CHAPTER 1

INTRODUCTION

1.1 Basic Concepts and Terminologies

Graph-theoretical ideas date back to at least the 1730's, when Leonhard Euler published his paper on the problem of Seven Bridges of Konigsberg [12]. This puzzle asks whether there is a continuous walk that crosses each of the seven bridges of Konigsberg only once and if so, whether a closed walk can be found. Furthermore, the large part of graph theory has been motivated by the study of games and recreational mathematics. Graphs are very convenient tools for representing the relationships among objects, which are represented by vertices. In their turn, relationships among vertices are represented by connections. In general, any mathematical object involving points and connections among them can be called a graph or a hyper graph. For a great diversity of problems such pictorial representations may lead to a solution. Examples of such applications include databases, physical networks, organic molecules, map colorings, signal-flow graphs, web graphs, tracing mazes as well as less tangible interactions occurring in social networks, ecosystems and in a flow of a computer program. The graph models can be further classified into different categories. For instance, two atoms in an organic molecule may have multiple connections between them, an electronic circuit may use a model in which each edge represents a direction, or a computer program may consist of loop structures. Therefore, for these examples we need multi graphs, directed graphs or graphs that allow loops. Thus, graphs can serve as mathematical models to solve an appropriate graph-theoretic problem, and then interpret the solution in terms of the original problem.

At present, graph theory is a dynamic field in both theory and applications. Graphs can be used as a modeling tool for many problems of practical importance. For instance, a network of cities, which are represented by vertices, and connections among them make a weighted graph. The well-known travelling salesman problem asks for the shortest
possible tour, which visits all the cities exactly once. And there are numerous applications like this.

Scientific conjectures are another important research area of graph theory. Scientific conjectures, obviously, express the most interesting theoretic statements, which are neither proved nor disproved. They are basically posed in order to draw attention of scientific community and advance progress in the corresponding field. Some conjectures have had a fundamental impact. For example, the development of graph theory over the last four decades has been strongly influenced by the Strong Perfect Graph Conjecture and perfect graphs introduced by Berge in the early 1960s [16]. This famous conjecture has been open for about 40 years, and various attempts to prove it have given rise to many powerful methods, important concepts and interesting results in graph theory. The conjecture was recently proved, and now it is referred to as the Strong Perfect Graph Theorem.

1.2 Domination

Domination in graph theory is a natural model for many location problems in operations research. As an example, consider the following fire station problem. Suppose a county has decided to build some fire stations, which must serve all of the towns in the county. The fire stations are to be located in some towns so that every town either has a fire station or is a neighbor of a town which has a fire station. To save money, the county wants to build the minimum number of fire stations satisfying the above requirements.

Domination has many other applications in the real world. The recent book by Haynes, Hedetniemini and Slater [12] illustrates many interesting examples, including dominating queens, sets of representatives, school bus routing, computer communication networks, (r, d)-configurations, radio stations, social network theory, landing surveying, kernels of games . . . etc. Among them, the classical problems of covering chessboards by the minimum number of chess pieces were important in stimulating the study of domination, which commenced in the early 1970’s. These problems certainly date back to De Jaenisch [69] and have been mentioned in the literature frequently since that time.
Introduction

A simple example is to determine the minimum number of kings dominating the entire chessboard. The answer for an \( m \times n \) chessboard is \( \left\lfloor \frac{m}{3} \right\rfloor \left\lfloor \frac{n}{3} \right\rfloor \). In the Chinese chess game, a king only dominates the four neighbor cells which have common sides with the cell the king lies. In this case, the Chinese king domination problem for an \( m \times n \) chessboard is harder. Figure 1.1 shows optimal solutions to both cases for a \( 3 \times 5 \) board.

![Chessboard Diagram](image)

Figure 1.1: King domination on a \( 3 \times 5 \) chessboard

We can abstract the above problems into the concept of domination in terms of graphs as follows. A dominating set of a graph \( G=(V, E) \) is a subset \( D \) of \( V \) such that every vertex not in \( D \) is adjacent to at least one vertex in \( D \). The domination number \( \gamma(G) \) of a graph \( G \) is the minimum size of a dominating set of \( G \).

For the fire station problem, we consider the graph \( G \) having all towns of the county as its vertices and a town is adjacent to its neighbor towns. The fire station problem is just the domination problem, as \( \gamma(G) \) is the minimum number of fire stations needed.

For the king domination problem on an \( m \times n \) chessboard, we consider the king’s graph \( G \), whose vertices correspond to the \( mn \) squares in the chessboard and two vertices are adjacent if and only if their corresponding squares have a common point or side. For the Chinese king domination problem, the vertex set is the same but two vertices are adjacent if and only if their corresponding squares have a common side. Figure 1.2 shows the corresponding graphs for the king and the Chinese king domination problems on a \( 3 \times 5 \) chessboard. \( \gamma(G) \) is then the minimum number of kings needed. Black vertices in the graph form a minimum dominating set.
Although many theoretic theorems for the domination problem have been established for a long time, the first algorithmic result on this topic was given by Cockayne, Goodman and Hedetemiemi [47] in 1975. They gave a linear-time 2 algorithm for the domination problem in trees by using a labeling method. On the other hand, at about the same time Garey and Johnson (see [86]) constructed the first (unpublished) proof that the domination problem is NP-complete for general graphs. Since then, many algorithmic results are studied for variants of the domination problem in different classes of graphs.

1.3 Definitions and Notations

1.3.1 Definition of Graph

A graph $G$ consists of a pair $(V(G), E(G))$ where $V(G)$ is a nonempty finite set whose elements are called points or vertices and $E(G)$ is a set of unordered pairs of distinct elements of $V(G)$. The elements of $E(G)$ are called lines or edges of the graph $G$. A graph $G$ with $p$ vertices and $q$ edges is called a $(p, q)$ graph.

Example:
Here $V(G) = \{a, b, c, d\}, E(G) = \{\{a, b\}, \{a, c\}, \{a, d\}\}$. This graph $G = (V, E)$ is a $(4,3)$ graph. The number $p$ is referred to as the order of $G$ and the number $q$ is referred to as the size of $G$. If $G$ is a $(p, q)$ graph then $p \geq 1$ and $0 \leq q \leq p(p - 1)/2$.

**Incidency:**
If $e = \{u, v\} \in E(G)$, the edge $e$ is said to join the vertices $u$ and $v$. We say that the vertices $u$ and $v$ are incident with the edge $e$.

**Adjacency:**
If $e = uv$ we say that the vertices $u$ and $v$ are adjacent. If two distinct edges $e$ and $f$ have a common vertex $v$, we say that the edges $e$ and $f$ are adjacent.

### 1.3.2 Directed and Undirected Graph

A graph $G = (V, E)$ is directed if the edge set is composed of ordered vertex (node) pairs. A graph is undirected if the edge set is composed of unordered vertex pair.

![Directed and Undirected Graph](image)

**1.3.3 Vertex Degrees**

**Degree:**
The degree $d_G(v)$ of a vertex $v$ in $G$ is the number of edges of $G$ incident with $v$, each loop counting as two edges. A vertex of degree 0 is called an isolated vertex. A vertex of degree 1 is called an end vertex or pendant vertex.

