People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

Mouloud Mammeri University of Tizi Ouzou
Faculty of Science
Department of Mathematics

MSc in Operational Research
Methods and models of Decision Making

A dissertation submitted in partial fulfilment of the requirements of
the Master of Science in Operational Research

Theme:

The Study of Indifference Graphs

Presented by:
Ouafa Hammoutene
U.M.M.T.O.

Dissertation committee:

Aouane Mohouhand	Assistant Professor	UMMTO	Chairman
Oubakouk Lynda	Assistant Professor	UMMTO	Examiner
Sadi Bachir	Associate Professor	UMMTO	Supervisor

October 2016

Abstract

The concept of indifference has for long baffled the human mind and many debates have arised around the matter. Graph theory has dedicated a new class of graphs to represent indifference relations called indifference graphs. The latter is a set of vertices linked by edges whenever any two are equal in preference. This class of graphs is a subclass of interval graphs. Therefore, their greedy recognition algorithm makes good use of the principal properties of interval graphs to its advantage. Consequently, several computational problems which are initially NP-hard find themselves greedy algorithms to seek tangible solutions in the case of an indifference graph. The purpose of this work is to investigate the class of indifference graphs and its algorithmic properties. Moreover, it presents a program developed using the algorithms, introduced in the setting of this work, that recognizes chordal graphs, interval graphs, and indifference graphs; as well as solve a couple of computational problems.

Keywords: Graph theory, indifference graphs, algorithms, computational problems

Declaration

I, *Ouafa Hammoutene*, declare that this dissertation titled, "*The Study of Indifference Graphs*" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

Acknowledgements

"Appreciation is a wonderful thing. It makes what is excellent in others belong to us as well." –Voltaire

I would like to take this chance to express my deepest gratitude to everyone who has helped to make this dissertation possible.

Firstly, I am grateful to my supervisor, Dr. Bachir Sadi, Associate Professor at the University of Mouloud Mammeri, whose expertise, generous guidance and support made it possible for me to work on a topic of great interest to me.

My gratitude is also addressed to the board of examiners Mr. Aouane and Mrs. Oubakouk for accepting to examine this work and offering guidance and advice.

I am also thankful to the teachers of the Department of Mathematics that accepted to enlighten me with the right piece of advice.

Finally, I am greatly indebted to all those that have encouraged me and gave me all their support during the course of completion of this dissertation.

“Indeed, there is much in pure humanitarian culture, as opposed to rigid scientific training, which encourages absorption in the affairs of mankind, and more or less indifference to the unfathomed abysses of star-strown space that yawn interminably about this terrestrial grain of dust.”
- H. P. Lovecraft

Table of contents

List of figures

List of tables

List of algorithms

List of Abbreviations

AI	Artificial Intelligence
BFS	Breadth First Search
LBFS	Lexicographic Breadth First Search
MCS	Maximum Cardinality Search
PEO	Perfect Elimination Ordering

List of Notations

V	Set of the vertices of a graph G
E	Set of the edges of a graph G
$deg(v)$	Degree of a vertex; the number of vertices adjacent to v
$adj(v)$	Set of all the vertices of G adjacent to v
$\chi(G)$	The chromatic number of G

Introduction

There are numerous disciplines that require the ability to measure variables which are not as easy to measure as physical variables such as mass or density. Such disciplines vary from social sciences to artificial intelligence passing by marketing and seriation. In the interest to study such fields, it is required to acquire tools that permit the ability to reason about preference, agreement, and indifference. This has given rise to a theory of measurement which covers social scientific variables as well as physical variables. Within the setting of this work, graph theory is used to approach one of the measurement problems which is indifference.

There are two fundamental comparative value concepts which are preference and indifference. The notion of preference has a central role in many disciplines, including moral philosophy and decision theory. Correspondingly, indifference is best defined as the absence of preference as it arises from a failure to detect any distinction. This work emphasizes on the study of the structural properties of indifference.

Graph theory is a tool for formulating problems, making them precise, and defining fundamental interrelationships. In fact, the formulation of a problem in a precise way helps in understanding the problem. Once a problem has been modeled by a graph, the notions of graph theory can be used to define concepts which are useful in analyzing the problem. On the whole, the use of precise, graph theoretical reasoning provides tools to help in making decisions on problems arising from real-life situations or of practical nature. Therefore, in this work, graph theory is adopted to study the notion of indifference through a class of indifference graphs.

Indifference graphs are part of a larger family of graphs namely chordal and interval graphs. Therefore they are essential to the study of indifference graphs; especially when it comes to building a greedy recognition algorithm. For this, a particular characterization of indifference graphs resides in a special ordering of their vertices is crucial to the greedy algorithm.

Given a graph, it is essential to have the means to recognize an indifference graph as this class permits easy resolution approaches of some hard computational problems. Therefore, this work presents the fundamental notions of computational theory and algorithmic analysis to identify the necessary requirements for building a recognition algorithm. Accordingly, a program is developed to recognize all the classes of graphs previously cited as well as solve a couple of well-known computational problems.

The present work is composed of four chapters.

The first chapter introduces one of the measurability theory tools which is indifference and illustrates how graph theory offers adequate tools of representing such a relation between individuals, objects, or situations. It also presents several classes of graphs namely chordal graphs, interval graphs, and indifference graphs including numerous properties that characterize them. In addition, several examples are presented to illustrate fundamental concepts and characterizations.

In the second chapter, a descriptive of computational complexity and its importance are presented within its first chapter. More importantly, it presents greedy recognition algorithms to recognize chordal graphs, interval graphs and indifference graphs. In addition, it presents various illustrations of how the algorithms proceed.

The third chapter presents a couple of computational problems accompanied with simple greedy algorithms namely the graph coloring problem and the maximum matching problem. These problems are known to be NP-hard; however, for a particular instance, in the case of indifference graphs, they accept greedy algorithms to solve them.

Finally, the last chapter discusses the programming language used to code the program developed in the setting of this work. Moreover, it discusses the different data structures the program manipulates. It also presents a case study that showcases the ability of graph theory to model a given situation and how it provides tools to solve the problem; more importantly, it displays the performance of the program with the given results.

Chapter 1

Indifference graphs

This chapter introduces one of the measurability theory tools which is indifference and illustrates how graph theory offers adequate tools of representing such a relation between individuals, objects, or situations. It also presents several classes of graphs namely chordal graphs, interval graphs, and indifference graphs including numerous properties that characterize them. In addition, several examples are presented to illustrate fundamental concepts and characterizations.

1.1 The concept of indifference

There are many disciplines that require the ability to measure variables which are not as easy to measure as physical variables such as mass or density. The following list specifies some of such fields of study.

- Political science: understanding political fundamentals and cultural theory [58]
- Social science: understanding social inequalities [45]
- Economics: utility theory as a mechanism to quantify preference [25, 48]
- Marketing: analyzing consumer preferences and consumer behavior [18, 38, 42, 49]
- Seriation: organization of data in order to set complex interrelationships into a coherent pattern [45]
- Psychology: understanding and predicting decisions [38]
- Artificial intelligence: analyzing and predicting AI motivations and/or decisions [4]

In the interest to study such fields, it is required to acquire tools that permit the ability to reason about preference, agreement, and indifference. This has given rise to a theory of measurement which covers social scientific variables as well as physical variables. Within the setting of this work, graph theory is used to approach one of the measurement problems which is indifference [45].

There are two fundamental comparative value concepts which are “better” and “equal in value to” which are respectively equivalent to *preference* and *indifference* [53]. In this work, the relations of preference and indifference between alternatives are denoted by the symbols $\succ$ and $\sim$ respectively or alternatively by P and I. Indifference is best defined as the absence of preference as it arises from a failure to detect any distinction [50]. The notion of preference has a central role in many disciplines, including moral philosophy and decision theory. The study of the structural properties of preference can be traced back to Aristotle’s book *Topics* [54].

Let $\succ$ be the binary relation on X representing preference between two alternatives x and y; it is noted as follows.

$$x \succ y, \qquad x, y \in X$$

In this notation, it is said that x ‘is preferred to’ y. Moreover, an indifference relation between two alternatives x and y is noted as follows. Let $\sim$ be the binary relation representing the later relation.

$$x \sim y \Leftrightarrow (\textit{not } x \succ y,\ \textit{not } y \succ x), \quad x, y \in X \tag{1.1}$$

Theorem 1.1. *[20]*
Suppose $\succ$ on X is asymmetric. Then,

1. *$\succ$ is irreflexive (not $x \succ x$);*
2. *exactly one of $x \succ y$, $y \succ x$, $x \sim y$ holds for each x, y in X;*
3. *$\sim$ is reflexive ($x \sim x$) and symmetric ($x \sim y \Rightarrow y \sim x$).*

The measurement of preference corresponds to an assignment of numbers which ‘upholds ’ the expressed inclination. For this, let X be the set on which $\succ$ is defined. To measure preference, an assignment $u(x)$ is appointed to each $x \in X$ such that x ‘is preferred to’ y if and only if x gets a higher value than y; i.e. for all x, $y \in X$,

$$x \succ y \Leftrightarrow u(x) > u(y). \tag{1.2}$$

Similarly, let Y be the set on which $\sim$ is defined. To measure indifference, an assignment $u(x)$ is appointed to each $x \in Y$ such that a subject ‘is indifferent’ to x and y if and only if x gets an equal value as y; i.e. for all x, $y \in Y$,

$$x \sim y \Leftrightarrow u(x) = u(y). \tag{1.3}$$

It is important to note that the existence of a function u satisfying (1.5) implies that indifference is a transitive relation [44]. Transitivity of the indifference relation as well as the strict preference relation between alternatives are necessary for proving utility theory [54].

Indifference transitivity. If $x \sim x'$ and $x' \sim x''$, then from the definition of indifference above (equation 1.1), $x \succ x'$ and $x' \succ x''$ so that $x \succ x''$, and also $x'' \succ x'$ and $x' \succ x$ so that $x'' \succ x$. Taken together, this shows that if $x \sim x'$ and $x' \sim x''$ then $x \sim x''$. ■

The following examples illustrate indifference intransitivity.

Example 1 [Intransitive indifference] [20]

1. A subject who prefers sugarless coffee will be indifferent between a cup of sugarless coffee and a cup of coffee with one grain of sugar in it, especially that the subject will notice no difference in taste or effect between the two. Furthermore, he might be indifferent between a cup of coffee with x grains of sugar and another cup cup with $x+1$ grains of sugar for ($x \in \mathbb{N}$). However he prefers a cup of sugarless coffee to a cup with 1000 grains of sugar, in a manner that $\sim$ is intransitive. Transitivity of indifference would imply that the subject is indifferent between a cup of sugarless coffee and a cup of coffee with five spoons of sugar. This example was suggested by Luce, in 1956 [36].

