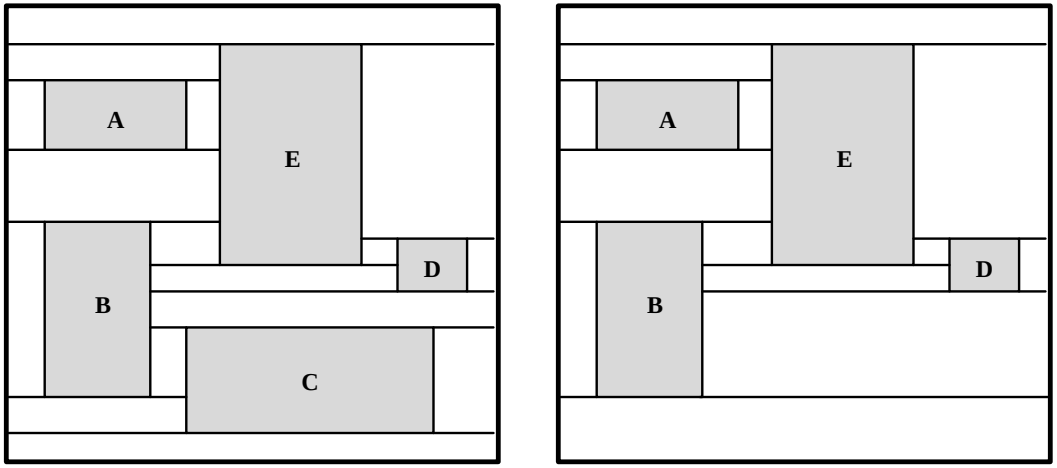
Enumerate Tiles Using Corner Stitches



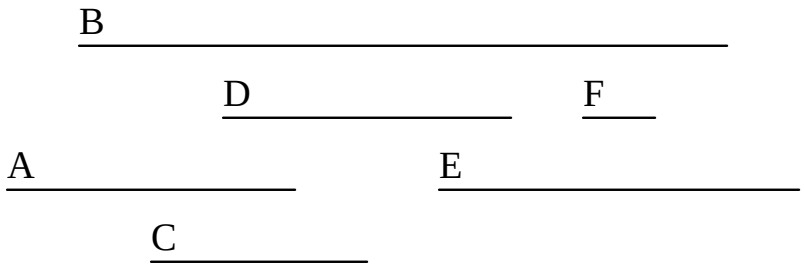
Block Creation Using Corner Stitches



Block Deletion Using Corner Stitches

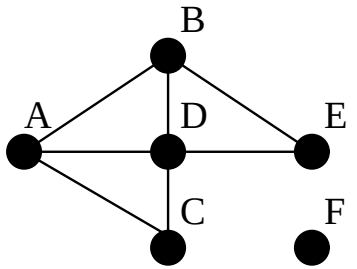


Graphs Associated with A Set of Intervals



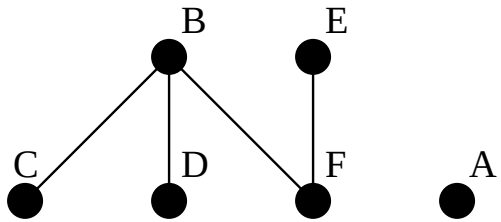
(a)

G_O



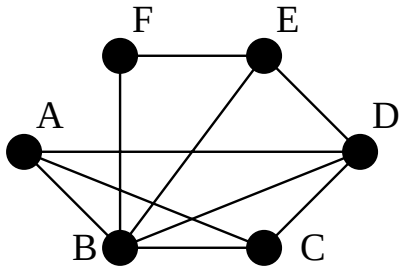
(b)

G_C



(c)

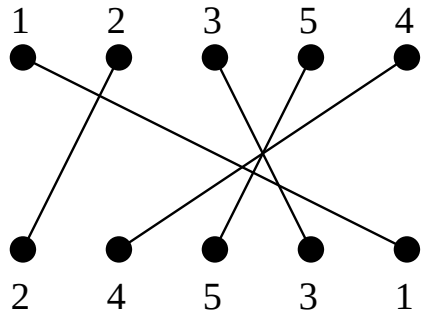
G_I



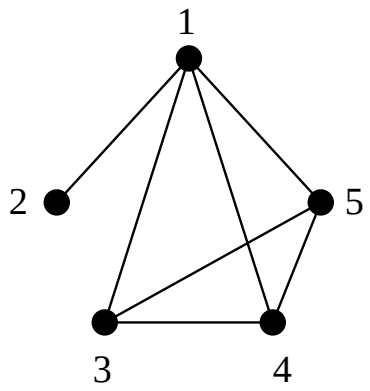
(d)