**Theorem 1.1:**
The sum of the degrees of the vertices of a graph $G$ is twice the number of edges. (i.e.) $\sum v \in V d_G(v) = 2q$. 

\[\text{(Image insertion)}\]
Corollary 1.1:

In any graph, the number of vertices of odd degree is even.

Minimum degree, Maximum degree:

For any graph $G$, we define

$$\delta(G) = \min\{\deg v : v \in V(G)\}, \Delta(G) = \max\{\deg v : v \in V(G)\}.$$ 

Regular graph:

If all the vertices of $G$ have the same degree $r$ then $\delta(G) = \Delta(G) = r$ and in this case $G$ is called a regular graph of degree $r$. A graph $G$ is $k$-regular if $d_G(v) = k$ for all $v \in V$. A regular graph is

![Graph with degree r = 0](image)

![Graph with degree r = 2](image)

one that is $k$-regular for some $k$. Complete graphs, complete bipartite graphs, and $k$-cubes are regular. A regular graph of degree 3 is called a cubic graph. The complete graph $K_p$ is regular of degree $p - 1$.

1.3.4 Loop and Multiple Edges

A loop is an edge whose endpoints are equal i.e., an edge joining a vertex to itself is called a loop. We say that the graph has multiple edges if in the graph two or more edges joining the same pair of vertices.

![Multiple Edge](image)
1.3.5 Graph Isomorphism

Two graphs \( G \) and \( H \) are said to be isomorphic (written \( G \cong H \)) if there are bijections \( \theta : V(G) \rightarrow V(H) \) and \( \varphi : E(G) \rightarrow E(H) \) such that \( \varphi G(e) = uv \) if and only if \( \varphi H(\varphi(e)) = \theta(u)\theta(v) \). Such a pair \( (\theta, \varphi) \) of mappings is called an isomorphism between \( G \) and \( H \).

For example, two unlabeled graphs, such as

![Graph Isomorphism Example](image)

1.3.6 Neighbor Vertex and Neighborhood

We write \( v_i v_j \in E(G) \) to mean \( \{v_i, v_j\} \in E(G) \), and if \( e = v_i v_j \in E(G) \), we say \( v_i \) and \( v_j \) are adjacent. Formally, given a graph \( G = (V, E) \), two vertices \( v_i, v_j \in V \) are said to be neighbors, or adjacent nodes, if \( \{v_i, v_j\} \in E(G) \). If \( G \) is directed, we distinguish between incoming neighbors of \( v_i \) (those vertices \( v_j \in V \) such that \( (v_j, v_i) \in E \)) and outgoing neighbors of \( v_i \) (those vertices \( v_i \in V \) such that \( (v_i, v_j) \in E \)).

The open neighborhood \( N(v) \) of the vertex \( v \) consists of the set vertices adjacent to \( v \), that is, \( N(v) = \{w : vw \in E\} \). The closed neighborhood of \( v \) is \( N[v] = N(v) \cup \{v\} \). For a set \( S \subseteq V \), the open neighborhood \( N(S) \) is defined to be \( \cup_{v \in S} N(v) \), and the closed neighborhood of \( S \) is \( N[S] = N(S) \cup S \).

1.3.7 Connectivity

**Connectedness:**

Two vertices \( u \) and \( v \) of a graph \( G \) are said to be connected if there is a \((u, v)\) path in \( G \).
Disconnectedness:
Two vertices $u$ and $v$ of a graph $G$ are said to be disconnected if there does not exist a $(u,v)$ path in $G$.

Component:
The subgraphs $G[V_1], G[V_2], \ldots, G[V_n]$ are called the components of $G$. If $G$ has exactly one component, $G$ is connected; otherwise $G$ is disconnected.

Vertex Cut:
A vertex cut of $G$ is a subset $V'$ of $V$ such that $G - V'$ is disconnected. A $k$-vertex cut is vertex cut of $k$ elements. The only graphs which do not have vertex cuts are those that contain complete graphs as spanning subgraphs.

Edge cut:
An edge cut of $G$ is a subset of $E$ of the form $[S, S']$ where $S$ is a nonempty proper subset of $V$. A $k$-edge cut is an edge cut of $k$ elements. If $G$ is nontrivial and $E'$ is an edge cut of $G$, then $G - E'$ is disconnected.

Cut vertex:
A vertex $v$ of $G$ is a cut vertex if $E$ can be partitioned into two nonempty subsets $E_1$ and $E_2$ such that $G[E_1]$ and $G[E_2]$ have just the vertex $v$ in common. If $G$ is loopless and nontrivial, then $v$ is a cut vertex of $G$ if and only if $\omega(G - v) > \omega(G)$.

Cut edge:
A cut edge of $G$ is an edge $e$ such that $\omega(G - e) > \omega(G)$. It is also called a bridge.

$k$-connected:
A graph $G$ is said to be $k$-connected if the removal of any set of less than $k$ vertices does not disconnect the graph.
1.3.8 Interval Graphs

Consider the intervals $(0,3), (2,7), (-1,1), (2,3), (1,4), (6,8)$ which may be illustrated as

-2 -1 0 1 2 3 4 5 6 7 8 9 10

We can construct the resulting interval graphs by taking the interval as vertices, join two of these vertices by an edge whenever the corresponding intervals have at least one point in common.

Note that since the intervals $(-1,1)$ and $(1,4)$ are open intervals, they do not have a point in common.

1.4 Graph Domination.
1.4.1 History of Domination Theory

The mathematical study of dominating sets in graphs dates back to 1850’s with the study of the problem of determining the minimum number of queens which are necessary to cover an $n \times n$ chessboard. Domination parameters have been studied by
Introduction

different authors. Acharya B.D., Sampathkumar E., and Waliker H.B [1]. are some Indian Mathematicians who have made substantial contribution to the study of domination in graphs. In 1979 they published a technical report “Recent developments in the theory of domination in graphs

1.4.2 Dominating Sets

**Dominating set:** A set $D$ of vertices in a graph $G$, is a dominating set, if every vertex not in $D$ (i.e.) (every vertex in $V - D$) is adjacent to at least one vertex in $D$.

$$D_1 = \{1\}, D_2 = \{1,2\}, D_3 = \{3\}, D_4 = \{4,1\}, D_5 = \{1,2,3,4\}.$$ $D_1$ and $D_2$ are not dominating sets; $D_3, D_4, D_5$ are dominating sets.

**Minimal dominating set:**
A dominating set $D$ is called a minimal dominating set, if for every vertex $v$, $D - \{v\}$ is not a dominating set.

**Minimum dominating set:**
A dominating set $D$ is called a minimum dominating set, if $D$ consists of minimum number of vertices among all dominating sets.

1.4.3 Domination Number

**Domination number $\gamma(G)$:**
The number of vertices in a minimum dominating set is defined as the domination number of a graph $G$, and it is denoted by $\gamma(G)$. 

10
$D_1 = \{1\}, D_2 = \{2,3,4,5\}$. 

$D_1$ and $D_2$ are minimal dominating sets. 