2. In consumer preference theory and utility theory, an individual must judge between multidimensional alternatives. If $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_n)$ differ only on one dimension (e.g., $x_j = y_j$ for all $j > 1$ and $x_j \neq y_j$), even a small difference on this dimension may give rise to strict preference. However, approximately offsetting differences on several dimensions may give rise to indifference areas that lead to intransitive indifference. This is shown in the following two-dimensional example suggested by the work of Armstrong and May.
A subject is going to buy a car and has no definite preference between (Ford, \$2600) and (Chevrolet, \$2700), and also has no definite preference between (Ford, \$2600) and (Chevrolet, \$2705). However, (Chevrolet, \$2705) $\prec$ (Chevrolet, \$2700).

3. A subject has to vote on an allocation of funds for a community facility. He feels that \$200,000 is a good sum and his preference decreases as he moves away from this amount in either direction. Then perhaps $\$190,000 \prec \$191,000$ and $\$206,000 \prec \$205,000$. But judgment may be more difficult when you try to compare two amounts on different sides of his \$200,000 peak. For instance, he may feel that $\$191,000 \sim \$205,000$ and that $\$191,000 \sim \$206,000$.

Examples such as (1.2) and (1.3) indicate that indifference corresponds more to closeness than to equality [44]. In the case of preference, this led Luce [36] to ask for an assignment of numbers $u(x)$ fulfilling

$$x \succ y \Leftrightarrow u(x) > u(y) + \delta, \quad x, y \in X, \tag{1.4}$$

where δ is a positive number measuring closeness or proximity. In other words, a subject prefers x to y if and only if x is 'sufficiently ' better than y. Furthermore, in the case of indifference, examples such as (1.1), lead to the following assignment.

$$x \sim y \Leftrightarrow |u(x) - u(y)| \leq \delta, \quad x, y \in Y \tag{1.5}$$

The quantity δ represents the amount of fuzziness that must be filtered out. This enables subjects to be indifferent about events that differ by insignificant amounts [25]. It is important in measurement to acknowledge the circumstances under which there exists an assignment u as in (1.5) because it indicates when a singular measuring method or scaling data can be carried out. For this, Luce [36] introduced a generalization of simple orderings which he called semiorders. A semiorder, as defined by Scott and Suppes [51], is a binary relational system (X,P) that satisfies the following axioms for all x, y, z, $w \in X$,

(A1) P is irreflexive $\Leftrightarrow$ not xPx

(A2) $xPy \ \wedge \ zPw \ \Rightarrow \ xPw \ \vee \ zPy$

(A3) $xPy \ \wedge \ yPz \ \Rightarrow \ xPw \ \vee \ wPz$

Lehrer and Wagner [32] affirm that intransitivities in preference and indifference are harmless and probably an inevitable part of the human condition. They add that when such relations must function as principles of selection, however, intransitivity can lead to irrational choices. Furthermore, Rothbard [48] claims that indifference can never be demonstrated by action because every action necessarily signifies a choice, and every choice signifies a definite preference. Thus, action specifically implies the opposite of indifference. If a person is indifferent between two alternatives, then he cannot and will not choose between them. This concept has been criticized as Armstrong [5] points out that it is possible for an individual to be indifferent between two alternatives and yet choose one over the other [48]. This can be demonstrated by the famous Buridan's Ass illustration of a paradox in philosophy in the conception of free will.

The next section introduces graph theoretic tools presenting a better approach to the question about measuring judgments of indifference using the aforementioned axioms.

1.2 Graph theory and indifference

Graph theory is a branch of mathematics with a wide-ranging applicability being a powerful modeling and solving tool. Certain problems in sociology, communication science, computer science, psychology, physics, and linguistics can be formulated as problems in graph theory. Besides, many branches of mathematics, such as group theory, probability, and topology, have close connections with graph theory .

Some riddles and several problems of a practical nature have been involved in the development of various topics in graph theory. The following list illustrates some of these problems [7].

- The Königsberg bridge problem has inspired the development of Eulerian graphs.
- The "Around the World" game of Sir William Hamilton inspired the Hamiltonian graph theory.

- The theory of acyclic graphs was developed for solving problems of electrical networks.
- The study of graph trees was developed for enumerating isomers of organic compounds.
- The four-color problem formed the basis of the development of planarity in graph theory and combinatorial topology.

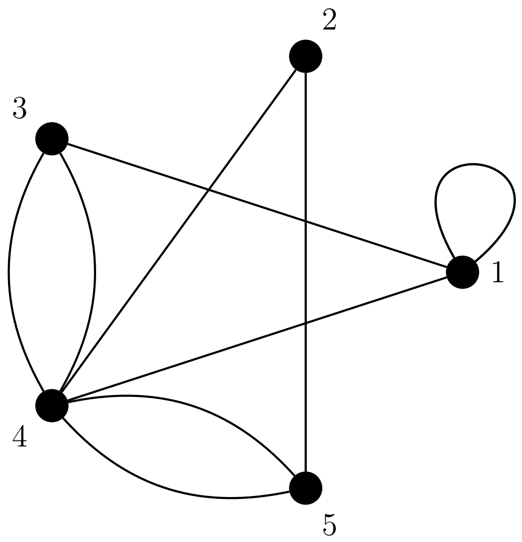
The use of precise, graph theoretical reasoning provides tools to help in making decisions on problems arising from the previously stated disciplines in section (1.1). Graph theory is a tool for formulating problems, making them precise, and defining fundamental interrelationships. In fact, the formulation of a problem in a precise way helps in understanding the problem. Once a problem has been modeled by a graph, the notions of graph theory can be used to define concepts which are useful in analyzing the problem. Thus, the formulation of a problem gives insight on why the problem is difficult to solve. For example, the formulation of a problem posed by the New York City Department of Sanitation as a graph coloring problem indicates why the problem is difficult; graph coloring is a hard problem to solve [45].

Definition 1.1 (Graph [27])**.** *A graph is a pair of sets (V,E) where V is a set (nonempty) of **vertices** and E is a set (possibly empty) of unordered pairs of elements of V called **edges**. Vertices are usually displayed by points in the plane representing objects, individuals, or situations. Similarly, edges are usually lines joining points (vertices) representing relations, preferences, or situations.*

*A graph may be **finite** or **infinite**, depending on whether the set V is finite or infinite. Also, a graph is **simple** if it does not allow that two vertices are joined by more than one edge (otherwise it's a multigraph).*

*If x and y are vertices of a graph G then x is **adjacent** to y if there is an edge between x and y (they are also said to be neighbors). A vertex x is **incident** with and edge e if x is an endpoint of e. An edge that is incident with only one vertex is called a **loop** and a graph that allows loops is called a pseudograph.*

In this work, the type of graphs considered are finite, undirected, simple, and loopless. The figure (1.1) illustrates an example of a pseudo-multi graph.



$V = \{1, 2, 3, 4, 5\}$
$E = \{(1, 1); (1, 3); (1, 4); (2, 4);$
$(2, 5); (3, 4); (4, 5)\}$
Vertex (1) is incident with a loop
Vertices (3) and (4) as well as
(4) and (5) allow multiple edges
joining them

Figure 1.1: Graph $G = (V, E)$, a pseudo-multi graph.

The next example illustrates some situations that can be modeled using graph theory.

Example 2

1. Let various traffic streams or directions of traffic be vertices of a graph and draw an edge x to y if the two traffic streams x and y are compatible. The compatibility graph like so is used to phase traffic lights [45].
2. Let the vertices of a graph be alternatives among which an individual is expressing his preferences, and draw an edge between two vertices if and only if the individual is indifferent between the two alternatives. The indifference graph obtained is used to measure the individual's opinion [45].
3. Let the vertices of a graph be possible code words for a rapid communication system, and draw an edge between two code words if it is possible to confuse them. This confusion graph is used to find codes with large capacity [45].

Roberts [45, 44, 43], Golumbic [25] and others analyze intransitive indifference using notions of graph theory. For instance, let $G = (V, E)$ be an indifference graph with points in V and an edge between x and y if and only if $x \sim y$.

It is important to note that the class of indifference graphs is a subclass of the class of interval graphs which is in turn a subclass of the class of chordal or triangulated graphs. Therefore, the following sections are devoted to the presentation of the two aforementioned classes of graph.

1.2.1 Chordal graphs

Hajnal and Suranyi (1958) were the first to introduce chordal graphs which are also called triangulated graphs. A graph G is a chordal graph if every cycle of length greater than three possesses a chord, which is an edge joining two nonconsecutive vertices of the cycle.

Chordality is a hereditary property inherited by all the induced subgraphs of a chordal graph because, otherwise, it contains a cycle of length greater than three with no chord and thus the graph would accept a cycle of length greater than three that has no chord.

Chordal graphs are also called perfect elimination graphs because a graph G is chordal if and only if it has a perfect elimination order (theorem 2.1). This is a fundamental characteristic which resides in a particular order of the vertices of chordal graphs.

Definition 1.2 (Perfect Elimination ordering [46])**.** *Let $G = (V, E)$ be a graph. An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of the vertices of V is called a perfect elimination order if each v_i is a simplicial vertex of the subgraph of G induced by $\{v_i, \ldots, v_n\}$.*

Simplicial vertices are important elements of chordal graphs. Figure (1.2) shows a chordal graph; one of its possible perfect elimination orders would be $\sigma = (4, 3, 2, 1)$, obtained by simplicial vertex elimination. On the other hand, figure (1.3) displays an example of a graph that is not chordal as it has no simplicial vertex.

Definition 1.3 (Subgraph)**.** *A subgraph G_1 of a graph G is a graph whose set of vertices and set of edges are all subsets of G.*
A graph G_1 is an induced subgraph of G if G_1 is isomorphic to a graph whose set of vertices V_1 is a subset of the vertex set V of G and whose set of edges E_1 consists of all the edges of G that are incident to vertices in V_1.

Definition 1.4 (Simplicial vertex)**.** *A vertex is simplicial if and only if its neighborhood is a complete subgraph, which is a clique that is not necessarily maximal.*

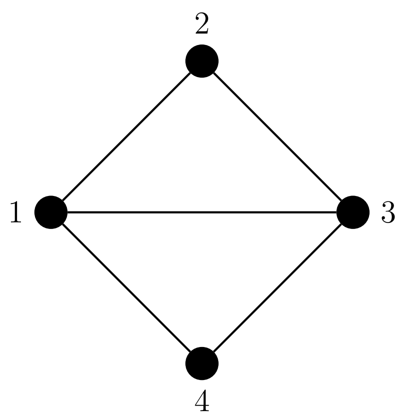


Figure 1.2: A chordal graph.

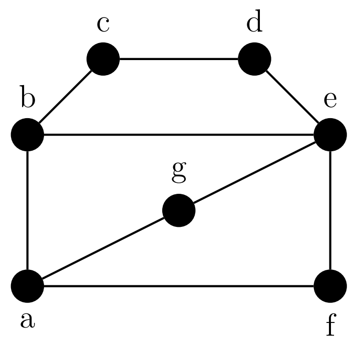


Figure 1.3: A graph that is not chordal.

