Chapter 1

Introduction

1.1 Graph algorithms

One of the most important way to solve many real-world problems is graph-theoretic modelling approaches. Unfortunately, it is unknown and we can't believe that most graph-theoretic problems, modelled from real life, have efficient algorithms as most of the general graph-theoretic problems belong to NP-class on. But those problems are polynomially solvable on special class graphs such as trees or IntGs or other special type of graphs, especially using a dynamic programming approach. Finding efficient algorithms is another powerful way to cope with the hardness of these problems.

Graph is a mathematical tool or object consisting of two finite sets, V of *vertices* and E of *edges*, which is denoted by $G = (V, E)$, where each element of set E is represented by an unordered pair of elements of set V. Usually, the graph is a representation of some problems, using which some objectives of the problems can be solved with the help of some theoretic algorithm(s). So, concerning some real life problems, real world entities or objects can be denoted by vertices and relationship among the entities can be denoted by edges, where the relationship is binary in nature. Hence, a real world (difficult) problem can thus be visualized through its graphical representation, keeping some objective(s) of the problem to be solved. Graphs are essential tools for understanding and solving problems in diverse disciplines. Applications of graphs have very wide range and cannot probably be listed exhaustively. Graphs arising out of real life problems are often very large in size and cannot be processed without the aid of computers. Most of the practical problems which can be modelled with the help of graph theory are found to be large graphs that are virtually impossible for hand computation. Large number problems that hitherto were of academic interest only are suddenly being solved by computer, and their solutions are applied to practical situations.

As is the case with all combinatorial problems, the manipulation and analysis of graphs and subgraphs is essentially non-numerical. That is, in graph-theoretic programs it is primarily the decision-making ability of the computer that is used rather than its ability to perform arithmetic operations.

In literature, many types of graphs are considered. In the field of application, the *perfect graph* is a very fruitful graph. In 1960s, Claude Berge introduced simplified form of perfect graph. Since that time many classes of graphs, interesting in their own right, have been shown to be perfect. Research, in the mean time, has proceeded along two lines:
(i) the first line of investigations includes the proof of the perfect graph theorems, studies of critically imperfect graphs and attempts at proving the strong perfect graph conjecture, and
(ii) the second line of approach concentrates to discover mathematical and algorithmic properties of special graphs like ComGs, CorGs, InvGs, PerGs, CirGs and TraGs, to name just a few.

1.2 Computational complexities of algorithms

An algorithm is characterized by various number of operations and some memory which is required to compute the result depending on the size of the input. The characteristics of the algorithm determine the meaning of complexity of the algorithm. Specifically, the complexity of an algorithm is determined by counting the number of operations and memory usage required to complete the program scales with the size of the input of the program. Technically, length in some encoding of the problem is the size of a problem instance. Measurement of the size of a graph problem by the number of vertices enough for our purposes. Many computer scientists have grouped various problems into the following complexity classes:
i) In the worst case **P** is a class of problems for which the recording time of the given algorithm is some polynomial function of the size of the input. For example, if an algorithm having an input size of 10 bits took 104 operations to calculate the result, then it is a polynomial time algorithm.
ii) **NP** is not deterministic polynomial time class. In NP a candidate for an answer to a problem can be expressed as a right answer or not in polynomial time.
iii) In NP, if a polynomial-time algorithm is used for construction of a polynomial-time algorithm for every problem, then the problem is **NP-hard**.
iv) If a problem is NP-hard and also belongs to NP, then it is **NP-complete** i.e. NP-complete class is the set of problems such that if any member is remained in P, then the set P becomes the set NP. A detail discussion about NP-completeness is available in [47].

There exist a huge number of NP-complete problems in graph theory. Some instances of NP-complete problems are p-center problem, independent set problem with maximum weight, dominating set problem with minimum weight, etc.

Heuristics and exhaustive search can be applied to solve NP-complete problems. A heuristic is an algorithm that can find all-feasible solutions, but it does not guarantee an optimal solution. Exhaustive search always results in exact solution at the risk of infeasibility to run it on instances of more than moderate size.

1.3 Graph theoretic definitions and notations

Here, we present some definitions of basic terms and notations relating to the graph theory and introduce other terms when necessary. Most of the graph theoretical terminologies used in this thesis are in conformity with those given in [9, 49]. We consider a graph G with vertex set V and the edge set E, where $|V| = n$ and $|E| = m$. Usually, the members of V are known as vertices or nodes and these are labelled by $1, 2, \ldots, n$ or $v_1, v_2, \ldots, v_n$ and $E = \{(u, v) : u, v \in V\}$. For an edge (u, v), the vertices u and v are known as *end vertices*. For an UndG, the pairs of the set E are unordered, i.e. $(u, v) = (v, u)$, but for a directed graph these pairs are order pair, i.e. $(u, v) \neq (v, u)$. For many applications there is often a positive real number, called a *cost*, which is attached to each edge. Such a graph is called a *network*.

If $(u, v) \in E$, then u and v are adjacent to each other, and if $(u, v) \notin E$ then u, v are called non-adjacent vertices. We denote $deg(u)$ as the degree of the vertex u, i. e., $deg(u) = |\{v : (u, v) \in E\}|$. If v is the common end vertex of some edges, then these edges are known as *adjacent edges*. The sets $N(u) = \{u|(u, v) \in E\}$ and $N[u] = N(u) \cup \{u\}$ are represent respectively the *open* and *close neighborhood* sets of the vertex $v \in G$. If some edges of E contains same end nodes then these edges are known as *parallel* edges. If two end nodes of an edge are same then that edge is known as *loop*. A vertex whose degree is zero, is known as *isolated vertex*. If a graph has neither loops nor parallel edges, is known as *simple graph*. If a graph allows loops and/or parallel edges/multiple edges, then it will be known as *pseudograph*. We presume all graphs in our thesis as finite, simple and undirected.

If the degree of each node of a graph is zero, the it is known as *Null graph or trivial graph*. In a *regular* graph, the degrees of all vertices are same. If each node of a simple graph is adjacent with other nodes then this graph is known as *complete graph*.

If $H(V', E')$ be a graph such that $V' \subseteq V$ and $E' \subseteq E$, then H is called a *sub-graph* of $G(V, E)$. For any subgraph H of a graph G following results holds:

(i) Each graph is a sub-graph of itself; (ii) A sub-subgraph of G is a sub-graph of G;

(iii) Every node of G is a sub-graph of the same graph;

(iv) An edge with its end nodes in G is a sub-graph of G.