$D_1 = \{1\}$ is the minimum dominating set since it has minimum number of vertices than $D_2$. 

$\gamma(G) = 1$ is the domination number of $G$. 

**Observation:**

For a complete graph $K_p$ with $p$ vertices, $\gamma(K_p) = 1$. 

For a path $P_p$ with $p$ vertices, $\gamma(P_p) = \lfloor p/3 \rfloor$ 

For a cycle $C_p$ with $p$ vertices, $\gamma(C_p) = \lfloor p/3 \rfloor$ 

For a wheel $W_p$ with $p$ vertices, $\gamma(W_p) = 1$. 

For a complete bipartite graph $K_{m,n}$, $\gamma(K_{m,n}) = \min(m,n)$. 


A dominating set $D$ is minimal dominating set if and only if for each vertex $v$ in $D$, one of the following conditions holds: $v$ is an isolated vertex of $D$; There exists a vertex $u$ in $V - D$ such that intersection of $N(u)$ and $D = \{v\}$. 


Let $G$ be a graph without isolated vertices. If $D$ is a minimal dominating set, then $V - D$ is a dominating set.
1.4.4 Connected Domination Number

**Connected dominating set:**
A dominating set $D$ of a graph $G$ is a connected dominating set, if the induced subgraph $< D >$ is connected.

**Minimal connected dominating set:**
A connected dominating set $D$ is called a minimal connected dominating set, if for every vertex $v, D - \{v\}$ is not a connected dominating set.

**Minimum connected dominating set:**
A connected dominating set $D$ is called a minimum connected dominating set, if $D$ consists of minimum number of vertices among all connected dominating sets.

**Connected domination number $\gamma_c(G)$:**
The number of vertices in a minimum connected dominating set is defined as the connected domination number of a graph $G$, and it is denoted by $\gamma_c(G)$

$$D_1 = \{4,5,6\}$$ is a connected dominating set.
$$D_2 = \{4,5\}$$ is a minimal connected dominating set.
$$D_1 = \{4,5,6\}$$ is a minimum connected dominating set.
$$\gamma_c(G) = 3$$ is the connected domination number of $G$.

**Observation:**
For a complete graph $K_p$ with $p$ vertices, $\gamma_c(K_p) = 1$.
For a path $P_p$ with $p$ vertices, $\gamma_c(P_p) = p - 2$ if $p \geq 3$. Otherwise.
For a cycle $C_p$ with $p$ vertices, $\gamma_c(C_p) = p - 2$.
For a wheel $W_p$ with $p$ vertices, $\gamma_c(W_p) = 1$.
For a complete bipartite graph $K_{m,n}$, $\gamma_c(K_{m,n}) = min(m,n)$. 
If $H$ is a connected spanning subgraph of $G$, then $\gamma c(G) \leq \gamma c(H)$.

For any connected graph $G$ with $p \geq 3$ vertices, $\gamma c(G) \leq p - 2$.

1.4.5 Independent Domination Number

**Independent dominating set:**
A dominating set $D$ of a graph $G$ is an independent dominating set, if the induced subgraph $< D >$ has no edges.

**Minimal independent dominating set:**
An independent dominating set $D$ is called a minimal independent dominating set, if for every vertex $v, D - \{v\}$ is not an independent dominating set.

**Minimum independent dominating set:**
An independent dominating set $D$ is called a minimum independent dominating set, if $D$ consists of minimum number of vertices among all independent dominating sets.

**Independent domination number $\gamma_i(G)$:**
The number of vertices in a minimum independent dominating set is defined as the independent domination number of a graph $G$, and it is denoted by $\gamma_i(G)$.

$D_1 = \{1,2,3,5\}, D_2 = \{4,6,7\}$ are independent dominating sets.

$D_3 = \{1,2,3,6,7\}$ is a minimal independent dominating set.

$D_2 = \{4,6,7\}$ is the minimum independent dominating set since it has minimum number of vertices than $D_1$.

$\gamma_i(G) = 3$ is the independent domination number of $G$. 

\[ \gamma_i(G) = 3 \]
1.4.6 Total Domination Number

**Total dominating set:**

A dominating set $D$ of a graph $G$ is a total dominating set, if the induced subgraph $<D>$ has no isolated vertices.

**Minimal total dominating set:**

A total dominating set $D$ is called a minimal total dominating set, if for every vertex $v$, $D - \{v\}$ is not a total dominating set.

**Minimum total dominating set:**

A total dominating set $D$ is called a minimum total dominating set, if $D$ consists of minimum number of vertices among all total dominating sets.

**Total domination number $\gamma_t(G)$:**

The number of vertices in a minimum total dominating set is defined as the total domination number of a graph $G$, and it is denoted by $\gamma_t(G)$. This concept was introduced by Cockayne, Dawes and Hedetniemi [2] in “Total domination in graphs, Networks, 10 (1980) 211-219”.

\[ D1 = \{1,2,3,4,5\} \text{ is a total dominating set.} \]
\[ D2 = \{1,2,4,5\} \text{ is a minimal total dominating set.} \]
\[ D3 = \{2,3,4\} \text{ is a minimum total dominating set.} \]
\[ \gamma_t(G) = 3 \text{ is the total domination number of } G. \]
1.4.7 Inverse Domination Number

**Inverse dominating set:**

Let $D$ be a minimum dominating set in a graph $G$. If $V - D$ contains a dominating set $D'$ of $G$, then $D'$ is called an inverse dominating set with respect to $D$.

**Minimum inverse dominating set:**

An inverse dominating set $D$ is called a minimum inverse dominating set, if $D$ consists of minimum number of vertices among all inverse dominating sets.

**Inverse domination number $\gamma^{-1}(G)$:**

The number of vertices in a minimum inverse dominating set is defined as the inverse domination number of a graph $G$, and it is denoted by $\gamma^{-1}(G)$. This concept was introduced by Kulli V.R. and Sigarkanti S.C[3] in “Inverse domination in graphs, Nat. Acad. Sci. Lett., 14(1991) 473-475”.

![Graph Diagram]

$D_1 = \{2,5\}, D_2 = \{2,4\}, D_3 = \{3,5\}$ are the minimum dominating sets. Their corresponding inverse dominating sets are $D_1^* = \{1,3,4\}, D_2^* = \{3,5\}$, $D_3^* = \{2,4\}$ respectively. Thus $\gamma(G) = 2$ is the domination number of $G$, $\gamma^{-1}(G) = 2$ is the inverse domination number of $G$.

1.5 Applications of graph domination

Domination is a rapidly developing area of research in graph theory. The concept of domination has existed for a long time and early discussions on the topic can be found in works of Ore and Berge. The summary of the literature shows the following well-known problems, which are considered among the earliest applications for dominating sets.
Introduction

Queens Problem

This problem was mentioned by Ore. According to the rules of chess a queen can, in one move, advance any number of squares horizontally, diagonally, or vertically (assuming that no other chess figure is on its way). How to place a minimum number of queens on a chessboard so that each square is controlled by at least one queen? See one of the solutions in Figure 1.3.