Theorem 1.2. *[21] A graph G is chordal if and only if it has a perfect elimination order. Furthermore, in such a case, any simplicial vertex can start a perfect elimination order.*

Lemma 1.2.1 (Dirac, 1961)**.** *Every chordal graph G has a simplicial vertex. If G is not a clique, then it has two nonadjacent simplicial vertices.*

Chordal graphs can be recognized in two repeated steps until no vertex remains [47]:

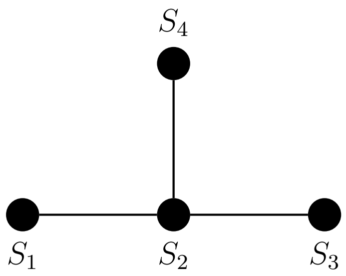
1. detect a simplicial vertex
2. eliminate it from the graph

So, if all the vertices of a graph are eliminated then the graph is chordal and if at some stage no simplicial vertex is detected then, the graph is not chordal. This approach is used to build a strong greedy recognition algorithm of this class of graphs (see chapter 2).

1.2.2 Interval graphs

The intersection graph of a family of nonempty sets is obtained by representing each set in the family by a vertex and connecting two vertices by an edge if and only if their corresponding sets have nonempty intersections. This graph is associated with $\mathcal{F} = \{S_1, S_2, \ldots, S_p\}$ a family of sets in a manner that the vertices of the graph represent the sets in $\mathcal{F}$, and there is an edge between two sets S_i and S_j if and only if they have nonempty intersection. There are a large number of applications within which intersection graphs are used. Example (3) briefly illustrates one of the numerous applications [45, 44].

> **Example 3** [45] A collection of large corporations and the network of corporate interlocks can be represented by an intersection graph (Levine 1976). For instance, the corporations of “Fortune 800” can be set up as a family of sets. For the ith corporation, S_i would consist of members of the Board of Directors of this corporation. In order to understand the network of corporate interlocks, the graph of the family of S_i is studied.



$S_1 = \{a, b, c, d\}$
$S_2 = \{c, e, h\}$
$S_3 = \{e, f, g\}$
$S_4 = \{h\}$

Figure 1.4: The intersection graph of a family of sets representing corporations [45].

Theorem 1.3 (Marczewski 1945)**.** *Every graph is isomorphic to the intersection graph of some family of sets.*

Proof. Given $G = (V, E)$, let $\{(u, v)/(u, v) \in E\} \cup \{u\}$.
It is easy to see that for all $u \neq v$ in V, $(u, v) \in E \Leftrightarrow S(u) \cap S(v) \neq \emptyset$.
Thus, G is isomorphic to the intersection graph of the sets $S(u)$. ∎

Considering a special class of sets, leads to an interesting application of intersection graphs. The intersection graph of a family of intervals on the real line is called an interval graph. In other words, it is the intersection graph of a family of intervals of a linearly ordered set. Interval graphs have a wide set of applications; several of their examples can be found in [11].

Definition 1.5 (Interval graph)**.** *An undirected graph $G = (V, E)$ is an interval graph if its vertices can be put into a one-to-one correspondence with a set of intervals of a linearly ordered set, such that two vertices are connected by an edge if and only if their corresponding intervals have nonempty intersection (figure 1.5), i.e.* $\forall x, y \in X$,

$$(x, y) \in E \Leftrightarrow I(x) \cap I(y) \neq \emptyset \tag{1.6}$$

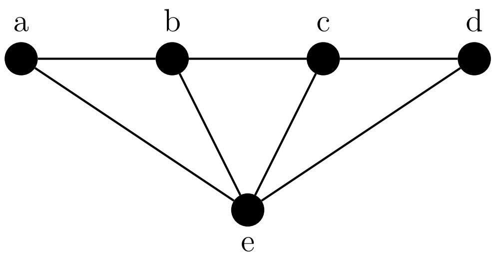


Figure 1.5: An interval graph $G = (V, E)$.

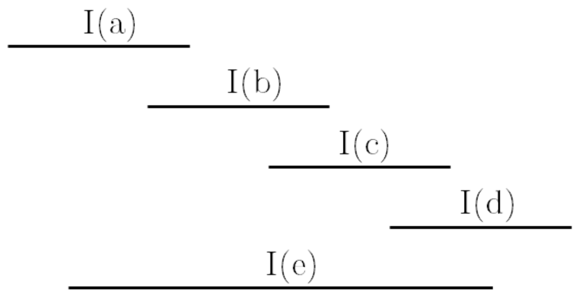


Figure 1.6: The interval representation of the graph G.

While it is easy to observe that a C_3 (a cycle composed of 3 vertices) is an interval graph, the following example considers the case of a C_4 and illustrates what it is not an interval graph.

Example 4 A C_4 is not an interval graph [45].
In this example, if there were an interval assignment satisfying (1.6), then $I(a)$ and $I(b)$ would have to overlap, since there is an edge between a and b. Without loss of generality, $I(b)$ is slightly to the right of $I(a)$ (NB: $I(b)$ cannot completely lie inside $I(a)$ because otherwise $I(c)$ will not overlap $I(b)$ without overlapping $I(a)$. Similarly, $I(a)$ cannot lie inside $I(b)$.) $I(c)$ overlaps $I(b)$ but not $I(a)$, so it is represented as in figure (1.7). Next, $I(d)$ must overlap both $I(a)$ and $I(c)$, but not $I(b)$. Unfortunately, $I(d)$, under such circumstances cannot be represented. A similar argument holds for any C_k with $k \geq 4$.

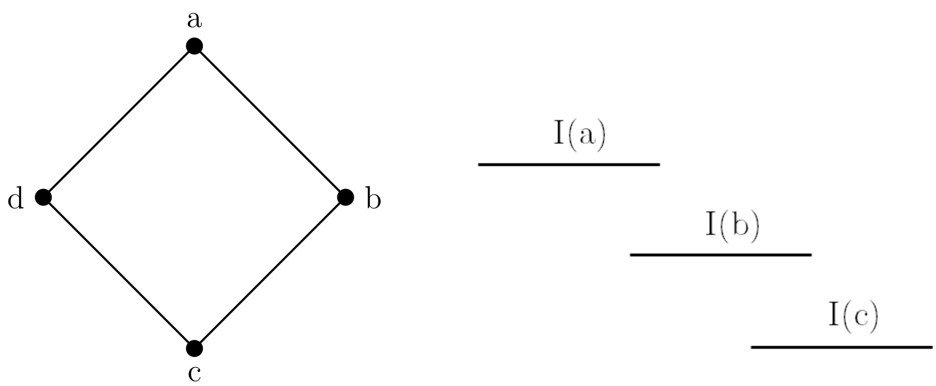


Figure 1.7: Illustrating that a C_4 is not an interval graph.

A graph $K = (W, F)$ is a subgraph of a graph $G = (V, E)$ if $W \subseteq V$ and $F \subseteq E$. K is said to be an induced subgraph if F consists of all the edges from E joining vertices in W. It is self-evident that if G is an interval graph, then every induced subgraph must also be an interval graph. Nonetheless, it is not the case of every subgraph. Subsequently, an interval graph has the property that no C_4 graph, is an induced subgraph. This implies that interval graphs are chordal.

Theorem 1.4 (Hajós, 1957)**.** *An interval graph G is a chordal graph.*

Proof. Let $G = (V, E)$ be an interval graph having a chordless cycle (v_1, v_2, ..., v_{k-1}, v_1) with $k > 3$ and let $I(k)$ denote the interval representing the vertex v_k. Choose a point $p_i \in I(i-1) \cap I(i)$, for $i = 1, 2, \ldots, k-1$. Since $I(i-1)$ and $I(i+1)$ do not overlap, the points p_i constitute a strictly increasing or a strictly decreasing sequence. Therefore, it is impossible for $I(1)$ and $I(k-1)$ to intersect. This contradicts the assumption that $(v_1, v_{k-1}) \in E$. ■

However, not all chordal graphs are interval graphs, figure (1.8) illustrates a C_4-free graph that is not an interval graph. In terms of a forbidden subgraph configuration theorem, which characterizes a class of graphs by stating which configurations cannot appear as induced subgraphs, indifference graphs do not contain the graphs stated in theorem (1.5) as induced subgraphs.

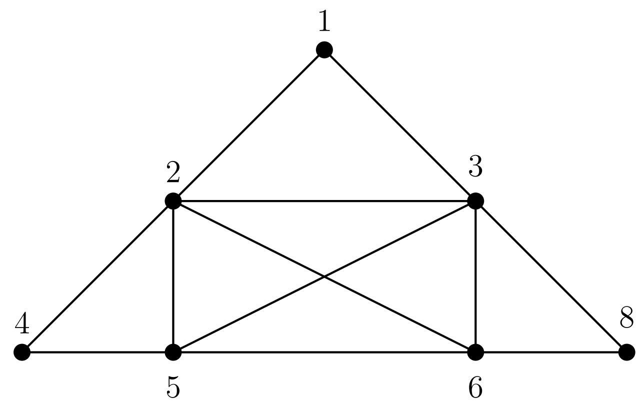
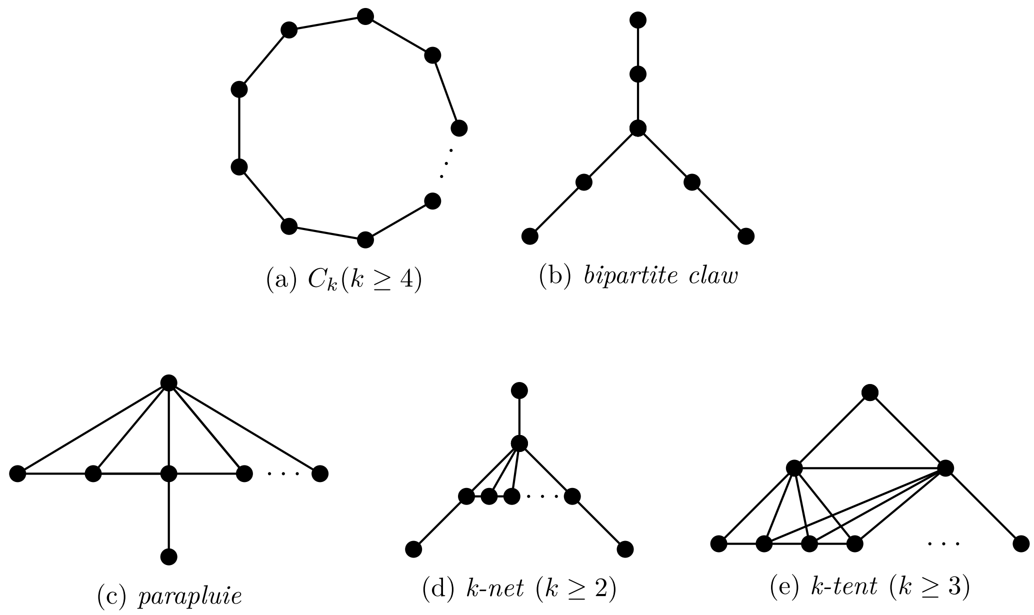


Figure 1.8: A chordal graph that is not an interval graph.

Theorem 1.5 (Forbidden subgraph configurations [33]). *A graph G is an interval graph if and only if it does not contain any of the graphs* (a), (b), (c), (d), (e), *illustrated in figure (1.9), as induced subgraphs.*



(a) $C_k (k \geq 4)$ (b) *bipartite claw*

(c) *parapluie* (d) *k-net* ($k \geq 2$) (e) *k-tent* ($k \geq 3$)

Figure 1.9: Interval graph forbidden subgraphs.