If $H(V', E')$ be a graph such that $V' \subseteq V$ and $E' = \{(u, v) \in E(G) |\ u, v \in V(H)\}$, then H is known as an *induced sub-graph* of G.

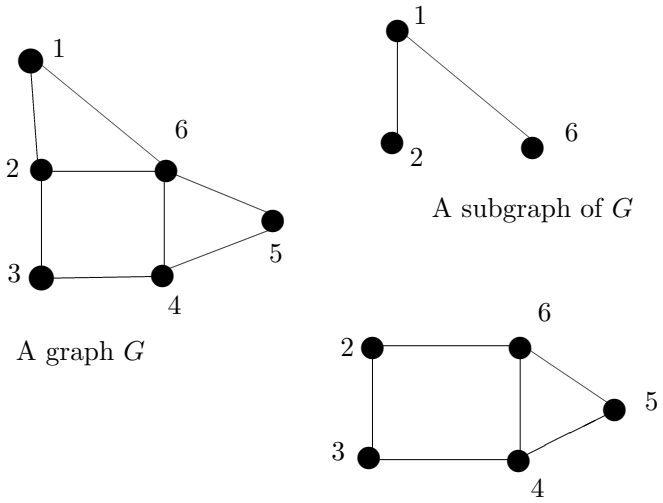


Figure 1.1: Examples of subgraphs.

Generally, K_n represents a *complete graph* or a *clique* with n nodes and is called a *clique* of size n.

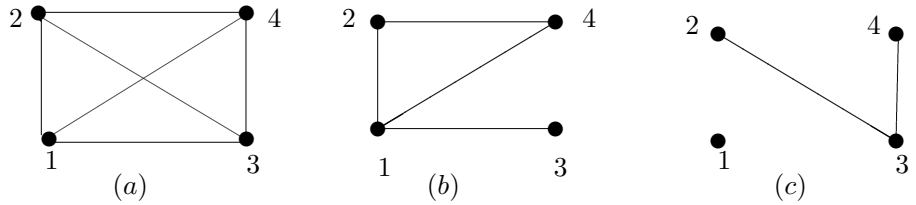


Figure 1.2: (a) The graph K_4, (b) is the complement of (c) and vice versa.

A finite chain of nodes and edges is known as a *walk*, where the chain starting and ending with nodes only. vertices may repeat in a walk but not edges. Usually, walks are of two types; open walk and close walk. If starting and ending nodes of a walk are same then it is called a *closed walk*, otherwise it is said to be an *open walk*. An open walk in which no vertices are repeated is known as a *path*. The number of edges in a path is called length of that path. It is to be noted that a path contains n vertices has only $n - 1$ edges. A closed path is known as a *cycle*.

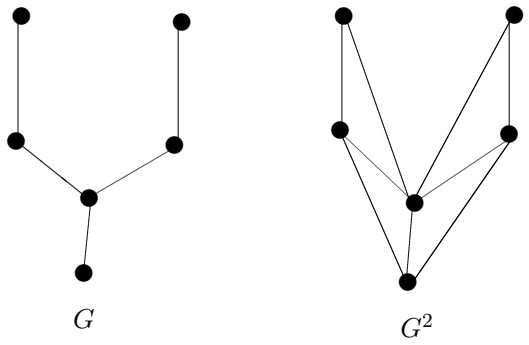


Figure 1.3: A graph and its square.

For an unweighted graph, a path between any two specified nodes contains minimum number of edges is known as the *shortest path*. We use $d(u, v)$ as the shortest distance or distance between the nodes u and v. A path whose length just greater than the length of the shortest path between any two specified nodes is known as the *Next-to-shortest path*.

If every pair of nodes of a graph are joined by at least one path, then that graph is known as a *connected graph*, otherwise it is called a *disconnected graph*.

The *eccentricity* of a node $v \in G$ indicates the maximum length among all the paths from v to other nodes. Usually, eccentricity of v is denoted by $e(v)$. We denote radius of a graph G by $radius(G)$, where $radius(G) = infimum\{e(x) : x \in V\}$. Furthermore, we set diameter of a graph G as $diameter(G)$, where $diameter(G) = maximum\{e(x) : x \in V\} = max\{d(x, y) : x, y \in V\}$. It is to be kept in mind that $radius(G) \leq diameter(G) \leq 2\, radius(G)$.

A tree T is called a *spanning tree* of a connected graph G if T is cycle free and $V(T) = V(G)$.

A sub-graph H of a graph G is known to be a *spanning sub-graph* of G if $V(H) = V(G)$. In this context, we can say that a *spanning tree* of a graph G is also a spanning sub-graph of G. For a spanning rooted tree T of a graph G is called a *depth-first search (DFS) tree* if for every edge $(x, y) \in E$, one of the two nodes incident with the edge (x, y) is an ancestor of some other edge in T. On the other hand T is called a *(BFS)-tree* if $\forall$ edge $(x, y) \in E$, the difference of the levels of the two vertices incident with (x, y) is no more than 1 in T, where the *level* of a node v in T is the distance from the root to the node v. We express $diameter(T)$ as the diameter of T, where $diameter(T) = maximum\{d(x, y) : x, y \in V(G)\}$. A spanning tree with least diameter is known as *minimum diameter spanning tree*.

A set $S \subseteq V$ is called an *independent set* or *stable set* of a graph $G = (V, E)$ if all vertices in S are not adjacent to each other. An independent set with the maximum cardinality is known as *maximum independent set* or *largest independent set*. The cardinality number of

largest independent set of G is known as *stability number* of G and it is expressed by $\alpha(G)$.