![Queens dominating the chessboard](image)

Figure 1.3: Queens dominating the chessboard

Locating Radar Stations Problem

The problem was discussed by Berge in [16]. A number of strategic locations are to be kept under surveillance. The goal is to locate a radar for the surveillance at as few of these locations as possible. How a set of locations in which the radar stations are to be placed can be determined?

Problem of Communications in a Network

Suppose that there is a network of cities with communication links. How to set up transmitting stations at some of the cities so that every city can receive a message from at least one of the transmitting stations? This problem was discussed in detail by J.Wu and Liu in [119].

Nuclear Power Plants Problem

A similar known problem is a nuclear power plants problem. There are various locations and an arc can be drawn from location x to location y if it is possible for a watchman stationed at x to observe a warning light located at y. How many guards are needed to observe all of the warning lights, and where should they be located? At present,
domination is considered to be one of the fundamental concepts in graph theory and its various applications to ad hoc networks, biological networks, distributed computing, social networks and web graphs partly explain the increased interest. Such applications usually aim to select a subset of nodes that will provide some definite service such that every node in the network is 'close' to some node in the subset. The following examples show when the concept of domination can be applied in modelling real-life problems.

Modelling Biological Networks

Using graph theory as a modelling tool in biological networks allows the utilization of the most graphical invariants in such a way that it is possible to identify secondary RNA (Ribonucleic acid) motifs numerically. Those graphical invariants are variations of the domination number of a graph. The results of the research carried out in [49] show that the variations of the domination number can be used for correctly distinguishing among the trees that represent native structures and those that are not likely candidates to represent RNA.

Modeling Social Networks

Dominating sets can be used in modelling social networks and studying the dynamics of relations among numerous individuals in different domains. A social network is a social structure made of individuals (or groups of individuals), which are connected by one or more specific types of interdependency. The choice of initial sets of target individuals is an important problem in the theory of social networks. In the work of Kelleher and Cozzens, social networks are modeled in terms of graph theory and it was shown that some of these sets can be found by using the properties of dominating sets in graphs.

Facility Location Problems

The dominating sets in graphs are natural models for facility location problems in operational research. Facility location problems are concerned with the location of one or more facilities in a way that optimizes a certain objective such as minimizing transportation cost, providing equitable service to customers and capturing the largest market share.
Introduction

Coding Theory

The concept of domination is also applied in coding theory as discussed by Kalbeisch, Stanton and Horton and Cockayne and Hedetniemi [24]. If one defines a graph, the vertices of which are the $n$-dimensional vectors with coordinates chosen from \{1, ..., $p$\}, $p > 1$, and two vertices are adjacent if they differ in one coordinate, then the sets of vectors which are $(n, p)$ covering sets, single error correcting codes, or perfect covering sets are all dominating sets of the graph with determined additional properties.

Multiple Domination Problems

An important role is played by multiple domination. Multiple domination can be used to construct hierarchical overlay networks in peer-to-peer applications for more efficient index searching. The hierarchical overlay networks usually serve as distributed databases for index searching, e.g. in modern file sharing and instant messaging computer network applications. Dominating sets of several kinds are used for balancing efficiency and fault tolerance [27] as well as in the distributed construction of minimum spanning trees. Another good example of direct, important and quickly developing applications of multiple domination in modern computer networks is a wireless sensor network.

A wireless sensor network (WSN) usually consists of up to several hundred small autonomous devices to measure some physical parameters. Each device contains a processing unit and a limited memory as well as a radio transmitter and a receiver to be able to communicate with its neighbours. Also, it contains a limited power battery and is constrained in energy consumption. There is a base station, which is a special sensor node used as a sink to collect information gathered by other sensor nodes and to provide a connection between the WSN and a usual network. A routing algorithm allows the sensor nodes to self-organize into a WSN. As stated in [66], an important goal in WSN design is to maximize the functional lifetime of a sensor network by using energy efficient distributed algorithms, networking and routing techniques. To maximize the functional lifetime, it is important to select some sensor nodes to behave as a backbone set to support routing communications. The backbone set can be considered as a dominating set in the corresponding graph. Dominating sets of several different kinds have proved to be useful and effective for modelling backbone sets. In the recent literature particular
Introduction

Attention has been paid to construction of k-connected k-dominating sets in WSNs, and several probabilistic and deterministic approaches have been proposed and analyzed. The backbone set of sensor nodes should be selected as small as possible and, on the other hand, it should guarantee high efficiency and reliability of networking and communications. This trade-off requires construction of multiple dominating sets providing energy efficient and reliable data dissemination and communication protocols.

A homogeneous WSN consists of wireless sensor devices of the same kind. All the devices have the same set of limited resources and, originally, no hierarchy is imposed on the network structure and communications. In a network of this kind, the only special sensor node is a base station. For all the other nodes, it is necessary to construct and switch the backbone sets and communications efficiently so that all the network nodes stay in operation as long as possible. Therefore, in this case, it is important to be able to construct and switch dominating sets and route communications uniformly and efficiently with respect to the energy consumption of each particular sensor node. This has to be done to optimize the functional lifetime of the whole network. Usually, a WSN is mathematically modelled as a unit or quasi-unit disk graph.

These are the most natural and general graph models for a WSN. In a unit disk graph model, nodes correspond to sensor locations in the Euclidean plane and are assumed to have identical (unit) transmission ranges. An edge between two nodes means that they can communicate directly, i.e. the distance between them is at most one. A survey of known results on unit disk graphs, including algorithms for constructing dominating sets, can be found in. A quasi-unit disk graph model takes into consideration possible transmission obstacles and is much closer to reality: we are sure to have an edge between two nodes if the distance between them is at most a parameter $d$, $0 < d < 1$. If the distance between two nodes is in the range from $d$ to $1$, the existence of an edge is not specified. A description of several more restricted geometric graph models for WSN design, e.g. the related neighborhood graph, Gabriel graph, Yao graph etc., can be found in [59]. Domination is an area in graph theory with an extensive research activity. A book by Haynes, Hedetniemi and Slater on domination published in 1998 lists 1222 articles in this area.
1.6 Algorithms

1.6.1 Definitions and Preliminaries

The word algorithm comes from the name of a Persian author, Abu Ja’far Mohammed Ibn Musa al Khowarizmi (825 A.D), who wrote a text book on Mathematics. This word has taken on a special significance in computer science, where 'algorithm' has come to refer to a method that can be used by a computer for the solution of a problem. This is what makes algorithm different from words such as process, technique, method etc...

An algorithm is a finite set of instructions that, if followed accomplishes a particular task. An algorithm is composed of a finite set of steps, each of each of which may require one or more operations. The possibility of a computer carrying out these operations necessitates that certain constraints be placed on the type of operations an algorithm can include. Algorithms that are definite and effective are also called computational procedures. A program is the expression of an algorithm in a programming language. The study of algorithms includes many important and active areas of research. They are how to devise algorithms, how to analyze algorithms and how to test a program.

There are many ways in which an algorithm can be specified. These includes

1. A carefully worded statement written in a normal language such as English.
2. A list of steps defined in Mathematics and a normal language
3. A flow chart.
4. A program written in a semi-formal fashion such as pidgin-Pascal.
5. A formal program capable of being executed by a computer.