The following fundamental results concerning interval graphs are relied upon to characterize interval graphs and develop recognition algorithms.

Theorem 1.6. *[24] A graph G is an interval graph if and only if G is a chordal graph and its complement $\bar{G}$ is a comparability graph.*

Definition 1.6 (Comparability graph [41])**.** *A graph G is called a comparability graph if there exists a partial order $<$ on the set of its vertices such that for u and v are adjacent in G if and only if $u < v$ or $v < u$. A comparability graph $G = (V, U)$ is transitively orientable, i.e. its edges can be directed such that if $(a, b) \in U$ and $(b, c) \in U$ are directed edges, then there exists the directed edge $(a, c) \in U$.*

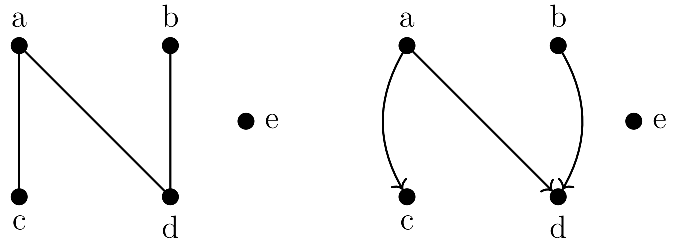


Figure 1.10: The complement of the graph of figure (1.5) $\bar{G}$ and a transitive orientation for the complement.

Theorem 1.7. *[24] A graph G is an interval graph if and only if the maximal cliques of G can be linearly ordered such that, for every vertex v of G, the maximal cliques containing vertex v occur consecutively.*

The clique incidence matrix M of a graph G is defined as follows. The rows of M correspond to the maximal cliques (see §2.2.1), and the columns to the vertices of G such that

$$m_{ij} = \begin{cases} 1 & , \textit{ vertex } j \textit{ belongs to the maximal clique } i; \\ 0 & , \textit{ otherwise.} \end{cases}$$

Furthermore, if there is an ordering of maximal cliques which is consecutive as in theorem (1.7), the corresponding ordering of rows of M gives rise to a matrix with the 1's in each column appearing consecutively. A matrix A is said of 0's and 1's has the consecutive 1's property (according to columns) if it is possible to permute the rows so that the 1's in each column appear consecutively [45, 21].

Theorem 1.8. *[21] A graph G is an interval graph if and only if its clique incidence matrix M has the consecutive 1's property.*

The following example illustrates the previous theorem.

> **Example 5** The following matrix is the clique incidence matrix for the graph C_4 shown in figure (1.7). It is self-evident that there is no permutation of the rows so that the 1's in each column appear consecutively.

$$M = \begin{array}{c} \\ ab \\ bc \\ cd \\ ad \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Theorem 1.9. *[41, 35] A graph $G = (V, E)$ is an interval graph if and only if there exists a linear order $<*$ on V such that for every choice of vertices u, v, w with $u < *v < *w$ and $v < w$, $(u, w) \in E$ implies $(u, v) \in E$, i.e.*

$$\forall u, v, w \in V,\ u < *v < *w \text{ and } (u, w) \in E \Rightarrow (u, v) \in E. \tag{1.7}$$

The ordering $v_1 < *v_2 < * \ldots < *v_n$ satisfying (1.7) is referred to as canonical; it is the principal characteristic of interval graphs that greedy recognition algorithms use.

The following example describes one of the various applications of interval graphs in modeling real-life situations.

Example 6 The mobile frequency assignment problem [45].
Mobile radio systems, widely used by police patrols, mountaineers, fishermen, emergency units, construction units, etc., operate in different zones. Each zone receives a band of frequencies which can be used within it. These bands generally form unions of intervals. Thus, the mobile phones in one zone can cause interference with those in another zone. In this case, their bands of frequencies should not overlap. When assigning bands to zones, one wants to take these conflicts into account, and also meet certain requirements on minimal bandwidth for given zones.
According to Gilbert (1972), this problem can be formulated using graph theory. The vertices of a conflict graph are the zones, and two zones are adjacent if and only if they conflict. Then, each zone i should be assigned a band $B(i)$ (an interval) in a manner that if there is an edge between zones i and j, it means that $B(i) \cap B(j) = \emptyset$. Furthermore, each $B(j)$ obtained should be of minimal length.
Modeled in this manner, the mobile frequency assignment problem can be reduced to the traffic light phasing problem (presented by [45] in § 3.6, p.22). Roberts considers the feasible green light assignment on the complementary graph of the conflict graph. Moreover, the mobile frequency assignment is reducible to a graph coloring problem discussed in chapter 3. More about frequency assignment problems can be found in [3].

Interval graphs and indifference

Interval graphs arise likewise in the measurement of preference and indifference. The following definition describes how an interval graph may represent a situation of indifference relations.

Definition 1.7. *[20] A graph $G = (X, E)$, where X is the set of vertices and E the set of edges such as an edge exists between x and y $(x, y \in X)$ if and only if $x \sim y$, is an interval graph if and only if a real interval $I(x)$ can be assigned to each x in X so that $I(x)$ and $I(y)$ intersect if and only if $x \sim y$, i.e.*

$$x \sim y \Leftrightarrow I(x) \cap I(y) \neq \emptyset, \ \forall x, y \in X. \tag{1.8}$$

Lekkerkerker and Boland [33] affirm that an interval graph is chordal and asteroidal triple-free (theorem 1.10). An asteroidal triple in a graph is a subset of three mutually non-adjacent vertices such that for every pair of vertices, there is a path in the graph that does not pass through the third vertex or any of its neighbors. Such vertices are called extreme vertices [44]. For example, in the graph of figure (1.8) the three vertices 1, 4, and 8 form an asteroidal triple.

Theorem 1.10. *[33] Let X be a finite set and suppose $\sim$ is a binary relation on X that is reflexive and symmetric. Then (X,$\sim$) is an interval graph if and only if the following never hold for (X,$\sim$):*

(a) $(x_1 \sim x_2,\ x_2 \sim x_3,\ \ldots,\ x_{m-1} \sim x_m,\ x_m \sim x_1)$ and not $x_i \sim x_j$ for any other $i \neq j (1 \leq i, j \leq m)$ and $m > 3$;

(b) for each pair in $\{x, y, z\}$ $(x, y, z \in X)$, there is a path connecting the pair and the third vertex is $\sim$ to a vertex in the path.

Modeling real-life situations, involving linear dependencies, can be particularly challenging, therefore, interval graphs are invaluable tools to model these situations. They find applications to archaeology, biology, psychology, sociology, and many others. Seriation is an important element in the social sciences as it attempts the organization of data by putting complex interrelationships into a coherent pattern, namely a sequence or in serial order (chronologically, opinion-wise, etc.). The following are a few applications of seriation [44].

- Archaeology: sequence dating a collection of artifacts (example 7).
- Psychology: putting certain traits in a developmental order; ordering individuals according to their opinions.
- Political science: ordering political candidates from liberal to conservative.

Example 7 Application of interval graphs in archaeology [45]**.**
One approach to seriation is the interest in overlap information. For instance, archaeology tries to learn whether or not the time intervals during which two artifacts existed overlapped. This information can be obtained from observing graves. An assignment of time intervals is set up so that artifact a and artifact b were found in common in some grave if and only if the time interval associated with a overlaps with the time interval associated with b. Such an assignment can be obtained if and only if the 'found in common in some grave' graph is an interval graph. The intervals illustrate a possible chronological order. Such an approach to seriation in archaeology was introduced by Kendall (1963).

1.2.3 Indifference graphs

Section 1 of this chapter highlights how the use of graph theory provides invaluable tools in one of the measurement problems which is the measurement of indifference, in particular intransitive indifference. In this perspective, a graph G can be defined such as its set V of vertices represents objects or alternatives being compared and the set E of edges where an edge exists between two vertices if and only if a subject is indifferent between both vertices. Measurement of intransitive indifference, as previously stated, corresponds to the assignment of a real number $f(x)$ to each x in V so that for all x, y in V,

$$x \sim y \Leftrightarrow |f(x) - f(y)| < \delta. \tag{1.9}$$

The following example displays an indifference graph $G = (V, E)$ where an assignment of real number $f(x)$ is associated to V.

Example 8 An indifference graph [44].
If $\delta = 1$, an assignment of numbers satisfying (1.9) is given by $f(a) = 0$, $f(b) = 0.3$, $f(c) = 1.1$, $f(d) = 1.4$, and $f(e) = 0.7$

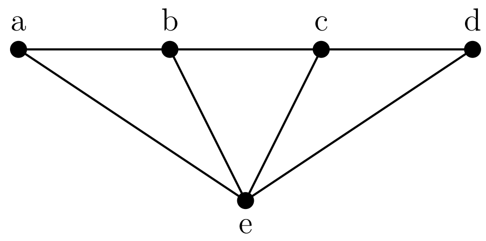


Figure 1.11: An indifference graph $G = (V, E)$.

Figure (1.12) illustrates a graph that is not an indifference graph; the argument is displayed in figure (1.13).

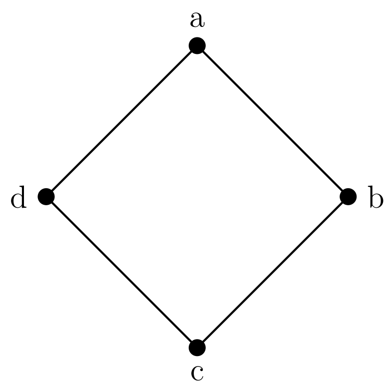


Figure 1.12: The graph C_4 is not an indifference graph.

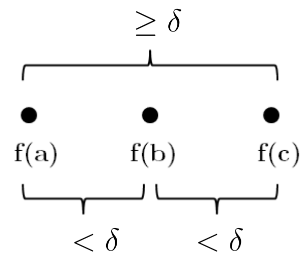
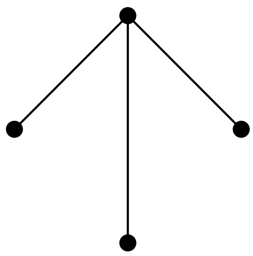


Figure 1.13: The argument that C_4 is not an indifference graph [44].

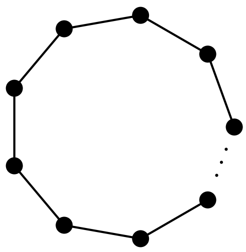
If there exists an assignment f that satisfies (1.13), then $f(b) \geq f(a)$. Since a and b are joined by an edge and similarly b and c, the numbers $f(a)$ and $f(b)$ must be within δ and so must the numbers $f(b)$ and $f(c)$. However, a and c are not joined by an edge, so the numbers $f(a)$ and $f(c)$ must be at least δ apart. It follows that there numbers are arranged as in figure (1.13). What of $f(d)$? It must be within δ of $f(a)$ and $f(c)$, but not within δ of $f(b)$. Thus, this is impossible and C_4 is not an indifference graph. Neither is C_k, $k \geq 4$ [44].