(a)intervals (b)overlap graph (c)containment graph (d)interval graph

Matching Diagram and Permutation Graph

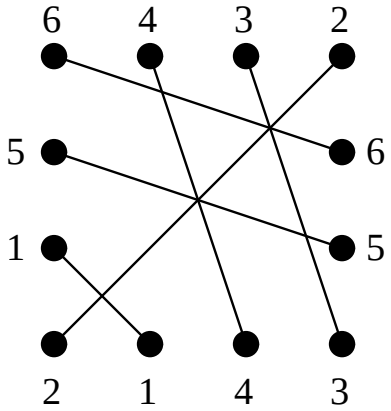


(a)

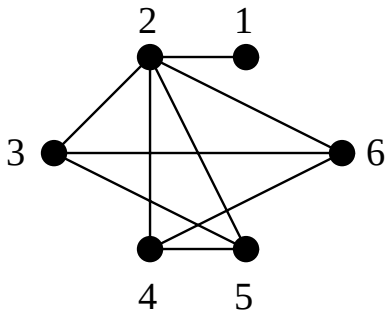


(b)

Switchbox and Its Circle Graph

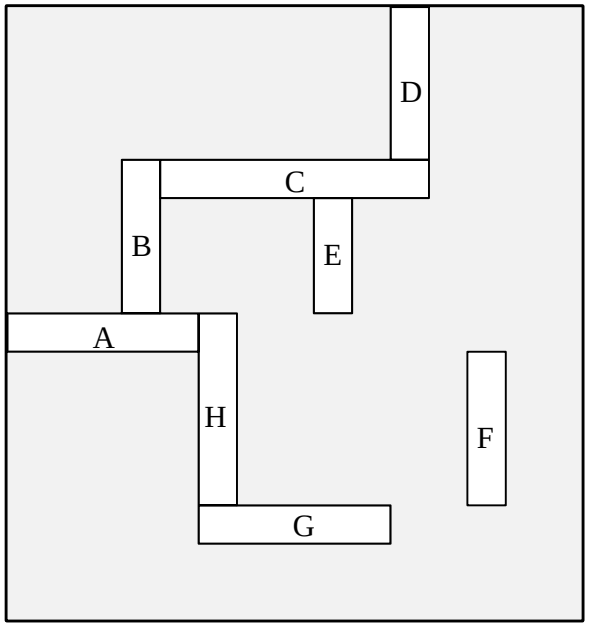


(a)

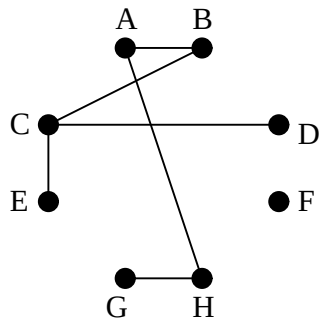


(b)