1.6.2 Complexity and Approximability.

In this section, we give some basic notions of problems, their complexity, and algorithms to solve them. A more complete overview can be found in [23]. In order to do optimization, and to analyze algorithms and complexity, we need to define the essence of optimization: a problem.
**Definition 1.1** A problem is given by a problem description $\Pi$, which is a pair $(I, S)$ such that

- $I$ denotes the set of instances of $\Pi$, and
- for each $x \in I$, $S(x)$ is the set of feasible solutions.

We consider two types of problems called decision and optimization problems, which are given by refining the above general problem description.

**Definition 1.2.** Let $\Pi$ be a problem description whose set $I$ of instances is partitioned into two sets $Y$ and $N = I \setminus Y$, called positive and negative instances as follows:

$$x \in Y \iff S(n) \neq \emptyset$$

Seeking an answer to the question of whether an instance $x \in I$ belongs to $Y$ is called a **decision problem**.

The aim of a decision problem is to recognize the positive instances, that is, those instances that have feasible solutions. For optimization problems, we look at problem descriptions, where additionally each feasible solution is evaluated by an objective function as follows. For an instance $x \in I$ together with a feasible solution $y \in S(x)$, there is an objective function value (or measure) $f(x, y) \in \mathbb{N}$ of that solution. For each instance, the objective function may take on different values for different feasible solutions. We are interested in solutions which minimize or maximize this value with respect to all feasible solutions for such an instance.

**Definition 1.3.** In a problem description $\Pi$ for each pair of instance $x \in I$ and feasible solution $y \in S(x)$, let the objective function $f(x, y)$ assign a positive integer to this pair. The problem $\Pi_{\text{min}}$ is:

Given an instance $x \in I$, find a solution $y^+ \in S(x)$ such that for every $y \in S(x)$, $f(x, y^+) \leq f(x, y)$ holds. is called a **minimization problem**, and $y^+$ is called an optimal solution for this problem. Analogously, a **maximization problem** is defined by finding a solution with maximum objective value over all feasible solutions. The question of minimization or maximization for an optimization problem is called goal of the problem. Note that in the above definition, the range of an objective function $f$ can easily be relaxed to rational numbers. It is easy to see that any optimization problem $\Pi$ can be transformed in a
straightforward way into corresponding decision problem. This is done by looking at
the following, modified set of feasible solutions in case of minimization.

Given an instance \( x \in I \) and a rational number \( \tau \), let

\[
S\tau(x) = \{ y \in S(x) \mid f(x,y) \leq \tau \}
\]

Denote the set of feasible solutions for the corresponding decision problem.

Clearly, for an instance \( x \), the decision problem based on \( S\tau(x) \) is equivalent to the
following problem statement

Given an instance \( x \in I \) and a rational number \( \tau \), is there a solution \( y \in S(x) \) so that
\( f(x,y) \leq \tau \)?

The maximization case is obtained in a symmetric way.

For all the above problems, we can describe strategies and approaches which
generate answers to the problems by algorithms. Generally speaking, an algorithm \( A \) for
a problem \( \Pi \) is a finite sequence of instructions or calculations to be performed whenever
an instance \( x \in I \) is presented to \( A \), and that returns some solution from \( S(x) \) when all
these instructions have been performed. The solution returned is referred to as output of
\( A \) for the instance, or \( A(x) \). For optimization problems, an output is usually considered
together with its objective value \( f(x,A(x)) \).

The instance \( x \in I \) is also called input, especially when considered in relation with
an algorithm for the problem at hand, and the value given by the objective function
\( f(x,y) \) for a solution \( y \in S(x) \) is usually called cost.

For a problem \( \Pi \), we define the length \( |x| \) of an instance \( x \in I \) as the number of bits
used to specify \( x \) in some fixed encoding. The time-complexity, i.e. the number of
elementary steps it takes for an algorithm to return a solution, is measured as a function
of the length of an input instance. The computation time, or run time, of an algorithm
is measured by the number of basic operations it performs, e.g. additions, comparisons,
multiplications etc. We suppose that all algorithms run on the same machine model, the
random access machine (RAM), which is polynomially related to the Turing machine
and other reasonable computational models.

The run time is usually expressed in asymptotic notation. For two functions
\( f : \mathbb{N} \to \mathbb{R}_+ \) and \( g : \mathbb{N} \to \mathbb{R}_+ \), we say that "\( f \) is in the order of \( g \)", written \( f(n) = O(g(n)) \) for short, if there exist \( c, n_0 \in \mathbb{N} \) such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \).
While this $O$-notation gives an upper bound, we can analogously define $f(n) = \Omega(g(n))$ for lower bounds to say that "$f$ is at least in the order of $g". It is $f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$. If the run time of an algorithm $A$ is bounded by a polynomial $p(|x|)$ for all possible instances $x \in I$, thus, for any instance $x$ and some fixed $k \in \mathbb{N}$, $A$ is an algorithm with run time $O(|x|^{k})$, then we say that $A$ is efficient. A problem that can be solved by an efficient algorithm is referred to as polynomially solvable.

A problem that can be solved in polynomial time is generally considered easy. However, in the next chapters, we concentrate on problems which are hard. The notion of hard problems is formalized based on decision problems. The resulting complexity hierarchy of problems has been extended ever since the first appearance of the class $NP$ together with $NP - complete$ problems in the 1970s [11]

1.6.3 Decision Problems and the Class $NP$

The class $NP$ consists of all decision problems that can be certified in polynomial time: That is, a decision problem is in $NP$ if there is a (necessarily polynomial size) "certificate" which shows that an input instance is a positive instance, and that there is a polynomial time algorithm which can verify this certificate with respect to the problem.

The class $P$ consists of all those decision problems that can be recognized in polynomial time, i.e. for which the decision of an instance to be a positive instance can be reached in polynomial time.

Clearly, $P \subseteq NP$ holds. However, the question whether $P = NP$ or $P \neq NP$ is open. It is widely believed that $P \neq NP$, and a good reason for this is given by the class of $NP - complete$ Problems. The class of $NP - complete$ problem has the following compelling property: if one can prove that a single $NP - complete$ problem actually is in $P$, then $P = NP$. The reason for this comes from the fact that all $NP - complete$ problems are polynomially reducible to each other.

Let $\Pi_1, \Pi_2 \in NP$. The problem $\Pi_1$ is $polynomially\ reducible\ to\ \Pi_2$, if there exists a polynomial time algorithm that for every instance $x_1$ of $\Pi_1$ produces an instance $x_2$ of $\Pi_2$ so that $x_1$ is a positive instance of $\Pi_1$ if and only if $x_2$ is a positive instance of $\Pi_2$. Practically, this means that we can use an efficient algorithm for $\Pi_2$, to solve $\Pi_1$. 


efficiently: simply transform the instance of \( \Pi_1 \), into the respective instance of \( \Pi_2 \), and then solve this instance.

With the introduction of a polynomial reduction, we can formally introduce *NP-complete* problems [23].

**Definition 1.4.** A decision problem \( \Pi \) is *NP-complete* if

- \( \Pi \) is in NP, and
- all other NP problems are polynomially reducible to \( \Pi \).