In terms of a forbidden subgraph configuration theorem, which characterizes a class of graphs by stating which configurations cannot appear as induced subgraphs, indifference graphs do not contain the graphs stated in theorem (1.5) as induced subgraphs.

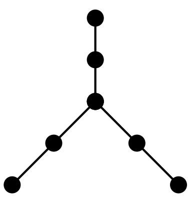
Theorem 1.11 (Forbidden subgraph configurations [43])**.** *A graph G is an indifference graph if and only if it does not contain any of the graphs (a), (b), (c), (d), (e) illustrated in figure (1.14), as induced subgraphs.*



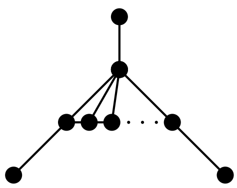
(a) $K_{1,3}$ *(claw)*



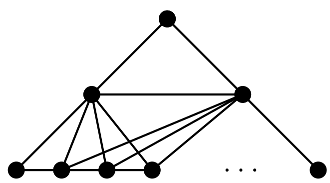
(b) $C_k (k \geq 4)$



(c) *bipartite claw*



(d) *k-net* $(k \geq 2)$



(e) *k-tent* $(k \geq 3)$

Figure 1.14: Indifference graph forbidden subgraphs

Let $G = (V, E)$ be an interval graph and let $\{I(v) = [a_v, b_v]\}$ be an interval representation of G where a_v and b_v $(a_v < b_v)$ are the left and right endpoint of the interval $I(v)$. G is called a unit interval graph if all the intervals in the representation are of the same length, i.e. they have unit length. Likewise, G is called a proper interval graph if no interval in the representation is properly contained in another. Evidently, unit interval graphs are proper interval graphs as illustrated through theorem (1.12)(see also [25, 13]). Roberts [43] has proved the fundamental result that implies that unit interval graphs, proper interval graphs, and indifference graphs are equivalent.

Theorem 1.12. *[43] A graph G is a unit interval graph if and only if it is an indifference graph.*

Proof. There exists a function satisfying equation (1.5) with $\delta > 0$ if and only if there is such a u with $\delta = 1$. Let

$$I(a) = \left[u(a) - \frac{1}{2},\ u(a) + \frac{1}{2}\right].$$

Then, it is easy that u satisfies (1.5) with $\delta = 1$ if and only if G is the intersection graph of the closed unit intervals $I(a)$. ∎

Theorem 1.13. *[43] Let $G = (V, E)$ be a graph. The following conditions are equivalent:*

(i) G is an indifference graph.

(ii) G is a comparability graph.

(iii) G is an interval graph containing no induced claw($K_{1,3}$).

(iv) G is a unit interval graph.

(v) G is a proper interval graph.

Indifference graphs are also characterized by an ordering of their vertices just like interval graphs. Aside from the fact it satisfies (1.7), it also satisfies a stronger property (theorem 1.14) which is elementary for all optimal greedy recognition algorithms.

Theorem 1.14. *[35] A graph $G = (V, E)$ is an indifference graph if and only if there exists a linear order $\lessdot$ on V such that for every choice of vertices u, v, w,*

$$u \lessdot v \lessdot w \text{ and } (u, w) \in E \Rightarrow (u, v) \in E \text{ and } (v, w) \in E. \tag{1.10}$$

Proof. Let $\lessdot$ be a linear order on V with the properties specified in (1.10). In particular, $\lessdot$ satisfies the condition specified in theorem (1.9), and so G is an interval graph.

By theorem (1.6), to prove that G in an indifference graph, G has to contain no induced claw (figure 1.14.a).

Suppose that G contains an induced claw with vertices a, b, c, d and edges ab, ac, and ad so that a contradiction may arise.

First, note that vertex a cannot precede b, c, and d in the linear order $\lessdot$; otherwise, let z stand for the vertex among b, c, and d that comes first in the order $\lessdot$. Since $(z, a) \in E$, it follows that z is adjacent to the remaining two such that it contradicts that $\{a, b, c, d\}$ induces a claw.

Now, the symmetry of the claw allows to assume, without loss of generality, that b precedes a, c, and d and that d follows a, b, and c in $\lessdot$. However, the ordering $b \lessdot c \lessdot a$ implies $(b, c) \in E$; similarly, the ordering $a \lessdot c \lessdot d$ implies that $(c, d) \in E$. Either case leads to a contradiction.

Conversely, let G be an indifference graph. In particular, G is an interval graph. For every vertex x of G, let $I(x) = [a_x, b_x]$ stand for the corresponding interval. Define a linear order $\lessdot$ on V by setting

$$u \lessdot v \ whenever \ (a_u < a_v) \ or \ \left[(a_u = a_v) and (b_u \leq b_v)\right]. \tag{1.11}$$

Let u, v, and w be arbitrary vertices in G satisfying $u \lessdot v \lessdot w$ and assume that $(u, w) \in E$ (the intervals $I(u)$ and $I(w)$ overlap). Now the assumption that $u \lessdot v \lessdot w$, together with (1.11), implies that

$$a_u \leq a_v \leq a_w.$$

In case $a_u = a_v$, the conclusion is immediate; therefore assume that

$$a_u < a_w.$$

Since $I(u)$ and $I(w)$ overlap, it must be the case that

$$a_w \leq b_u.$$

But, $a_v \leq a_w$ guarantees that $a_v \leq b_u$, and so $(u, v) \in E$. Next, since $I(u)$ cannot properly contain $I(v)$, $a_u \leq a_v$ guarantees that $b_u \leq b_v$. It follows that $a_w \leq b_v$, implying that $(v, w) \in E$. ■

Theorem (1.14) entails the following elementary results.

Corollary 1.14.1. *[35] Let $G = (V, E)$ be an indifference graph and let $\lessdot$ be a linear order on V satisfying (1.10). For every choice of subscripts i, j with $1 \leq i < j \leq n$ and $v_i v_j \in E$, the vertices v_i, v_{i+1}, $\ldots$, v_j are pairwise adjacent.*

Proof. To see this, let v_p and v_q be arbitrary vertices with $i \leq p < q \leq j$. Now the fact that v_i and v_j are adjacent, together with (1.10), imply that v_p and v_q are adjacent, and so, by (1.10) again, v_p and v_q must be adjacent, as claimed. ■

Corollary 1.14.2. *[35] An interval graph G is an indifference graph if and only if a canonical ordering $<*$ of the vertices of G satisfies (1.10).*

Proof. If a canonical ordering $<*$ of G satisfies (1.10), then by theorem (1.14), G is an indifference graph. Conversely, if G is an indifference graph, then by theorem

(1.14), an ordering $\lessdot$ of its vertices satisfying (1.10) is found. But, $\lessdot$ also satisfies (1.7) and the conclusion follows. ■

In order to obtain a characterization of indifference graphs leading to an optimal recognition algorithm, consider the following.

- An interval graph $G = (V, E)$ with a canonical ordering $<*$;

- for every i ($i \in \{1, \ldots, n\}$), define First[i]=min$\{i, k\}$ such that $(v_i, v_k) \in E$; and

- for every i ($i \in \{1, \ldots, n\}$), define Last[i]=max$\{i, k\}$ such that $(v_i, v_k) \in E$.

Thus, the following theorem is central to the indifference graph recognition algorithm.

Theorem 1.15. *[35] Let G be an interval graph with a canonical ordering $<*$. G is an indifference graph if and only if for every v_i ($i \in \{1, \ldots, n\}$) of G, $deg(v_i)$ =Last[i]−First[i].*

Proof. First, let G be an indifference graph. Proceed by induction on the number of vertices in G. If G is disconnected, then the conclusion follows by the induction hypothesis applied to every component of G separately.

Therefore, G is assumed to be connected. By corollary (1.14.2), without loss of generality, it is assumed that $<*$ satisfies (1.10); it follows, in particular that every vertex v_i with $i \in \{2, \ldots, n-1\}$ is adjacent to all the vertices $v_{First[i]}$, $v_{First[i]+1}$, $\ldots$, v_{i-1}, v_{i+1}, $\ldots$, $v_{Last[i]-1}$, $v_{Last[i]}$, and so $deg(v_i)$ = Last[i]−First[i].

Further, if $i = 1$, then First[v_i]=1 and v_i is adjacent to v_{i+1}, $\ldots$, $v_{Last[i]}$, confirming that $deg(v_i)$ = Last[i]−First[i]. Finally, if $i = n$, then Last[v_i]=n, and v_i is adjacent to $v_{First[i]}$, $\ldots$, $v_{Last[i]-1}$, and so $deg(v_i)$ = Last[i]−First[i].

Conversely, assume that G contains an induced claw with vertices a, b, c, d and edges ab, ac, and ad. Symmetry allows to assume that $b<*c<*d$. But, in case $c<*a$, $deg(b) < Last\,[b] - First\,[b]$ since b is not adjacent to c and $First\,[d]<*c$. ■

Theorem 1.16. *[35] Let G be an indifference graph with a canonical ordering $\lessdot$, and let x, y ($x \lessdot y$) be distinct vertices of G. If $x = v_1$, v_2, $\ldots$, $v_p = y$ is an arbitrary chordless path joining x and y, then for all $i \in \{1, \ldots, p-1\}$, $v_i \lessdot v_{i+1}$.*

Proof. It is essential to note that

$$v_j \lessdot y \quad for\ all\ j = 1, 2, \ldots, p-1. \tag{1.12}$$

If the statement is is false, then we find a subscript $j (j < p)$ such that

$$v_{j+1} \lessdot v_j.$$

Now (1.12) implies that $v_{j+1} \lessdot v_j \lessdot y$. Observe that

$$v_j \lessdot v_{p-1},$$

for otherwise condition (1.10) guarantees that v_j and v_p are adjacent, a contradiction. Let k be the first subgraph larger than $j+1$ for which

$$v_j \lessdot v_k.$$

Since $v_{j+1} \lessdot v_j$, it must be the case that $k \geq j+2$. By choice of k, $v_{k-1} \lessdot v_j \lessdot v_k$. However, now (1.10) implies that v_j and v_k are adjacent, a contradiction. ∎

Corollary 1.16.1. *[35] Let G be a connected indifference graph with a canonical ordering $\lessdot$. For all subscripts i ($i \in \{2, \dots, n-1\}$), First[i]<i<Last[i].*

Proof. Let i be an arbitrary subscript with $i \in \{2, \dots, n-1\}$. Since G is connected, there exists a path in G joining v_1 and v_i. Let $v_1 = x_1, x_2, \dots, x_t = v_i$, and so First$[i]$<$i$. ∎

Chapter 2

Greedy recognition algorithm

Since the dawn of time, humankind has implicitly used algorithms to carry out everyday tasks ranging from lighting a fire to complex AI programs exploring outer space. In fact, an algorithm is an assortment of instructions using methods ranging from intuition to mechanical devices to computers in order to change or develop a situation or solve a problem. In short, it's an output from a set of inputs in a finite number of steps [6], a set of elementary operations manipulating the inputs called instructions. This sequence of instructions needs to be translated to a programming language to be executed by a computer; Sedgewick and Wayne [52] define an algorithm as being a finite, deterministic, and effective problem-solving method suitable for implementation as a computer program.