Two vertices u and v are called *adjacent* to each other if $(u, v) \in E$. A vertex v is dominated by a subset D of V if v is adjacent with at least one vertex in D. A set D of G is said to be a dominating set G if each vertex of $V - D$ is adjacent to at least one vertex of D. For a graph G, a dominating set D with fewest cardinality among all dominating sets is known as *minimum dominating set*. A dominating set is called *independent* if there is no edges between the vertices of D. A dominating set D is called *total* if each node of V is adjacent to at least one node in D. On the other hand a dominating set is known as *connected* if there exists at least one path between every pair of nodes in D. If a subgraph formed by a connected dominating set of G then it will be known as *connected dominating path*. A connected dominating path with minimum length is called *minimum connected dominating path*. The order of a minimum cardinality dominating set in a graph is familiar as *domination number*. A subset D of V is called a *k-tuple dominating set* of a graph G with vertex set V and edge set E if each node of $V - D$ is adjacent with at least k nodes in D. We use $\gamma_{\times k}(G)$ to represent k-tuple domination number which is the order of a minimum k-tuple dominating set of G.

In weighted tree $T = (V_T, E_T)$, we represent eccentricity of a node x by the symbol *eccentricity e(x)*, where $e(x) = \max\{d(x, v_i)$, for all $v_i \in V_T\}$ and $d(x, v_i)$ indicates the total weights of the edges lie on the path between x and v_i.

A *center* of a tree T is a node with minimum eccentricity in the same tree, i.e. if $e(x) = \min\{e(y)$, for all $y \in V\}$, then x is known as the *1-center*. It is to be noted that a tree is either mono-centric or bi-centric.

The radius of a tree T is the eccentricity of a center of that tree and we denote it by $\rho(T)$, i.e. $\rho(T) = \{\min_{x \in T} e(x)\}$.

The length of the farthest path in a tree T is known as the diameter of the given tree i.e. the highest eccentricity indicates the diameter. Let the weighted tree T with $(n+1)$ vertices and n edges. In a tree T, the Inv1C problem is a graph theoretical problem where we change weights associated with the edges in T under some certain limitations in such a way that a pre-specified node becomes 1-center of T.

In a graph $G = (V, E)$ with n nodes, the *average distance* of G, denoted by $\mu(G)$ represents is the average of the distances between all unordered pairs of nodes in G,

$$\mu(G) = \frac{2}{n(n-1)} \sum_{x,y \in V(G)} d_G(x, y)$$

A spanning tree T of a graph G having minimum average distance is known as a MADST of G.

1.4 Some special graphs and related definitions

In real life, many problems can be modelled using graphs with some special properties. A graph having some special properties is called a *special graph*. Some important special graphs are tree, perfect graphs, IntGs, InvGs, TolGs, PerGs, TraGs, BipGs, CirGs, circular PerGs, circular TraGs, series-parallel graphs, CorGs, star graphs, disk graphs, asteroidal triple-free graphs, i-triangulated graphs, ComGs, CacGs and many others.

Let $\mathcal{S}$ be a family of non-empty sets. Also, we assume that $\mathcal{S}$ is finite. Now, a UndG G having set of vertices V and set of edges E is known to be IntG defined on $\mathcal{S}$ if each node of G is formed for each member of $\mathcal{S}$ and two nodes are adjacent iffy their corresponding sets in $\mathcal{S}$ have non-empty intersection. Usually $\mathcal{S}$ represents an intersection model of the IntG G and $G(\mathcal{S})$ is used to represent the Int defined on $\mathcal{S}$. The characterization of IntGs can be done by several ways and it has lots of applications on real life. Depending on the collection of the sets $\mathcal{S}$, one can define a variety of IntG classes.

Tolerance graphs (TolGs)

Suppose $G = (V, E)$ be an UndG, where $V = \{u_1, u_2, \ldots, u_n\}$. Now, G is known to be a TolG if each node u_i of V is assigned with a closed interval I_1 in a family of closed intervals $I = \{I_1, I_2, \ldots, I_n\}$ on a line and a tolerance t_1 in a family of positive numbers $T = \{t_1, t_2, \ldots, t_n\}$ such that two nodes x and y are adjacent iffy $|I_i \bigcap I_j| \geq \min(t_i, t_j)$, where $|I_i|$ represents the length of I_i. Usually the *tolerance representation* of a graph G is denoted by pair (I, t). Furthermore, this representation (I, t) is known to *bounded* if $t_v \leq |I_v|$ for all $v \in V$. Again, a TolG is called a *bounded TolG* if its tolerance representation is bounded. If we restrict all the tolerances t_u to be equal to any fixed positive constant c, then we obtained exactly the class of InvGs. If we restrict the tolerances such that $t_v = |I_v|$ for all vertices v, then we obtain exactly the class of PerGs.

Circular-arc graphs (CirGs)

Suppose $G = (V, E)$ be an UndG. Now, G is known to be a CirG if we create each node of V corresponding to each arc in the set of arcs S defined on a circle such that $(x, y) \in E$ iffy the arcs corresponding to the nodes x and y have common intersection. We use $\mathcal{S}$ as a circular-arc representation or circular-arc model of G. A CirG is known to a *proper CirG* if there is no arc properly contains the other arcs in its representation.

Other graphs

Suppose G= (V, E) be an UndG. Now, G is known to be a χ-perfect if $\omega(G_H)$= $\chi(G_H)$, and G is known to be a α-perfect if $\alpha(G_H)$= $k(G_H)$, for all $H \subseteq V$, where G_H is a induced subgraph formed by the subset H of vertices and $\omega(G), \chi(G), \alpha(G)$ and $k(G)$ represent respectively the clique number, the chromatic number, the stability number and the clique cover of G. If a graph is either a χ-perfect or a α-perfect then it is known to be a *perfect* graph.

A graph is known to be a *polygon-circle graph* if it is an Int of a family of polygons whose corner points lies on a circle.

A graph $G(V, E)$ is known to be a *unit disc graph* if we create each node of V corresponding to each disc in the set of discs S defined on the Euclidean plane such that $(x, y) \in E$ iffy the discs corresponding to the nodes x and y have common intersection.

A *line graph* of a graph $G(V, E)$ is an Int of a family of edges S in G, where we create each node of the line graph corresponding to each edge in S.

If a graph can be drawn in a surface S in such a way that every pair of edges do not intersect in that surface then it is known to be a *embedded* graph. An embedded graph drawn in a plane is familiar as *planar graph*.

If there is no cycle of length four or more in a graph G then it is known as *triangulated* graph. It is also known as CorG. Generally, there are two types of CorGs, one is strongly CorGs and other is weakly CorGs. As their names indicate, every strongly CorG is chordal, and every weakly CorG is chordal. Weakly CorGs are also perfect graphs.