Polynomial reduction is a transitive relation on the class NP. If a problem \( \Pi_1 \) is polynomially reducible to \( \Pi_2 \), and \( \Pi_2 \) is reducible to \( \Pi_3 \), then \( \Pi_1 \) is also reducible to \( \Pi_3 \). This fact can be exploited to show that a new problem \( \Pi \) is *NP-complete* since it now suffices to show that \( \Pi \) is in NP and that there is an *NP-complete* problem which is polynomially reducible to \( \Pi \). The start, i.e. the first problem shown to be *NP-complete*, was made by the satisfiability problem (SAT) [11], and today there are numerous problems that are known to be *NP-complete*.

1.6.4 NP-Optimization Problems

Similar to the class NP for decision problems, we introduce the class NPO for optimization problems [12]. For this class, we consider optimization problems for which the length of feasible solutions is polynomially bounded in terms of the length of an instance, and for which the value of the objective function can also be computed in polynomial time. More formally, an optimization problem \( \Pi \) is in the class NPO if

- all solutions are short, that is for every \( x \in I \), and every \( y \in S(x) \), we have \( |y| \leq p(|x|) \) for some polynomial \( p \),
- for any \( x \) and any \( y \) with \( |y| \leq p(|x|) \), the question whether \( y \in S(x) \) can be decided in polynomial time, and
- given \( x \in I \) and \( y \in S(x) \), the objective function \( f(x,y) \) is computable in polynomial time.

The class PO then corresponds to all optimization problems in NPO that can be solved to optimality in polynomial time. Again, we are interested in the hard problems, given by the following definition.
**Definition 1.5.** A problem $\Pi$ is said to be \(\text{NP-hard}\) if every problem in \(\text{NP}\) can be solved in polynomial time using a polynomial time algorithm that solves $\Pi$ as a subroutine.

Note that in the above definition, we no longer restrict $\Pi$ to be a decision problem. In particular, this definition includes all optimization problems for which the corresponding decision problem is \(\text{NP-complete}\). Furthermore, it is easy to see that if $P \neq \text{NP}$ holds, and then $\text{PO} \neq \text{NPO}$ has to hold, as well.

In order to show that a problem $\Pi$ is $\text{NP-hard}$, it suffices to show that an $\text{NP-complete}$ problem $\Pi'$ could be solved efficiently by efficiently solving $\Pi$. All other problems in $\text{NP}$ are polynomially reducible to $\Pi'$.

From an algorithmic point of view, knowing that an $\text{NPO}$ problem is $\text{NP-hard}$, we also know that we cannot compute an optimal solution in polynomial time, unless $P = \text{NP}$. In this case, there are two major options.

- On the one hand, we can abandon the idea of polynomial time, and still require an optimal solution from an algorithm. One way to achieve this would be complete enumeration, which may lead to exponential run time, in the worst case with an exponent only bounded by the polynomial bound on the size of all solutions.
- On the other hand, we can sacrifice optimality and start looking for approximate solutions which are computed by a polynomial time, i.e. efficient, algorithm.

The latter approach is.

Let $\Pi$ be an $\text{NPO}$ problem. For any instance $x \in I$, denote by $s^+(x)$ the cost of an optimal solution. The performance ratio of any feasible solution $y \in S(x)$ is then given by

$$R(x, y) := \max \left\{ \frac{f(x, y)}{s^+(x)}, \frac{s^+(x)}{f(x, y)} \right\}$$

Note that the performance ratio is always greater than or equal to 1, independent of the goal: the first fraction is used for minimization, the second for maximization problems. Clearly, the closer $R(x, y)$ is to 1, the closer a solution is to an optimal solution. The performance ratio is also called approximation ratio.
**Definition 1.6.** Let $\Pi$ be an NPO problem, and let $A$ be an algorithm that, for every instance $x \in \mathcal{I}$, returns a feasible solution $A(x) \in S(x)$.

Given a function $r : \mathbb{N} \rightarrow [1, \infty)$, the algorithm $A$ is an $r(n)$-approximate algorithm if for every instance $x \in \mathcal{I}$, the inequality

$$r(x, A(x)) \leq r(|x|)$$

holds. If there exists an $r(n)$-approximate polynomial time algorithm for $\Pi$, we say that $\Pi$ is approximable within $r(n)$ (in polynomial time).

An algorithm which actually gives a constant performance ratio independent of the length of the instance is referred to as constant-factor approximation. An NPO Problem $\Pi$ belongs to the class $APX$ if it is approximable within $\alpha$ (in polynomial time), for some constant $\alpha > 1$.

![Fig1.4:Relationships between some complexity classes for optimization problem.](image)

**Definition 1.7.** A family $\{A_\varepsilon\}_{\varepsilon > 0}$ of $(1 + \varepsilon)$-approximation algorithms is called an approximation scheme.

If, for each fixed $\varepsilon > 0$, the $(1 + \varepsilon)$-approximation algorithm $A_\varepsilon$ of an approximation scheme $\{A_\varepsilon\}_{\varepsilon > 0}$ runs in polynomial time on the input size, we call $\{A_\varepsilon\}_{\varepsilon > 0}$ a Polynomial-Time Approximation Scheme (PTAS).
An NPO problem that admits a PTAS is also in the class PTAS. We only demand the run time of a PTAS to be polynomial in the size of the input instance, and not in the desired performance ratio parameter $1/\varepsilon$. Computation times of $2^{1/\varepsilon} \cdot p(|x|)$ or $O(|x|^{1/\varepsilon})$ are allowed, obviously resulting in high run time when $\varepsilon$ is close to 0.

Figure 1.4 gives the relations between the complexity classes for NPO problems [54]. It is easy to see that the inclusions given are strict if and only if $P \neq NP$. Especially to stress the difference between the classes APX and PTAS, there is also the notion of APX-complete problems. APX-complete problems are those problems in NPO that are the most difficult ones to approximate. Loosely speaking, an NPO problem is APX-complete if there exists a bound $\alpha > 1$ such that there exists no polynomial time $\alpha$-approximation for this problem unless $P = NP$. Thus, if $P \neq NP$, for APX-complete problems, there cannot exist a polynomial-time approximation scheme. There is a long list of other complexity classes and notions, in fact, a whole “complexity zoo” [14]. Here, we only presented the major classes that are of importance for this thesis. For a rigorous introduction to the theory of NP, we refer the reader to the classic book by Garey and Johnson [23]. A list of NPO complete problems from various application areas is given by [12], including many further references.

1.6.5 Cheng's Greedy Algorithm.

In [24], Cheng et al. propose a greedy algorithm for MCDS in unit-disk graphs. Compared to the many heuristics, this algorithm relies on an MIS but the resultant CDS may not contain all the elements in the MIS.

Assume initially all nodes are colored white. The construction of a CDS contains four phases. In the first phase, an MIS is computed and all its members are colored red. In the second phase, a node that can decrease the maximum number of pieces is selected, where a piece is either a red node, or a connected black component. This node is colored black and all its non-black neighbors are colored gray. When the second phase is over, we still have some white nodes left. The third phase will compute a spanning tree for each connected component in the subgraph reduced by all white nodes. Connect each tree to the nearest black component with black nodes accordingly. All non-leaf tree nodes are
colored black while leaf nodes are colored gray. The last phase will seek chains of two gray nodes to connect disjoint black components.