The process of producing an algorithm is called computation. The computer's rapid adoption by society through the last decades, has brought computation into every aspect of modern life, and made computational problems important in various disciplines such as engineering, scientific discovery, astronomy, and biology. Algorithms, solvers of computational issues, have become omnipresent.

The first historic traces of algorithms go back to the Babylonian civilization, in Mesopotamia, around 2000 BC, in contexts such as documenting accounts and astronomy. Then come algorithms attributed to Euclid, Al Khawarizmi, princess Didon and many more. Over the years there has been various, to say the least, algorithms for different problems attempting to solve them efficiently through possible approaches. For little problems, it doesn't matter much which approach is adopted, as long as the algorithm correctly solves the problem. Whereas for bigger problems, the choice of approach matters immensely as the efficient use of time and space is to be considered.

Careful algorithm design is a crucial element of the process of solving problems with big data inputs, whatever the applications area [52]. Thus, the choice of the best algorithm for a particular task can be a complicated assignment, involving sophisticated mathematical analysis. Therefore a branch of computer science is dedicated to the study of algorithms and their efficiency. The analysis of algorithms and computational complexity are discussed further in this chapter.

A greedy algorithm builds up a solution by cutting it down to small-scale solutions. It goes through several phases resolving step-by-step to construct the needed solution. In other words, it explores local solutions to build a global solution. Finding the adequate solving approach can be complicated at times and time-consuming. Nevertheless, greedy algorithms are simple and efficient as they are easy to code.

In this chapter, a brief descriptive of computational complexity and its importance are presented within its first section. More importantly, it presents greedy recognition algorithms to recognize chordal graphs, interval graphs and indifference graphs.

2.1 Computational complexity

"If our brains were simple enough for us to understand them, we would be so simple that we could not."
–Ian Stewart

Algorithms tend to differ in the efficient usage of resources as some computational tasks happen to admit nonintuitive algorithms that are more efficient than algorithms used for long stretches of time. This brings to think of how to be sure that an algorithm is the best in performance to solve a given problem, or even, proof of the existence of an efficient algorithm, or better, proof of the nonexistence of such an algorithm. It is for this purpose that the theory of computational complexity thrives to achieve a capital break-through to bring some light upon possibility.

The two major classes of problems that resume the essence of computational complexity are the P class and the NP class. At the present time, there exists a simple mathematical model that suffices for studying many questions about computation and its efficiency: the Turing machine. The latter is able to simulate all physically realizable computational methods with little loss of efficiency [6]. The class P consists of all decision problems that are solvable by Turing machines in polynomial time. Thus, problems within this class are efficiently solvable as their polynomial time algorithms exist. Whereas the class NP contains one of two categories of intractable problems: the non deterministic intractable problems. The latter type of problems are discussed further in the computational intractability section.

2.1.1 Big-Oh notation

The big-Oh notation provides an abstract measurement by which the performance of algorithms can be judged. The computational efficiency of an algorithm is measured as the number of basic operations it performs as a function of its input length. This implies that the performance of an algorithm can be seized by a function T from the set $\mathbb{N}$ of natural numbers to itself such that $T(n)$ is equal to the maximum number of elementary operations that the algorithm performs on inputs of n.

Definition 2.1 (Big-Oh notation). *[6] If f, g are two functions from $\mathbb{N}$ to $\mathbb{N}$, then we say that $f = O(g)$ if there exists a constant c such that $f(n) \leq c \cdot g(n)$ for every sufficiently large n. To emphasize the input parameter, it is best to write $f(n) = O(g(n))$ instead of $f = O(g)$.*

Example 9 Here are two examples of the use of the big-Oh notation:

1. If $f(n) = 99n^2 + 9n + 9log(n)$ and $g(n) = n^2$ then $f = O(g)$. This relation is often written as $f(n) = O(n^2)$.
2. If $f(n) = min\{n, 10^6\}$ and $g(n) = 1$, for every n, then $f = O(g)$. This relation is also written as $f = O(1)$.

As seen above, computational efficiency is quantified by studying how the number of basic operations of an algorithm scales as the size of the input increases, table (2.1) demonstrates this for some polynomial and exponential time complexity functions. Figure (2.1) displays some of the algorithmic run times; it also highlights how influential it is to choose an efficient algorithm.

Time complexity	Size n				
	10	20	30	40	50
n	0.00001s	0.00002s	0.00003s	0.00004s	0.00005s
n^2	0.0001s	0.0004s	0.0009s	0.00016s	0.00025s
n^3	0.001s	0.008s	0.027s	0.064s	0.125s
2^n	0.001s	1.0s	17.9m	12.7d	35.7y
3^n	0.059s	58m	6.5y	3855c	2×10^8c

Table 2.1: Comparison of several polynomial and exponential time complexity functions [29].

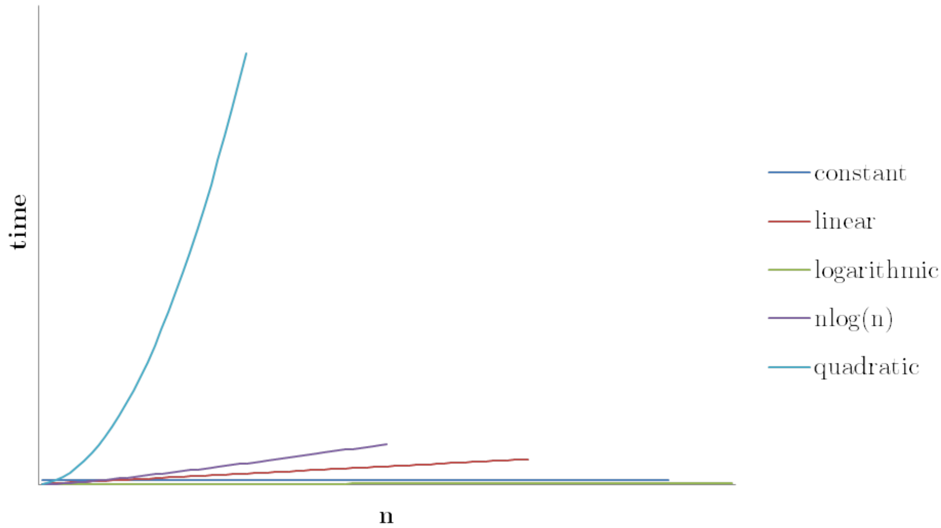


Figure 2.1: Algorithmic run time [8].

The greatest asset that the big Oh notation gives is that it is significant regardless of the type of hardware that is used. In fact, this notation isolates growth that is a key factor of algorithm analysis. For instance, an algorithm with a quadratic run time grows faster than one with a logarithmic run time; however, at some point as $n \to \infty$, the logarithmic algorithm becomes faster than the quadratic algorithm. So, if the first algorithm was inserted in the fastest existing machine and the latter in the slowest machine then the logarithmic algorithm will present results quicker than the quadratic algorithm that might not present any in a near future [8]. Actually, Arora and Barak [6] maintain that the efficiency of an algorithm is to a considerable extent much more important than the technology used to execute it.

2.1.2 Computational intractability

A problem is considered to be intractable if it is so hard that no polynomial time algorithm can possibly solve it. This mainly crosses out all the efficiently solvable problems with polynomial time algorithms. However, there are a few exponential time algorithms that have proved themselves to be quite useful in practice. For instance, the simplex algorithm for linear programming (proposed by Dantzig in 1947) has an exponential run-time and yet it possesses an impressive record of running fast in practice. Another example to mention is the branch and bound algorithm, for the knapsack problem, which also is an exponential time algorithm that has demonstrated great use in practice. Unfortunately, there are many exponential time algorithms and few of them only are regarded as useful in practice [29].

There are two main reasons of intractability:

1. the problem is so difficult that a solution takes an exponential amount of time to be discovered;
2. the problem requires a solution that is so extensive that it cannot be described by means of the input length that are bounded by a polynomial function.

The latter type of intractability indicates, in most cases, that the problem is not defined realistically as more information is required than there could ever be. Nevertheless, it is important to be able to recognize this type. Turing was the first to demonstrate that certain problems are so hard that they deny solutions such that no algorithm offering some solving approach exists whatsoever [29].

> **Example 10** [Turing, 1936] Turing proved that it is impossible to specify any algorithm which given an arbitrary computer program and an arbitrary input to that program, can decide whether or not the program will eventually halt when applied to that input (this example resembles John Conway's Game of Life).

The first type of intractability is called non deterministic, following the non deterministic computer models that attempted to solve intractable problems. Such computer models have the ability to pursue a number of independent computational tasks in parallel. The NP class gathers problems which as opposed to the undecidable problems, admit presented solutions that are at most polynomial in the length of the input to be efficiently verified. This means that if a solution is presented, the task of verifying, that such a solution solves the problem, has a polynomial run-time. Problems of this class that are nondeterministically intractable are called NP-complete problems belonging to the NP-hard class (a subclass of the NP class).

The problems in the class P are efficiently solvable which makes their solutions also efficiently verifiable. This implies that the class P is within the class NP. At the present time, it is still unknown if the class P is equal to the class NP. In fact, the implications of such an equality are bound to be revolutionary as it is of great practical, scientific, and philosophical interest [6].

Building a solid proof of inherent intractability is as hard as finding an efficient algorithm. Nevertheless, the theory of NP-completeness provides many unequivocal techniques for proving that a given problem is "just as hard" as a large number of other problems. One of the techniques for proving that a problem is intractable would be by reducing it to another intractable problem (that has been proved to be within the NP class). The foundations for the theory of NP-completeness were presented by Stephen Cook in 1971 where he emphasizes on the significance of polynomial time reducibility of decision problems. According to Cook's paper, it's sufficient to reduce a problem to a known NP-complete problem to affirm its intractability, provided that the transformation is polynomial. He also proved a particular NP-hard problem called the satisfiability problem. It is the first problem to be contained in the NP-hard class and has the property that every other NP problem can be reduced to it in polynomial time. This has a rather important implication as it means that if the satisfiability problem can be solved by a polynomial time algorithm, then so can every problem in the NP-hard class.

Discovering that a problem is NP-complete is the beginning of work on the problem. This knowledge provides valuable clues about what lines of approach are available to consider, as they have the potential of being most productive. For instance, the search for an efficient algorithm should be accorded little importance if not dropped and settle for less enthusiastic approaches:

- search for efficient algorithms that solve an instance of the general problem (heuristics);
- relax the problem, find a quick algorithm that builds a special designed solution for a variation of the general problem (meta-heuristics).

In summary, the essential application of the theory of NP-completeness is to assist directing problem-solving efforts towards approaches that have the greatest potential of leading to convenient efficient algorithms.