Strongly CorGs, introduced by Farber [40], specialize CorGs in several ways. They are characterized by several equivalent definitions utilizing chords in a cycle, removal orderings and forbidden subgraphs.

Suppose C= $u_1, u_2, \ldots, u_{2k}, u_1$ be a cycle whose length is $2k \geq 6$. We mention a chord $(u_l, u_k) \in E$ as an *odd chord* if only one among l and k is even, i.e. it provides two cycles of even length in C. If a graph G is chordal as well as its all cycles whose lengths six or mores contains an odd chord.

A graph $G = (V, E)$ is known to be *split* the vertex set V can be partitioned into two subset H and B among which one is stable set and other forms a clique.

Furthermore, if the set of vertices V of a graph G can be divided into two subset H and B each be a independent set then that graph is familiar as BipG, i.e. the extremities of each edge among which one is in H and other is in B. That BipG is usually represented by (H, B, E).

A BipG $G = (H, B, E)$ is known to be *complete BipG* if each member of H is adjacent

with all members of B. The complete BipG $G = (H, B, E)$ is represented by $K_{l,k}$, where $|H| = l$ and $|B| = k$.

Some special types of graph are displayed in Figure 1.4 and Figure 1.5.

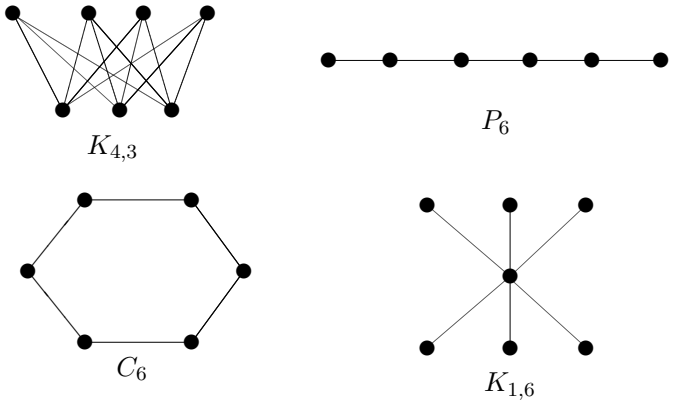
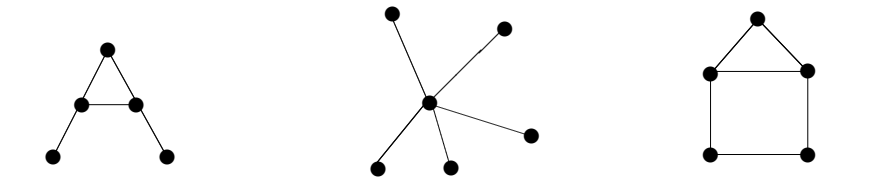
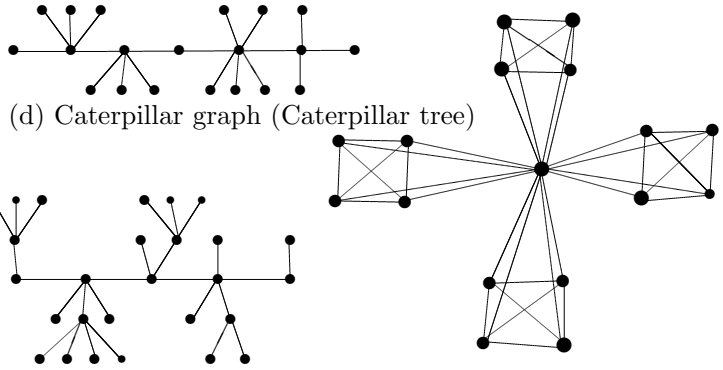


Figure 1.4: Examples of $K_{4,3}$, P_6, C_6 and $K_{1,6}$.



(a) The A-graph (b) The Star graph (c) The house graph



(d) Caterpillar graph (Caterpillar tree)

(e) Lobster graph (f) Windmill graph $W_d(5, 4)$

Figure 1.5: Some special types of graphs.

Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two graphs. Now, if we can define a one-one and onto mapping $g : V_1 \to V_2$) so that whenever $(x_1, y_1) \in E_1$ then $(g(x_1), g(y_1)) \in E_2$, then G_1 and G_2 are known to be isomorphic to each other. If G_1 and G_2 are isomorphic to each other then we can write $G_1 \cong G_2$.

A graph will be familiar as a ComG if this graph allows transitive orientation. In other words, if we assign direction to each edge of a given UndG then the directed edges (v_x, v_y) and (v_y, v_z) in arising directed graph implies that (v_x, v_z) in the same directed graph. It is to be kept in mind that The class of ComG is a proper subclass of perfect graphs.

A graph is familiar to be a *Co-graphs* if it can be reduced to single node by repeatedly complementing all connected subgraphs. PerGs is superclass of Co-graphs.

We define an undirected as a CacG if its each block is an link/edge or a cycle.

The *clique graph* of a graph G is symbolled by $C(G)$ and it is defined as the IntG of the collection of all cliques of the given G.

Some vital relations among several special class graphs are written as below.

(i) {BipGs} $\subseteq$ {ComGs},

(ii) {split graphs} $\subseteq$ {CorGs},

(iii) {InvGs} $\subseteq$ {directed PatG} $\subseteq$ {undirected PatG} $\subseteq$ {CorGs},

(iv) {InvGs} $\subseteq$ {strongly CorGs} $\subseteq$ {CorGs},

(v) {cographs} $\subseteq$ {PerGs} $\subseteq$ {ComGs}.

1.5 The graphs under consideration

In this thesis we have consider three special types of IntGs which are InvGs, PerGs and TraGs. All graphs considered in this work are simple, connected, undirected and finite.

1.5.1 Interval graphs

Suppose $G = (V, E)$ be an UndG. Now, G is known to be a InvG if we create each node of V for each interval in the set of intervals I defined on real line so that $(x, y) \in E$ iffy the intervals associated to the nodes x and y have common intersection. We use the symbol I to represent an IR of G and G to represent an *IntG* of I [50]. Let $I = \{j_1, j_2, \ldots, j_n\}$, where $j_z = [a_z, b_z]$ for $1 \leq z \leq n$, be the IR of the graph G, where a_z represents the left extremity and b_z is the right extremity of the interval i_z. Without loosing the generality, we presume the following notations [50]:

(i) each interval has both its extremities and two or more intervals do not have a same extremely,

(ii) intervals in IR and nodes in InvG are same,

(iii) our assumed InvG is connected & the array of arranged extremeties/endpoints is provided and

(iv) the intervals in I are numbered according to the ascending/growing right extremities.