The motivation of Cheng's algorithm are two fold. First, the greedy choice in Guha and Khuller's second algorithm [44] is the one that can decrease the maximum number of pieces, where a piece is either a connected black component, or a white node. Second, a unit-disk graph has at most 5 independent neighbors. Thus intuitively one can choose the greedy choice that can connect to as many independent nodes as possible. In other words, the node to be colored black at each step will try to cover more uncovered area, if we model vertices in a unit-disk graph as nodes in a flat area. Unfortunately Cheng's algorithm does not have a solid performance analysis.

1.6.6 Greedy Approach

Greedy Algorithm works by making the decision that seems most promising at any moment; it never reconsiders this decision, whatever situation may arise later. To construct the solution in an optimal way. Algorithm maintains two sets. One contains chosen items and the other contains rejected items.

1. The greedy algorithm consists of four functions.
2. A function that checks whether chosen set of items provide a solution.
3. A function that checks the feasibility of a set.
4. The selection function tells which of the candidates is the most promising.
5. An objective function, which does not appear explicitly, gives the value of a solution.

1.6.7 Divide-and-conquer

Divide-and-conquer is a top-down technique for designing algorithms that consists of dividing the problem into smaller sub problems hoping that the solutions of the sub problems are easier to find and then composing the partial solutions into the solution of the original problem. Little more formally, divide-and-conquer paradigm consists of following major phases:
Introduction

- Breaking the problem into several sub-problems that are similar to the original problem but smaller in size,
- Solve the sub-problem recursively (successively and independently), and then
- Combine these solutions to sub problems to create a solution to the original problem.

Binary Search (simplest application of divide-and-conquer)

Binary Search is an extremely well-known instance of divide-and-conquer paradigm. Given an ordered array of n elements, the basic idea of binary search is that for a given element we "probe" the middle element of the array. We continue in either the lower or upper segment of the array, depending on the outcome of the probe until we reached the required (given) element.

1.6.8 Dynamic programming

Dynamic programming is a fancy name for using divide-and-conquer technique with a table. As compared to divide-and-conquer, dynamic programming is more powerful and subtle design technique. Let me repeat, it is not a specific algorithm, but it is a meta-technique (like divide-and-conquer). This technique was developed back in the days when "programming" meant "tabular method" (like linear programming). It does not really refer to computer programming. Here in our advanced algorithm course, we'll also think of "programming" as a "tableau method" and certainly not writing code. Dynamic programming is a stage-wise search method suitable for optimization problems whose solutions may be viewed as the result of a sequence of decisions. The most attractive property of this strategy is that during the search for a solution it avoids full enumeration by pruning early partial decision solutions that cannot possibly lead to optimal solution. In many practical situations, this strategy hits the optimal solution in a polynomial number of decision steps. However, in the worst case, such a strategy may end up performing full enumeration.
The dynamic programming is among the most powerful for designing algorithms for optimization problem. This is true for two reasons. Firstly, dynamic programming solutions are based on few common elements. Secondly, dynamic programming problems are typical optimization problems i.e., find the minimum or maximum cost solution, subject to various constraints.

In other words, this technique used for optimization problems:

- Find a solution to the problem with the optimal value.
- Then perform minimization or maximization. (We'll see example of both in CLRS).

There are three basic elements that characterize a dynamic programming algorithm:

### 1.6.9 Approximate algorithm

An approximate algorithm is a way of dealing with NP-completeness for optimization problem. This technique does not guarantee the best solution. The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at most polynomial time.

Suppose we have some optimization problem instance $i$, which has a large number of feasible solutions. Also let $c(i)$ be the cost of solution produced by approximate algorithm and $c^*(i)$ be the cost of optimal solution. For minimization problem, we are interested in finding a solution of a given instance $i$ in the set of feasible solutions, such that $c(i)/c^*(i)$ be as small as possible. On the other hand, for maximization problem, we are interested in finding a solution in the feasible solution set such that $c^*(i)/c(i)$ be as small as possible.

### 1.7 Computational Complexity

Two important ways to characterize the effectiveness of an algorithm are its space complexity and time complexity. Time complexity of an algorithm concerns determining an expression of the number of steps needed as a function of the problem size. Since the step count measure is somewhat coarse, one does not aim at obtaining an exact step count. Instead, one attempts only to get asymptotic bounds on the step count. Asymptotic
analysis makes use of the $O$ (Big Oh) notation. Two other notational constructs used by computer scientists in the analysis of algorithms are $\Theta$ (Big Theta) notation and $\Omega$ (Big Omega) notation. The performance evaluation of an algorithm is obtained by totaling the number of occurrences of each operation when running the algorithm. The performance of an algorithm is evaluated as a function of the input size $n$ and is to be considered modulo a multiplicative constant. The following notations are commonly use notations in performance analysis and used to characterize the complexity of an algorithm.

### 1.7.1 $\Theta$-Notation

This notation bounds a function to within constant factors. We say $f(n) = \Theta(g(n))$ if there exist positive constants $n_0$, $c_1$ and $c_2$ such that to the right of $n_0$ the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive. In the set notation, we write as follows:

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_1, \text{and } n_0 \text{ such that } 0 \leq c_1 \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

We say that is $g(n)$ an asymptotically tight bound for $f(n)$.

Graphically, for all values of $n$ to the right of $n_0$, the value of $f(n)$ lies at or above $c_1 g(n)$ and at or below $c_2 g(n)$. In other words, for all $n \geq n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor. We say that $g(n)$ is an asymptotically tight bound for $f(n)$.
Introduction

In the set terminology, \( f(n) \) is said to be a member of the set \( O(g(n)) \) of functions. In other words, because \( O(g(n)) \) is a set, we could write

\[
f(n) \in O(g(n))
\]

to indicate that \( f(n) \) is a member of \( \Theta(g(n)) \). Instead, we write

\[
f(n) = \Theta(g(n))
\]
to express the same notation.

Historically, this notation is "\( f(n) = \Theta(g(n)) \)" although the idea that \( f(n) \) is equal to something called \( \Theta(g(n)) \) is misleading.

Example: \( \frac{n^2}{2} - 2n = (n^2) \), with \( c_1 = 1/4 \), \( c_2 = 1/2 \), and \( n_0 = 8 \).

1.7.2 \( \infty \)-Notation (Upper Bound)

This notation gives an upper bound for a function to within a constant factor. We write \( f(n) = O(g(n)) \) if there are positive constants \( n_0 \) and \( c \) such that to the right of \( n_0 \), the value of \( f(n) \) always lies on or below \( c g(n) \).