The theory of complexity is roughly 70 years old; it's yet a growing discipline. In fact many important results are less than 40 years of age. David Hilbert sums it up in an elegant way in his lecture (Hilbert, 1900), "Proofs of impossibility were effected by the ancients [...and] in later mathematics, the question as to the impossibility of certain solutions plays a preeminent part. [...] every definite mathematical problem must necessarily be susceptible to an exact settlement, either in the form of an actual answer to the question asked, or by the proof of the impossibility of its solution and therewith the necessary failure of all attempts. This conviction [...] is a powerful incentive to the worker. We hear within us the perpetual call: *There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no ignorance.*"

2.2 Chordal graph recognition algorithm

As indicated previously, one of the ways chordal graphs are characterized is by using perfect elimination orderings. In fact, an undirected graph is chordal if and only if it has a perfect elimination order [21, 47]. Considering the fact that a chordal graph G, which is not a clique, has two nonadjacent simplicial vertices (lemma 1.2.1), and every induced subgraph of a chordal graph is chordal, it is feasible to choose any vertex v of G and decide it will be last in the perfect elimination order; then choose any vertex w adjacent to v and put it in position $n-1$, and so on, choosing vertices in a backwards fashion to the perfect elimination order.

The precedent operation could be performed by a maximum cardinality search described by Tarjan and Yannakakis in 1984.

Theorem 2.1. *[21] An undirected graph G is chordal if and only if maximum cardinality search (G, σ) produces a perfect elimination order σ.*

Proof. Suppose G is chordal. Proceed by induction on the number of vertices to show the existence of PEO of G. The case $n = 1$ is trivial. Suppose $n > 1$ and every chordal graph with fewer vertices has a PEO. By Lemma 1.2.1, G has a simplicial vertex, say v. Now $G\ \{v\}$ is a chordal graph with fewer vertices than G; hence, by induction it has a PEO, say σ_1. If σ_2 orders the vertex v first, followed by the remaining vertices of G in the order determined by σ_1, then σ_2 is a PEO of G.

Conversely, suppose G has a PEO, say σ_1, given by $\{v_1, v_2, \ldots, v_n\}$. Seek a chord of an arbitrary cycle C in G of length greater than three. Let v, be the vertex on C whose label i is smaller than that of any other vertex on C. Since σ_1 is a PEO, $adj(v_i)$ is complete; whence C has at least one chord: namely, the edge joining the two neighboring vertices of v_i in C. ∎

In light of the previous facts, the greedy chordal graph recognition algorithm is composed of two major steps.

1. Maximum cardinality search [55]: It numbers the vertices from n to 1 in decreasing order. As the next vertex to number, it selects the vertex adjacent to the largest number of previously numbered vertices, breaking ties arbitrarily.

2. Perfect elimination order test [46]: It tests whether or not the obtained order σ from the maximum cardinality search is a perfect elimination order.

Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph. The following greedy algorithm establishes whether or not G is a chordal graph.

Algorithm 1 Chordal graph recognition algorithm

Input: $G = (V, E)$ an undirected graph

```
function CGR(G)
    MCS(G,σ)                    ▷ Calls Maximum Cardinality Search Algorithm
    if PEO(G,σ) then            ▷ Performs Perfect Elimination Order test
        return "True";
    else
        return "False";
    end if
end function
```

Output: "True" or "False" depending on whether or not G is a chordal graph

---

The complexity of the chordal graph recognition algorithm is $O(n+m)$ (see [22]). The following algorithm generates an ordering σ of the vertices of G according to the maximum cardinality search.

Algorithm 2 Maximum Cardinality Search

Input: $G = (V, E)$ an undirected graph

```
procedure MCS(G,σ)
    for all vertices v of G do
        set label[v] to zero;
    end for
    for all i from n downto 1 do
        choose an unnumbered vertex v with the largest label;
        set σ(v) to i ;
        for all unnumbered vertices w adjacent to vertex v do;
            increment label[v] by one ;
        end for
    end for
end procedure
```

Output: An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of the vertices of G

Theorem 2.2. *[22] The maximum cardinality search algorithm can be implemented in $O(n+m)$ time.*

Proof. Let vertices be labeled by the number of adjacent numbered vertices, and let T_i be the set of unnumbered vertices with the label i. Represent each set of S_i by a doubly linked list of vertices, to facilitate deletion, and maintain an array of sets of vertices S_i, for $0 \leq i \leq m-1$. Maintain also an index j to the nonempty set S_j with the largest label, and maintain for each vertex a pointer to the position of the vertex in the corresponding set. Maximum cardinality search proceeds by removing some vertex v from the set S_j, numbering vertex v and, for all unnumbered vertices w from the set containing it, say S_i, to the set S_{i+1}. This takes $O(1 + deg(v))$ time which means it takes the maximum cardinality search algorithm $O(n+m)$ time. ■

The following algorithm verifies whether or not the generated ordering σ of the vertices of G by the maximum cardinality search algorithm is a perfect elimination order.

Algorithm 3 Perfect Elimination Order

Input: $G = (V, E)$ an undirected graph

```
function PEO(G,σ)
return "True";
    for all i from 1 to n − 1 do
        set v to i;
        set m[v] to σ^{-1} {σ(w)|w ∈ adj(v), σ(w) > σ(v)};
        for all vertices w adjacent to vertex v do
            if σ(m[v]) < σ(w) and w ∉ adj(m[v])) then
                return "False";
            end if
        end for
    end for
end function
```

Output: An ordering $(v_1, v_2, \ldots, v_n)$ of the vertices of G

The perfect elimination order test algorithm can be implemented in $O(n + m)$ time [47, 22].

2.2.1 Generation of maximal cliques

A maximal clique or a maximal complete subgraph is a set of vertices that are pairwise adjacent. This clique cannot be enlarged by adding one more adjacent vertex; this implies that it doesn't exist exclusively within the vertex set of a larger clique.

It is interesting to note that chordal graphs make it easy to generate maximal cliques while it is an NP hard problem within arbitrary graphs. Let $\sigma = (v_1, v_2, \ldots, v_n)$ be the perfect elimination order produced by MCS(G,σ), algorithm (4) enumerates all maximal cliques of G according to the following theorem.

Theorem 2.3. *[21] Let $X(v) = \{w \in adj(v) | \sigma(v) < \sigma(w)\}$ be the set of neighbors of vertex v such as $\sigma(v) < \sigma(w)$. Every maximal clique of a chordal graph $G = (V, E)$ is of the form $\{v\} \bigcup X(v)$, for any vertex $v \in V$.*

Proof. The set $\{v\} \bigcup X(v)$ is complete, because σ is a perfect elimination order. Also, if w is the vertex of a maximal clique C with the smallest σ number, then C is of the form $\{w\} \bigcup X(w)$. ■

Lemma 2.3.1. *[21] A chordal graph has at most n ($n = |V|$) maximal cliques.*

Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph along with $\sigma = (v_1, v_2, \ldots, v_n)$ the perfect elimination order produced by MCS(G,σ). The following greedy algorithm generates all the maximal cliques of G (see [30]).

Algorithm 4 All Maximal Cliques

Input: $G = (V, E)$ an undirected graph

```
procedure AllMax(G,σ)
    for all vertices v of G do
        set size[v] to zero;
    end for
    for all i from 1 to n do
        set v to i;
        if deg(v) = 0 then
            output maximal clique {v};
        end if
        set X to {w ∈ adj(v)|σ(v) < σ(w)};
        if X ≠ 0 then
            if size[v] < |X| then
                output maximal clique {v} ⋃ X;
            end if
            set m[v] to σ^{-1}(min{σ(w)|w ∈ X});
            size[m[v]] to max{size[m[v]], |X| − 1};
        end if
    end for
end procedure
```

Output: An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of the vertices of G

The generation of all maximal cliques in a chordal graph takes $O(n + m)$ (see [22]).

Example 11

Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph. The maximum cardinality search sweeps through the graph and generates an ordering σ of the vertices of G as shown in figure (2.2).

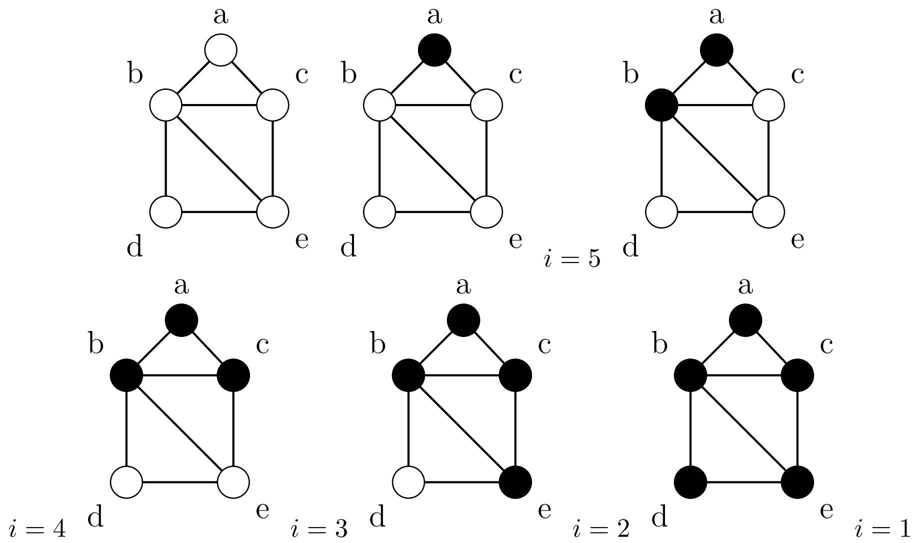


Figure 2.2: Maximum Cardinality Search sweep
on graph $G = (V, E)$

The Maximum Cardinality Search procedure generates $\sigma = (a, b, c, e, d)$.
The All Maximal Cliques procedure generates the following cliques:
$C_1 = \{c, a, b\}$;
$C_2 = \{d, b, e\}$;
$C_3 = \{e, b, c\}$.

2.3 Interval graph recognition algorithm

Interval graphs, as indicated previously, admit a characterization by a specific ordering of their vertices (see theorem 1.9). Thus, the recognition approach for interval graphs comprises two steps [23].

1. Generate an ordering σ of the vertices of G.

2. Test if the ordering in step 1 is obtained according to theorem (1.9).

Throughout the years, there has been different ordering characterizations for interval graphs; consequently, some have used the ordering of the maximal cliques of the graph (see theorem 1.7) instead of the ordering of vertices according to theorem (1.9). The following table summarizes the different approaches used to recognize interval graphs.

Authors	Year	Description	Complexity
Lekkerkerker and Boland [33]	1962	Asteroidal triplets and simplicial vertices	$O(n^4)$
Fulkerson and Gross [21]	1965	Consecutive 1's property	$O(n^4)$
Booth and Lueker [14]	1976	PQ-trees	$O(n+m)$
Korte and Möhring [40]	1989	MPQ-trees and Lexicographic Breadth First Search	$O(n+m)$
Hsu [28]	1993	Modular decomposition	$O(n+m)$
Corneil, Olariu, and Stewart[16]	1998	Four successive sweeps of LBFS	$O(n+m)$
Habib, Paul, McConnell, and Viennot [26]	2000	Lexicographic breadth first search and partition refinement on maximal cliques	$O(n+m)$
Corneil, Olariu, and Stewart[17]	2009	Six successive sweeps of LBFS	$O(n+m)$

Table 2.2: The different approaches to recognize interval graphs through time

Since interval graphs are characterized by a special ordering of their vertices according to (1.7), a polynomial time vertex ordering algorithm can be used to recognize this class of graphs called lexicographic breadth first search (LBFS) algorithm. The LBFS is a restricted version of the breadth first search (BFS) algorithm in a manner that the orderings of LBFS are the subset of BFS orderings.