If two or more intervals contain same extremities we apply the Algorithm CONVERT [82] to convert the given intervals into the intervals with different extremities.

Here we take the weighted InvGs, i.e. for each interval j, we assign a positive weight $w_j > 0$.

An InvG and its interval representation are displayed in Figure 1.6.

Many researchers studied InvGs and they used this graph as the mathematical model to solve several real life problems. A brief discussion about InvGs was found in [50, 81, 83, 84, 72, 91, 87]. To the best of our knowledge InvGs and its different subclass have lots of applications in archeology, protein sequencing [56], works scheduling [18], macro substitution [39], VLSI design, file arrangements [18], psychology, transportation, routing between two points nets [52], circuit routine [79, 57], genetics, molecular biology, sociology, circuit routing etc. An InvG and its interval representation are shown in Figure 1.6.

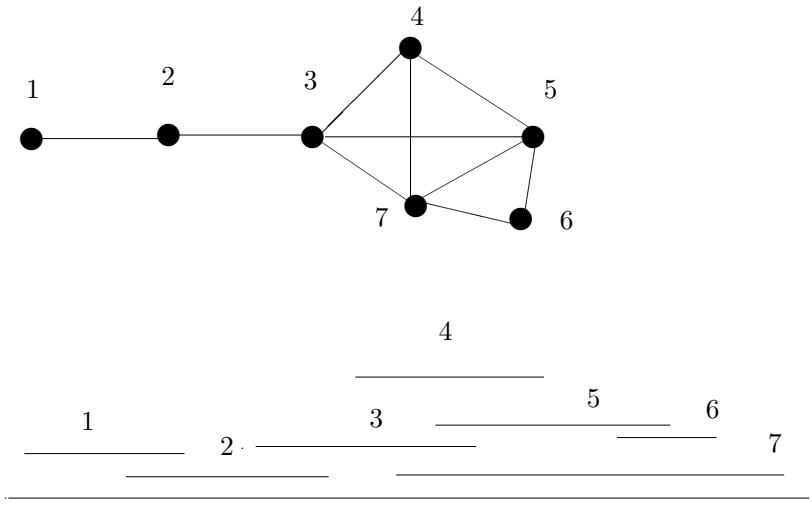


Figure 1.6: An InvG and its IR.

1.5.2 Permutation graphs

Suppose $G = (V, E)$ be an UndG , where $V = \{1, 2, \ldots, n\}$, $|V| = n$ and $|E| = m$. Now, the graph G is familiar as a PerG if and only if there is a permutation $\pi = \{\pi(1), \pi(2), \ldots,$

$\pi(n)\}$ on the set V of nodes such that $\forall\ i_1,\ i_2 \in V$, $(i_1,\ i_2) \in E$ iffy

$$(i_1 - i_2)(\pi^{-1}(i_1) - \pi^{-1}(i_2)) < 0,$$

, where for every $i_1 \in V$, $\pi^{-1}(i_1)$ indicates the location of the number i_1 in π [49]. Here, we consider a connected PerG. PerGs can be displayed as a subclass of IntGs [49]. Furthermore, PerGs are the subclass of ComGs [86]. Also, PerGs can be revealed by the MchD, in which two horizontal parallel lines, named as top line and bottom line are exist.

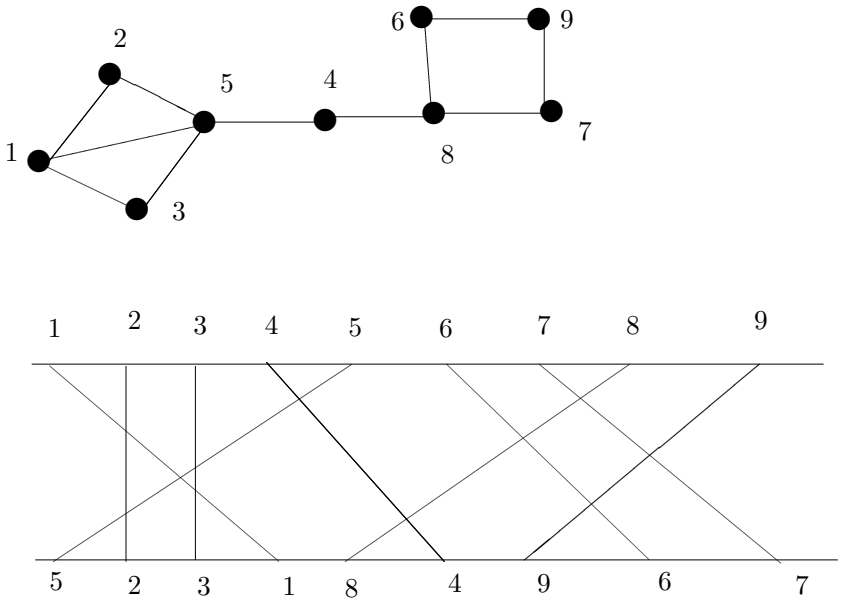


Figure 1.7: A PerG and its PerD.

We place the members of V on the top line, in ascending order and for each $i_1 = 1,\ 2,\ \ldots,$ n place the permutation number $\pi(i)$ on the bottom line just under the number i_1 on the top line. After then, we join two i_1's situated on the top line and on the bottom line by drawing a line segment between them, for each $i_1 = 1,\ 2,\ \ldots\ n$ [49]. We also label a drawn line segment by i_1 if it is drawn by joining two i_1's. Beside these, we create each node of the PerG for each line segment in the MchD. If two line segments i_1 and i_2 cuts each other in the MchD then $(i_1, i_2) \in E$. The converse is true also. A PerG and its corresponding permutation representation are displayed in Figure 1.7. In the MchD $\pi^{-1}(i_1)$ yields the position number of i_1 on the bottom line, for instance, $\pi^{-1}(4)$ yields the position of the number 4 on the bottom line which is here 6th. In [7, 88], we get brief discussion about PerG.