Neighborhood Graph

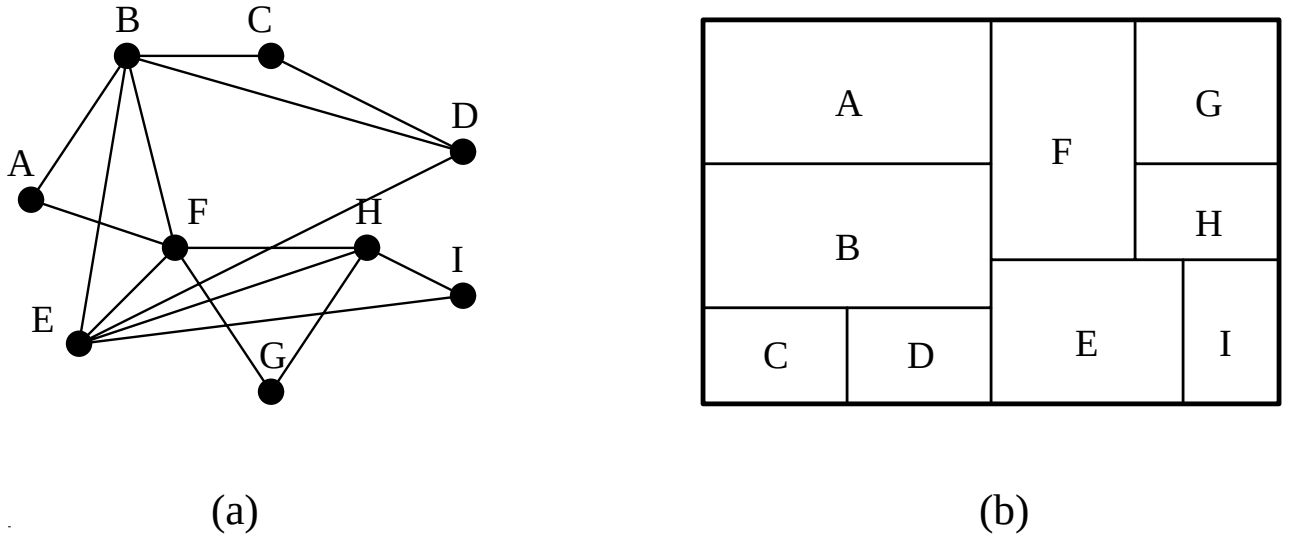


(a)



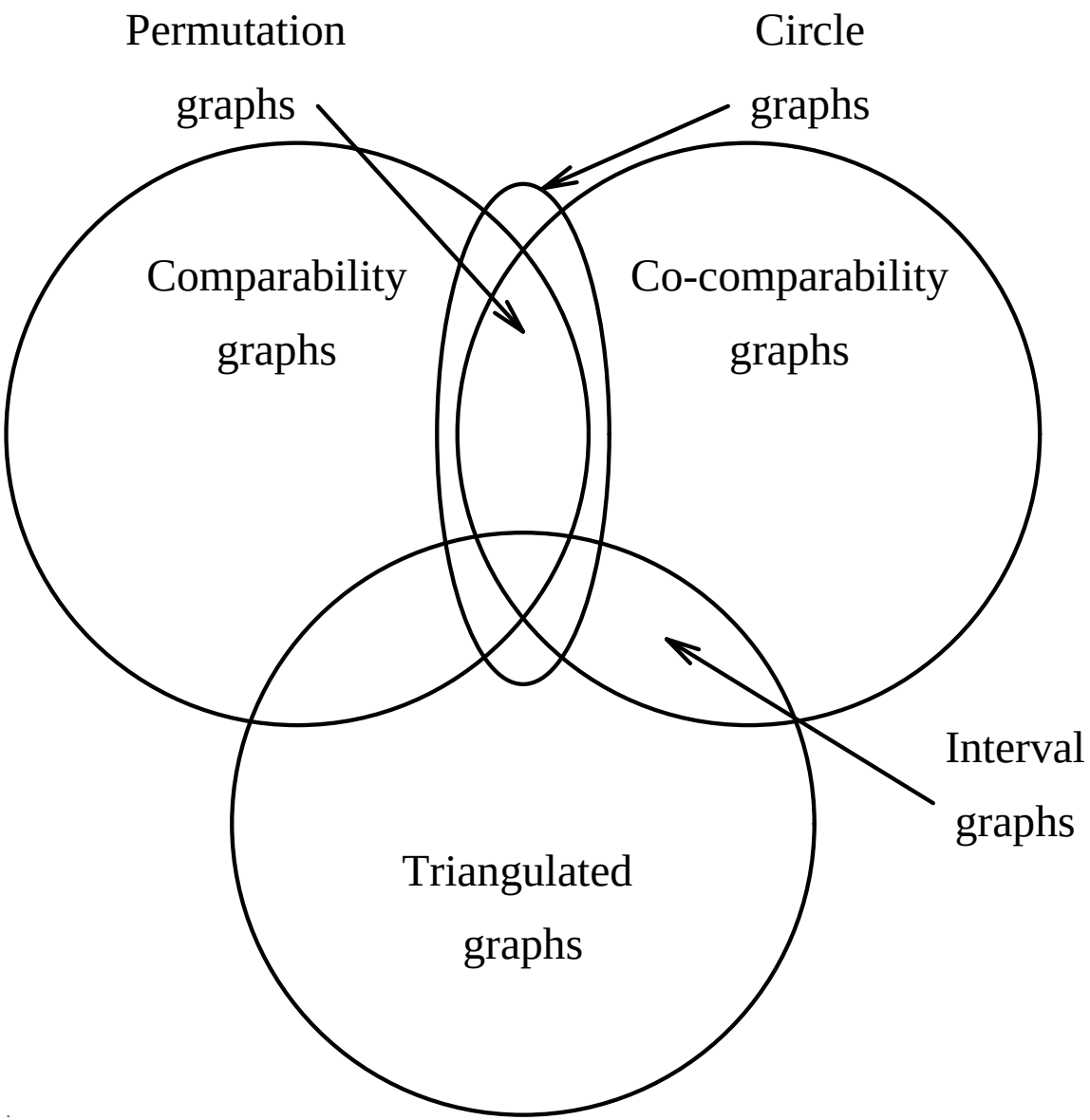
(b)