In the set notation, we write as follows: For a given function \( g(n) \), the set of functions

\[
O(g(n)) = \{ f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0 \}
\]

We say that the function \( g(n) \) is an asymptotic upper bound for the function \( f(n) \). We use \( \infty \)-notation to give an upper bound on a function, to within a constant factor.
Introduction

Graphically, for all values of n to the right of \( n_0 \), the value of the function \( f(n) \) is on or below \( g(n) \). We write \( f(n) = O(g(n)) \) to indicate that a function \( f(n) \) is a member of the set \( O(g(n)) \) i.e.

\[ f(n) \in O(g(n)) \]

Note that \( f(n) = \Theta(g(n)) \) implies \( f(n) = O(g(n)) \), since \( \Theta \)-notation is a stronger notation than \( O \)-notation.

Example: \( 2n^2 = O(n^3) \), with \( c = 1 \) and \( n_0 = 2 \).

Equivalently, we may also define \( f \) is of order \( g \) as follows:

If \( f(n) \) and \( g(n) \) are functions defined on the positive integers, then \( f(n) \) is \( O(g(n)) \) if and only if there is a \( c > 0 \) and an \( n_0 > 0 \) such that

\[ |f(n)| \leq |g(n)| \text{ for all } n > n_0 \]

Historical Note: The notation was introduced in 1892 by the German mathematician Paul Bachman.

1.7.3 \( \Omega \)-Notation (Lower Bound)

This notation gives a lower bound for a function to within a constant factor. We write \( f(n) = \Omega(g(n)) \) if there are positive constants \( n_0 \) and \( c \) such that to the right of \( n_0 \), the value of \( f(n) \) always lies on or above \( c g(n) \).

In the set notation, we write as follows: For a given function \( g(n) \), the set of functions

\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq c g(n) \leq f(n) \text{ for all } n \geq n_0 \} \]

We say that the function \( g(n) \) is an asymptotic lower bound for the function \( f(n) \).
Introduction

The intuition behind $\Omega$-notation is shown above.

Example: $\sqrt{n} = (lg n), with c = 1$ and $n0 = 16$.

1.7.4 Algorithm Analysis

The complexity of an algorithm is a function $g(n)$ that gives the upper bound of the number of operation (or running time) performed by an algorithm when the input size is $n$. There are two interpretations of upper bound. Worst-case Complexity. The running time for any given size input will be lower than the upper bound except possibly for some values of the input where the maximum is reached.

Average-case Complexity. The running time for any given size input will be the average number of operations over all problem instances for a given size. Because, it is quite difficult to estimate the statistical behavior of the input, most of the time we content ourselves to a worst case behavior. Most of the time, the complexity of $g(n)$ is approximated by its family $o(f(n))$ where $f(n)$ is one of the following functions. $n$ (linear complexity), $\log n$ (logarithmic complexity), $n^a$ where $a \geq 2$ (polynomial complexity), $a^n$ (exponential complexity).

1.7.5 Optimality

Once the complexity of an algorithm has been estimated, the question arises whether this algorithm is optimal. An algorithm for a given problem is optimal if its complexity reaches the lower bound over all the algorithms solving this problem. For example, any algorithm solving “the intersection of $n$ segments” problem will execute at least $n^2$ operations in the worst case even if it does nothing but print the output. This is abbreviated by saying that the problem has $\Omega(n^2)$ complexity. If one finds an $O(n^2)$ algorithm that solve this problem, it will be optimal and of complexity $\theta(n^2)$.

1.7.6 Reduction

Another technique for estimating the complexity of a problem is the transformation of problems, also called problem reduction. As an example, suppose we know a lower bound for a problem A, and that we would like to estimate a lower bound for a problem B. If we can transform A into B by a transformation step whose cost is less
than that for solving A, then B has the same bound as A. The Convex hull problem nicely illustrates "reduction" technique. A lower bound of Convex-hull problem established by reducing the sorting problem (complexity: $\Theta(n \log n)$) to the Convex hull problem.

1.8 Outline of the Thesis

Chapter 1 is introductory in nature. Domination is a rapidly developing area of research in graph theory, and its various applications to ad hoc networks, distributed computing, social networks and web graphs partly explain the increased interest. The study of domination in graphs originated around 1850 with the problem of placing minimum number of queens on an $n \times n$ chess board so as to dominate every square. The theory of domination in graphs introduced by Ore [23] and Berge [16] is an emerging area of research in graph theory today. Berge [16] presents the problem of five queens, namely, place five queens on the chessboard so that every square is covered by at least one queen. The solution $n^2$ cells of the $n \times n$ chessboard and vertices $u$ and $v$ are adjacent if a queen move from $u$ to $v$ in one move. This leads to domination in graphs. Another problem is that not only the squares are covered or dominated by one or more queens but each queen is covered by another queen, This leads to total domination in graphs.

In chapter 2 Literature reviews on the current trends and techniques used for Theory of domination in graphs for an Application of Mobile Ad-hoc networks and routing in Mobile network Mobile Ad-hoc networks have been widely researched for many years research on wireless ad hoc networks has been ongoing for decades. The idea for wireless ad hoc networks was triggered during the project called Deference Advanced Research Project Agency (DARPA) packet radio networks (PRNet). Which evolved into the survivable adaptive radio networks .

In Chapter3 we studied the problem of the design of Mobile Ad-hoc networks from the point of view of the Independent and Dominating sets executing the efficient algorithms that create independent and dominating sets, both from the structural, e.g. maximal sets, and the optimization, e.g. maximum cardinality sets, point of view. Also, due to the wireless structure and resource-poor nature of the underlying communication network, we are interested in algorithms that account for this fact. We now go on to
Introduction

present a model for the local communication characteristics of wireless networks which we use to describe locally executed algorithms.

In chapter 4 we have studied the Connected Dominating Set (CDS) and Total Dominating sets problem in Circular Graphs with only Bidirectional links (CGB). The circular graphs can be used to model wireless ad hoc networks where nodes have different transmission ranges. We have proposed three algorithms and shown that the obtained CDS is within a constant factor of the optimal CDS. The main approach in our algorithms is to construct a maximal independent set and then connect them. Through the theoretical analysis. We have shown that using a Steiner tree with the minimum number of Steiner nodes to interconnect the maximal independent set can help to reduce the size of the CDS. In addition, choosing a node with the largest transmission range as a dominator can further reduce the CDS size. Moreover, we have also presented the size relationship between an independent set and a CDS of a given network. We have pointed out some important characteristics of a CGB.

In chapter 5, we have studied matching domination applications in wireless networks and propose an algorithm for finding dominating sets in wireless networks. The main result proved in this chapter is the set produced by the algorithm is a matching dominating set and it is minimum.

Chapter 6 deals with applications of Global Dominating Sets in Web Graphs (GDCWG). We study the size of generalized dominating sets in two graph processes which are widely used to model aspects of the world-wide web. On the one hand, we show that graphs generated this way have fairly large dominating sets (i.e. linear in the size of the graph). On the other hand, we present efficient strategies to construct small dominating sets. The algorithmic results represent an application of a particular analysis technique which can be used to characterize the asymptotic behavior of a number of dynamic processes related to the web.

We investigated the property of Global Dominating Sets of an web graphs. It is proved that for a circular arc graph, any minimum dominating set is a global dominating set if the dominating number is greater than 3. We present a linear-time approximation scheme for this problem on graph with bounded growth. The scheme is robust and thus returns for any undirected graph given as input a meaningful output.