1.5.3 Circular-arc graphs

Suppose $G = (V, E)$ be an UndG. Now, G is known to be a CirG if we create each node of V for each arc in the set of arcs S on the circumference of a circle C such that $(x, y) \in E$ iffy the arcs corresponding to the nodes x and y have common intersection. We use the symbol S to represent an *arc representation* of G and G to represent an *CirG* of S. CirG is a special kind of IntG.

Suppose $S = \{A_1, A_2, \ldots, A_n\}$ be a collection of n arcs situated on the circumference of a circle C. Each arc has two extremities/endpoints, the starting extremity is known as head and the terminal extremity is known as tail, when it is travelled in clockwise manner. We assign a positive integer to each endpoint of the arcs. These integers are called the coordinates of the respective endpoints of the arcs. For instance, we represent each arc A_i, $i = 1, 2, \ldots, n$, by (h_i, t_i), where h_i indicates the head and t_i indicates the tail, starting with a fixed point on A on the circumference of C.

CirGs have a vital role in genetic research [95], traffic planning [96], compiler design, operation research, etc. A brief discussion about CirG was found in [64, 65, 66, 67, 85]

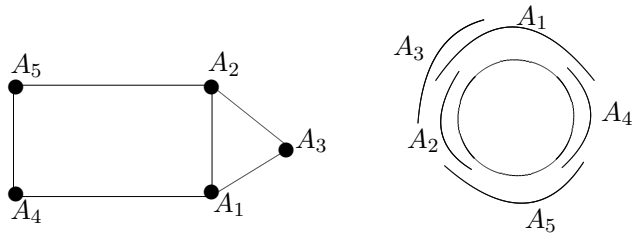


Figure 1.8: A CirG and its CirD.

1.5.4 Trapezoid graphs

A TraG is a special kind of Int. This graph can be displayed by a TraD which contains two horizontal parallel lines, one is known as top line and other is known as bottom line. In TraD, a *trapezoid* T_j is represented by four corner points $[a_j, b_j, c_j, d_j]$, where a_j (called as left endpoint) and $b_j (> a_j)$ (called as right endpoint) lie on the top line and c_j (called as left endpoint) and $d_j (> c_j)$ (called as right endpoint) lie on the bottom line in TraD. Suppose $T = \{T_1, T_2, \ldots, T_n\}$, be a collection of the n trapezoids. Let $G = (V, E)$ be an UndG, where $|V| = n$, $|E| = m$ edges and $V = \{1, 2, \ldots, n\}$. Now, G is known to be a TraG if G is a Int of T in TraD, i.e. if we create each node j of G for each trapezoid T_j and two nodes x and y are adjacent iffy there associative trapezoid T_x and T_y intersect each other in TraD [26].

It is to be kept in mind that two trapezoids T_x and $T_y(> x)$ intersect if and only if either $(a_y - b_x) < 0$ or $(c_y - d_x) < 0$ or both. A TraG and its corresponding TraD are displayed in Figure 1.9.

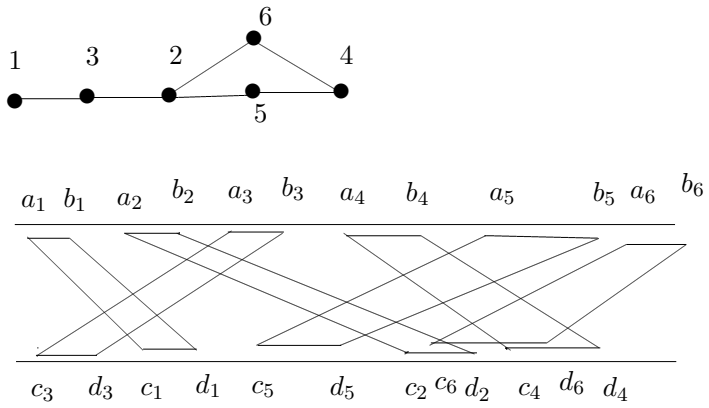


Figure 1.9: A TraG and its corresponding TraD.

Here we consider a connected graph $G = (V, E)$. Without loosing the generality of TraGs we presume the following matters :

(i) each trapezoid has only four unique corner points

(ii) there is a one to one relation between the trapezoids in the TraD and nodes in the TraG,

(iii) we label the trapezoids in TraD T according to the increasing right end points on the top line, i.e. $T_1 < T_2 < T_3 < \cdots < T_n$ iffy $b_1 < b_2 < \cdots < b_n$.

TraGs are the superclass of PerGs and InvGs [49] and subclass of CcoGs [60]. Recognition of a TraGs can be done in $O(n^2)$ time [62]. This graph plays a vital role in graph theory. We first found TraGs in [24, 26]. After then it was researched deeply in [24, 26]. This graph has several variation like circular TraG and circle TraGs [61]. Besides these a brief discussion on TraGs are available in [22, 97, 42, 59, 48]. TraGs help us to construct model of channel routine problem in a single-layer-per-net model. This graphs also have several applications in VLSI design, bio-informatics, etc. [1, 26].

1.5.5 Fuzzy interval graph

In graph theory, researchers usually consider a collection of sets of intervals which are in linear ordered to discuss about IntGs. This types of InvGs has lots of applications in real life.

It is to be kept in mind that different collection of sets have the same Int in both cases (crisp and fuzzy). Specially, the characterization of intersection properties of a finite collection of intervals (real) with fuzzy numbers can be done with the help of a collection of fuzzy intervals

stated on a finite set. For this reason we focus our attention to fuzzy intervals having finite support.

We consider a linearly ordered set V. It is well known that a fuzzy interval usually denoted by the symbol τ on V is normal a swell as convex fuzzy subset of V. In other words, there is an $u \in V$ such that $\tau(u) = 1$ and the ordering $a \leq b \leq c$ implies that $\tau(b) \geq \tau(a) \wedge \tau(c)$. We also consider a fuzzy number as a fuzzy interval. A fuzzy InvG is nothing but a fuzzy Int of a set which is a finite collection of fuzzy intervals. The node set of a fuzzy InvG is crisp according to the normality of the fuzzy intervals.