Rectangular Dual



(a) A graph (b) The rectangular dual of the graph

Relationship Between Different Graphs



Graph Problems in Physical Design

Independent Set Problem

Instance: Graph $G = (V, E)$, positive integer $K \leq |V|$.

Question: Does G contain an independent set of size K or more, i.e., a subset $V' \subset V$ such that $|V'| \geq K$ and such that no two vertices in V' are joined by an edge in E?

Graph Problems in Physical Design (cont.)

Clique Problem

Instance: Graph $G = (V, E)$, positive integer $K \leq |V|$.

Question: Does G contain a clique of size K or more, i.e., a subset $V' \subset V$ such that $|V'| \geq K$ and such that every two vertices in V' are joined by an edge in E?

Graph Problems in Physical Design (cont.)

Graph K-Colorability

Instance: Graph $G = (V, E)$, positive integer $K \leq |V|$.

Question: Is G K-colorable, i.e., does there exist a function $f : V \rightarrow \{ 1, 2, \ldots, K \}$ such that $f(u) \neq f(v)$whenever $\{u, v\} \in E$?

Algorithms for Special Graphs

- Interval Graphs
  - Maximum independent set can be found in $O(n \log n)$ time.
  - Maximum clique can be found in $O(n^2)$ time by using MAX-CLIQUE.
- Permutation Graphs
  - Maximum independent set can be found in $O(n \log n)$ time.
- Circle Graphs
  - Maximum independent set can be found in $O(n^2)$ time by using MIS.
  - Maximum k-independent set can be found in $O(kn^2)$ time by using MKIS.
  - Maximum clique can be found in $O(kn^2 \log n)$ time.

Interval Graphs

Maximum Independent Set

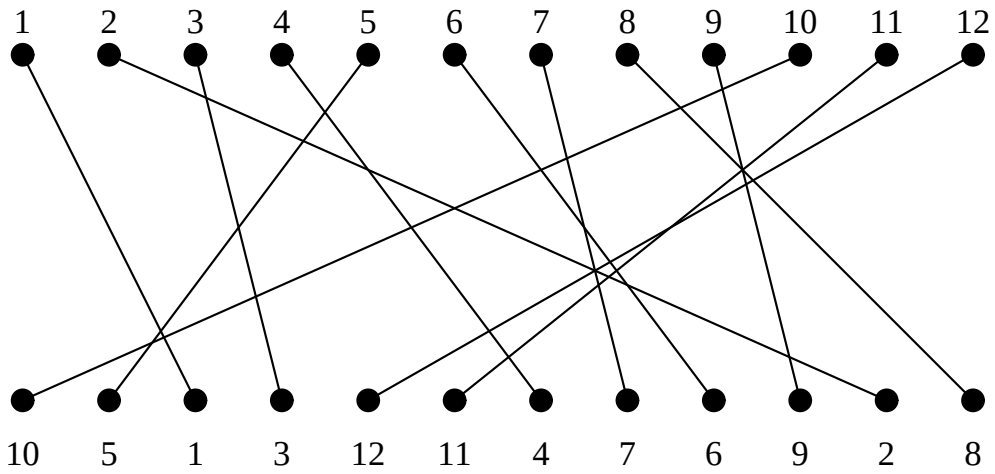
Theorem: Given an interval graph, the MIS can be found in $O(n \log n)$ time, where n is the total number vertices in the graph.