1.6 Motivation of the work

There are several sequential algorithms which are used for solving the graph theoretic problems. These algorithms are very carefully designed by researchers. Among these general graph problems, most problems belong to NP-class and these problems can be solved polynomially by some particular classes of IntGs like InvGs, PerGs, CirGs, TraGs etc. Therefore, if any problem can be modelled on any one of InvGs, PerGs or TraGs then the problem can be solved more efficiently, some times even optimal on sequential and parallel computing systems. The graphs can be represented as intersections of "lines" or "arcs of concentric circles" or "trapezoid" which are the central parts of solving problems, are associated to frequency allotments in radio networks and also for computing molecular compliances, etc. An InvG, a PerG, a CirG and a TraG containing n nodes and m edges can be stored in the memory of a computer using $O(n)$ space only, instead of $O(m + n)$ or $O(n^2)$ or more space for arbitrary graphs.

Inv1C problem is the variant of an inverse shortest path problem. For instance, we can predict the movements of earthquakes by applying Inv1C concept in geologic areas. For this purpose we separate geological areas into a number of cells. Then we create each node for each cell and then we establish the adjacency relations between the nodes(Moser [77]).

Some well known estimations exist for computing the transmission times between the cells, but it is very difficult to find optimal or precise values transmission time. To compute the times required for transmitting the information between the cells it is essential to notice an earthquake. Also, we have to estimate the times to reaching the resulting seismic perturbations at different cells. We also consider that earthquakes move through the shortest paths. This problem is similar to an inverse shortest path problem.

Inv1C problem also helps to deduce the real cost in facility location problems. For instance let's consider that we have a road network and there are some facilities like hospitals, fire

stations, banks, schools, medicine shops, post office, administrative buildings, fuel pumps, parks, etc. Our object is to place/construct these facilities in such a way that the distances between the facilities and the customers is minimum. Besides this sometimes we see that the facilities are already existed and these can not be relocated as reconstruction or relocation costs are very high. In that situation, we should modify (reconstruct) the road network minimally so that the road construction cost is minimum as well as the distances between the facilities and the customers is minimum do not exceed some given bounds. This is a real example of the inverse center location problem. This problem can be applied in modeling/planning traffic networks, to establish tolls (Dial [33]). Inverse center location problem has also applications in various use of inverse methods in other different fields [38, 68].

MADTs is another vital problem in graph theory. Many researchers mentioned MADTs as minimum routing cost spanning trees. It is mostly used to design communication networks [55]. In that case usually we construct a tree sub-network of a given network in such a way that traversing cost or time one node to any other node is minimum.

Furthermore, above problems has lots of applications in real world like electrical networks, telecommunications, operations research, optimizing compilers for embedded systems, VLSI design etc.

1.7 Survey of related works of the thesis

Here we present a brief and up-to-date literature review of related works done on different graph classes.

As shown in [16, 103, 104], the inverse problems of many combinatorial/network optimization problems can be solved by strongly or weakly polynomial algorithms. It is observes that if the size of combinatorial/network is large and the an optimization problem can be solved in polynomial time, then its inverse problem can also be solved in polynomial time by similar methodology [101]. Heuberger [54] described a brief literature review on inverse optimization problems. In [17], Cai et al. showed that the Inv1C location problem is *NP*-hard on general un-weighted directed graphs, where as the underlying center location problem can be solved in polynomial time. Also, for a fixed non-input parameter p, Burkard et al. proved that inverse p-median location problems is solvable in polynomial time [12]. For $p = 1$,, they also designed an $O(n \log n)$ time algorithm for that problem by modifying the nodes weights on tree networks. After then, Galavi designed an $O(n)$ time algorithm to solve that inverse 1-median location problems [45]. Besides these, Burkard et al. proved that the time com-

plexity of solving the inverse 1-median problem on the plane with Manhattan/Chebyshev is $O(n \log n)$ [12]. In [13], they also studied the same problem on a cycle by modifying the nodes weights and unit cost. The same authors also designed an $O(n^2)$ time algorithm to solve this problem with the help of the methods of computational geometry. After that Gassner have designed an efficient algorithm to solve the inverse 1-maxian problem on tree networks by modifying the edge lengths which runs in $O(n \log n)$ time [44]. In [14, 15], Burkard et al. studied the inverse Fermat-Weber problem. They proposed a combinatorial approach to solves the same problem in $O(n \log n)$ time for unit cost, where they presumed that the pre-selected point which will be a 1-median in future does not concur with a given point in the plane. In [45], Galavii proved that the 1-median on a path having positive or negative weights stays either in one of the nodes with positive weights or stays in starting end point or in the ending end point of the path. Using this property He solved the inverse 1-median problem in $O(n)$ time on a path having negative weights. Moreover, in [46], Gassner proved that inverse version of the convex ordered median problems on general graphs belong to *NP*-hard class. He also showed that this problem is *NP*-hard on trees, even if the weights are all unit or for a K-centrum problem. Gassner also designed an $O(n^3k^2)$ time algorithm to solve the inverse unit-weight K-centrum problem having unit cost coefficients on a tree. In [102], Yang and Zhang discovered a method to solve the inverse vertex center problem on a tree in $O(n^2 \log n)$ time, where they assumed that modified edge lengths always will be positive. The same authors also proved that the inverse vertex center problem can be solved in $O(n^3 \log n)$ time on general graphs. In 2009, Alizadeh et al. solved Inv1C location problem on trees with edge length augmentation in $O(n \log n)$ time with the help of a set of fitted extended AVL-search trees [2]. In [3], Alizadeh et al. proved that inverse absolute problems can be solved in $O(n^2)$ time on trees without allowing the topology and the same problems can be solved in $O(n^2 r)$ time allowing the topology.

At present, many researcher are interested to study IOPs because of its applications in real world. A brief discussion on IOPs can be found in [37, 51, 54, 58].

Usually, the basic optimization models of network location problems are constructed to find the best location of single or multiple new facilities like hospitals, banks, schools, etc. in a network of demand points in such a way that a given function that based on the distance between the facilities and clients becomes minimum. Based upon the above model, during investigation, facilities or clients may either be putted down only at nodes or may also remains on edges of the given network. A brief discussion and illustrations on these problems can be found in [30, 34, 43, 63, 69, 78].

In 1992, Burton and Toint first studied an ISSP [11]. After then many researchers worked independently on IOPs.

In this thesis, we have proposed following problems :
(i) an optimal algorithm to find Inv1C location problem on weighted trees (where weights are assigned to the nodes) which runs in $O(n)$ time, where n is the cardinality of the vertex set V of the tree,
(ii) an $O(n)$ time algorithm to determine Inv1C location problem on weighted InvGs, where n is the order of the vertex set V of the InvG,
(iii) an $O(n)$ time algorithm to obtain Inv1C location problem on weighted CirGs, where n is the cardinality of the vertex set V of the CirG,
(iv) an optimal algorithm to compute Inv1C location problem on weighted PerGs which runs in $O(n)$ time, where $|V| = n.$,
and (v) an sequential algorithm to solve Inv1C location problem on weighted TraGs in $O(n)$ time, where n is the cardinality of the vertex set V of the TraG.

Usually, the problem of constructing a MADT on general graphs belongs to NP-hard class [55]. In [5], we found a polynomial approximation time algorithm for finding MADT. So, a question arises to us that for which particulars sub-classes of graphs, this problem can be solved in polynomial time. In 2003, Dahlhaus et al. have designed a linear time algorithm to construct a MADT of a given distance-hereditary graph [27]. In very recent, the same authors [28] have proposed an $O(n)$ time algorithm to find a MADT of an InvG. In 1997, Barefoot et al. [6] proved that if T is a MADT of a given graph G (which is connected), then there contains a node $c \in T$ such that each path in T begins at c is induced in G. It is still an open problem to us that is there any polynomial time algorithm to construct a MADT of a weighted InvG, where weighted are assigned to the nodes. In 1992, Olario et al., have proposed an optimal parallel algorithms to form a MADT on InvGs [80]. Furthermore, in 2013, Mondal et al. have proposed an $O(n^2)$-time algorithm to form MADST on TraGs [76]. The same authors [73] also, computed all pairs shortest paths between nodes of PerGs in $O(n^2)$-time.

Many authors mentioned MADTs of weighted graphs as minimum routing cost spanning trees [55]. In this problem everyone try to construct a spanning tree sub-network of a graph in such a way that one can arrive every vertex from every other vertex as fast as possible with average cost.

In this thesis, we have designed the following algorithms
i) an efficient algorithm to form a MADT of CirG G having n nodes, which runs in $O(n^2)$

time ,

ii) an efficient algorithm to form a MADT of a fuzzy InvG G having n nodes, which runs in $O(n^2)$ time,

iii) an efficient algorithm to form a MADT of a PerG G having n nodes, which runs in $O(n^2)$ time.

In this thesis, we propose an optimal algorithm to determine Inv1C location problem on weighted TraGs which runs in $O(n)$ time, where $|V| = n$.

1.8 An overview of the thesis

This thesis has been organized into seven chapters. The content of each chapter is presented briefly in the following.

Chapter 1

In this chapter, we present certain graph theoretic definitions and notations and define some relevant special graphs. Characterization of InvGs, PerGs, CirGs, TraGs are given. Motivation of the work also included in this chapter. Also a brief summary of total work carried out is also given. Finally we try to present an up to date survey of related works done on different graph classes.

Chapter 2

In 2nd chapter, we describe *the Inv1C location problem on the weighted trees*. Suppose T be a tree having $(n + 1)$ nodes and n edges associated with positive weights. In the Inv1C problem, we change edge weights in such a way that a pre-specified node becomes the 1-center and the modification (under certain boundary weight restrictions) cost is minimum. Here, we design an $O(n)$-time algorithm to determine an Inv1C location on the weighted trees, where n is the number of nodes.

Chapter 3

In third chapter we narrate *the solution of the Inv1C location problem on the weighted (associated with nodes) InvGs and construction of MADT on fuzzy InvGs*. In an Inv1C location problem, we modify the parameter like vertex weights of an IT T_{IG} of the weighted InvG $G = (V, E)$ in such a way that the total modification cost is minimum a pre-specified node $s \in V$ becomes the 1-center of the InvG G. Here, we also represent an optimal algorithm to obtain the solution of an Inv1C location problem on the weighted IT T_{IG} of the weighted InvG, where n is the cardinality of V of G under certain weight restrictions. Besides these, we also design here an $O(n)$-time algorithm to compute the *average distance* of a graph G (with finite number of nodes and edges). This is the average of the distances over all unordered

pairs of nodes. A MADST of G is a spanning tree of G having minimum average distance. Some authors refer that tree as a MRCST. Further, we design an $O(n^2)$-time algorithm to determine a MADST on the *fuzzy InvG*, where n is the cardinality of the node set of given graph.

Chapter 4

In this chapter, we describe the method of *determination of MADT on CirGs and solution of Inv1C location problem on the weighted CirGs*. For this purpose we have construct a MADST of G having least average distance. Such a tree is sometimes referred to as a MRCST. Here, we design an $O(n^2)$-time algorithm to find a MADST on *CirG*, where n is the number of nodes of the graph. Also in this chapter, we present an optimal algorithm to find an Inv1C location on the weighted tree T_{CIR} corresponding to the weighted circular-arc graph $G = (V, E)$, where the node weights can be modified within certain bounds. The time complexity of our proposed algorithm is $O(n)$, where n is the cardinality of the node set of the CirG G.

Chapter 5

In 5th chapter, we describe the methods of *finding of a MADT on PerGs and Inv1C location problem on weighted PerGs*. Here, we design an $O(n^2)$-time algorithm to form a MADST on permutation graphs, where n is the cardinality of the graph. Also, in this chapter we represent an $O(n^2)$-time algorithm to an Inv1C on the tree T_{PER} of the weighted PerG $G = (V, E)$, where the node weights can be modified under some certain restrictions. The T-complexity of our proposed algorithm is $O(n)$, where n is the cardinality of the node set of the weighted PerG G.

Chapter 6

This chapter contains *an optimal algorithm for determination of Inv1C location problem on the weighted TraGs*. Here, we design an $O(n)$-time algorithm to solve an Inv1C location on the weighted tree T_{TRP} of the weighted TraG $G = (V, E)$, where we modify the node weights can be modified under certain restrictions.

Chapter 7

In this chapter, some concluding remarks have been made. Also future scope of further research are discussed here.

1.9 Summary

This is the introductory chapter of the thesis. Discussion about graph algorithm, computational complexities of algorithms are made here. Some graph theoretic terms are defined.

Also some special class of IntGs such as InvGs, PerGs, TraGs, CirGs and other graphs are focused. Motivation of our work and the organization of the thesis is discussed in this chapter.