Algorithm MAX-CLIQUE

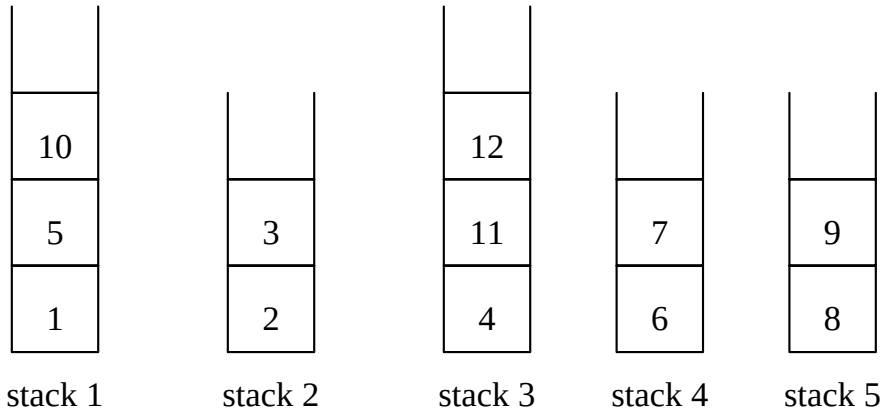
The algorithm finds a maximum clique in a given interval graph.

Algorithm MAX-CLIQUE($\mathcal{I}$)
begin
    SORT-INTERVAL($\mathcal{I}, A$);
    $cliq = 0$;
    $max\_cliq = 0$;
    for $i = 1$ to $2n$ **do**
        if $A[i] =$ L **then** $cliq = cliq + 1$;
            if $cliq > max\_cliq$ **then** $max\_cliq = cliq$;
        else $cliq = cliq - 1$;
    return max_cliq;
end.

Example of Maximum Independent Set in Permutation Graphs



(a)



(b)

Maximum Independent Set Algorithm for Circle Graphs

Algorithm MIS(V)
begin
  for $j = 0$ to $2N - 1$ **do**
    find k such that $kj \in C$ or $jk \in C$;
    for $i = 0$ to $j - 1$ **do**
      if $i \leq k \leq j$-1 and $|M(i, k-1)| + 1 +$
        $|M(k+1, j-1)| > |M(i, j-1)|$ **then**
        $M(i, j) = M(i, k-1) \cup \{v_{kj}\} \cup$
          $M(k+1, j-1)$;
      else $M(i, j) = M(i, j-1)$
end.

Theorem: The algorithm MIS finds a maximum independent set in a circle graph in time $O(n^2)$.

Algorithm MKIS for Circle Graphs

Algorithm MKIS $(G(V, E), k)$
begin
1. $V' = V$;
2. **for** $i = 1$ to k **do**
   $S_i = \text{MIS}(V')$;
   $V' = V' - S_i$;
3. return $S_1 \cup S_2 \cup \ldots \cup S_k$;

end.

MKIS for Circle Graphs (cont.)

Theorem:

Let γ_k be the performance ratio of the algorithm MKIS for k-MIS. Then,

$$\gamma_k \geq 1 - (1 - \frac{1}{k})^k$$

Corollary: Given a circle graph G, MKIS can be used to approximate a maximum bipartite set of G with a performance bound of at least 0.75.

Corollary: For any integer k, the performance ratio of the algorithm MKIS for k-MIS is at least $\gamma_k \geq 1 - e^{-1} \approx 63.2\%$.

The Maximum Clique Problem for Circle Graphs

Given a circle graph $G = (V, E)$, it is easy to show that for every vertex $v \in V$, the induced subgraph $G_v = (V_v, E_v)$ is a permutation graph, where,

$$V_v = \{v\} \cup \{Adj(v)\}$$
$$E_v = \{(u, v) | u \in V_v\}$$

For each G_v, maximum clique can be found using the algorithm presented before.

Summary

1. A VLSI layout is represented as a collection of tiles on several layers.
2. The underlying data structures for layout editors should enable editing operations to be fast.
3. The most popular data structure is corner stitch.
4. The main limitation of all the existing data structures is that they only work on rectangular objects.
5. Due to sheer size of VLSI circuits low time complexity is necessary for algorithms to be practical.
6. Due to NP-hardness of many problems, heuristic and approximation algorithms play a very important role in physical design automation.
7. Several special graphs are used to represent VLSI layouts. The study of algorithms of these graphs is essential to development of efficient algorithms for various phases in VLSI physical design cycle.