A BFS ordering is an assignment of order such that starting from an arbitrary selected vertex, explore neighbor vertices first in a given direction (righ or left) before moving to explore next level of neighbors. An LBFS is a BFS in which preference is given to vertices whose neighbors have been visited earliest in the search [56].

Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph. The following greedy algorithm determines whether or not G is an interval graph. It was inspired by the algorithm proposed by Corneil, Olariu, and Stewart [16].

Algorithm 5 Interval graph recognition algorithm

Input: $G = (V, E)$ an undirected graph

```
function IGR(G)
LBFS(G, σ, x)                          ▷ Generates ordering σ
LBFS+(G, σ, σ+)                        ▷ Generates ordering σ+
INDEX(G, σ+, i+)                       ▷ Generates vertices' index i+
SET(G, σ+, i+,A)                       ▷ Generates the sets of vertices A
LBFS+(G, σ+, σ++)                      ▷ Generates ordering σ++
INDEX(G, σ++, i++)                     ▷ Generates vertices' index i++
SET(G, σ++, i++,B)                     ▷ Generates the sets of vertices B
LBFS*(G, σ+, σ++, σ*)                  ▷ Generates ordering σ*

    if umbrella(G, σ*)=true then       ▷ Test if an umbrella exists within the ordering σ*
        return "False";
    else
        return "True";
    end if
end function
```

Output: "True" along with a canonical ordering $\sigma*$ or "False" depending on whether or not G is an interval graph

The complexity of the greedy interval graph recognition algorithm is $O(n + m)$ [16]. Moreover, proof of the correctness of this algorithm can also be found in [16]. Algorithm (6) describes the LBFS(G, σ, x) procedure that is the base procedure to LBFS+(G, σ, $\sigma+$) and LBFS∗(G, $\sigma+$, $\sigma++$, $\sigma*$) except for a few differences that are mentioned in table (2.3). Example (12) illustrates how the algorithm operates on a graph.

The procedures INDEX(G, $\sigma+$, $i+$) and SET(G, $\sigma+$, $i+$, A) are called to help choose the suitable vertex v to be considered in each iteration of the LBFS procedure described in algorithm (6). The procedure INDEX(G, $\sigma+$, $i+$) generates the index of each vertex of G according to $\sigma+$ such as it denotes the largest $\sigma+(w)$ ($w \in V$ and $w \in adj(v)$). On the other hand, the procedure SET(G, $\sigma+$, $i+$, A) generates a set A such as $A(v)$ is the set of neighbors of v that are before v in $\sigma+$ and are adjacent to a vertex after v in $\sigma+$.

The function umbrella(G, $\sigma*$) verifies whether or not the ordering $\sigma*$ obtained satisfies (1.7).

Algorithm 6 Lexicographic Breadth First Search

Input: $G = (V, E)$ an undirected graph, an ordering σ of the vertices of G, and a distinguished vertex x of G

```
function LBFS(G, σ, x)
    set label(x) to n                                         ▷ n = |V|
    for all vertices v in V − {u} do
        set label(x) to 0;
    end for
    for all i from 1 to n − 1 do
        pick an unnumbered vertex v lexicographically the largest label;      (⋆)
        set σ (v)(x) to |V| + 1 − i;
        for each unnumbered vertex w in adj(V) do
            set label(w) to label(w)+i;
        end for
    end for
end function
```

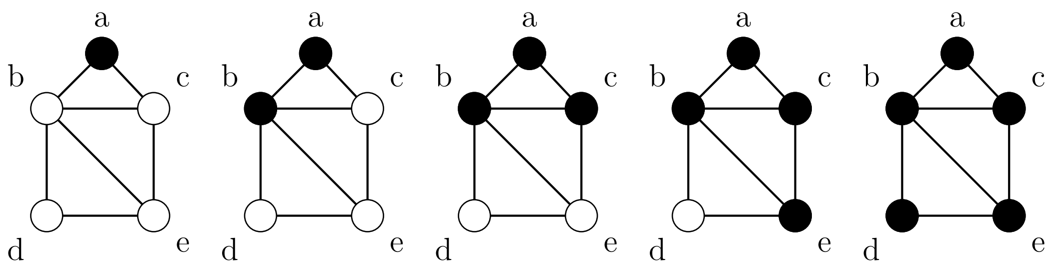
Output: An ordering σ of the vertices of G

	LBFS	LBFS+	LBFS*
Output	G and x	G and σ	G, $\sigma+$, and $\sigma{++}$
Choice of vertex x	Random choice	x is the last vertex in the ordering σ	x is the last vertex in the ordering $\sigma+$
Choice of vertex v	Random choice	v is the vertex appearing last in the ordering σ	v is the vertex chosen between a and b following the steps described in [16]*
Output	σ	$\sigma+$	$\sigma*$

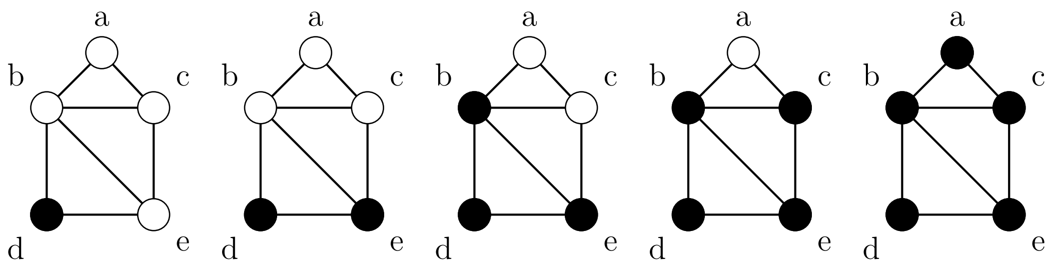
*Let a and b respectively be the last vertex in $\sigma+$ and $\sigma*$

Table 2.3: The variation of procedure LBFS to fit LBFS+ and LBFS*.

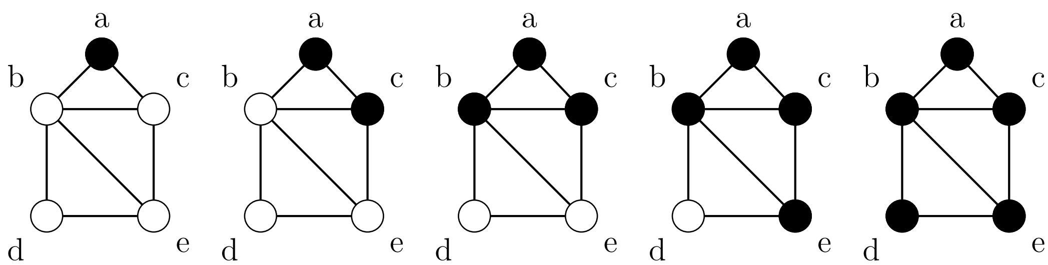
Example 12 Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph. The four LBFS sweeps through the graph generate the orderings σ, $\sigma+$, $\sigma++$, $\sigma*$ of the vertices of G (figure 2.3).



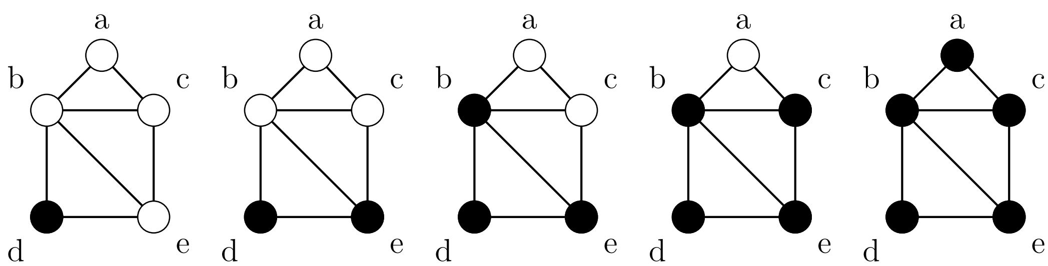
LBFS, $\sigma = (a, b, c, e, d)$



LBFS+, $\sigma+ = (d, e, b, c, a)$



LBFS++, $\sigma++ = (a, c, b, e, d)$



LBFS*, $\sigma* = (d, e, b, c, a)$

Figure 2.3: The four LBFS sweeps on graph $G = (V, E)$.

2.4 Indifference graph recognition algorithm

Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph. The following greedy algorithm determines whether G is an indifference graph.

Algorithm 7 Indifference graph recognition

Input: $G = (V, E)$ with an ordering $\sigma*$ of its vertices

```
  function TEST(G)
    if IGR(G)=true then

      for i ← 1 to n do
        determine First[i] and Last[i];
      end for
      for i ← 1 to n do
        if Last[i] − First[i] then return "False";
        end if
      end for
      return "True";
    else
      return "False";
  end if
end function
```

Output: "True" or "False" depending on whether or not G is an indifference graph

Theorem 2.4. *[35] The procedure* TEST(G) *is implemented in* $O(n + m)$ *time to decide whether* G *is an indifference graph.*

Proof. To update the adjacency structure of G according to $\sigma*$ takes $O(n+m)$ as it informs every vertex of its position in the canonical ordering. Whereas, computing First$[i]$ and Last$[i]$ takes $O(deg(v_i))$ for each vertex v_i as it scans the adjacency list of of v_i once retaining the largest and smallest subscript. ∎

Example 13 Let $G = (V, E)$ with $|V| = n$ and $|E| = m$ be an arbitrary finite, undirected, simple, and loopless graph (figure 2.4).

STAGE 1. Invoke the interval graph recognition algorithm (algorithm 5) of Corneil, Olariu, and Stewart [16]. If G is an interval graph, the algorithm also returns an ordering of the vertices of G satisfying (1.7) of theorem (1.9).
Algorithm (5) returns $\sigma* = (d, e, b, c, a)$ as seen in example (12) since G is an interval graph.

STAGE 2. Compute First$[i]$ and Last$[i]$ for every i ($i \in \{1, \ldots, n\}$), corresponding to every vertex v of G, by examining the adjacency matrix of G.
According to algorithm (7), the following is obtained for the graph G in figure (2.4).

$$\text{First}[i]\text{=[3,1,2,1,1]}$$
$$\text{Last}[i] \text{ =[5,5,5,3,4]}$$

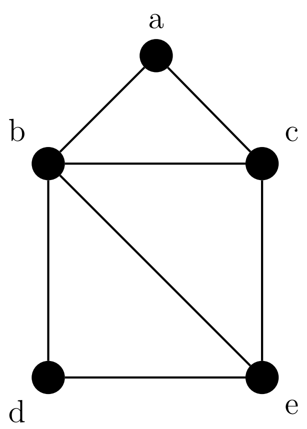
STAGE 3. For every i ($i \in \{1, \ldots, n\}$), check whether $deg(v_i)$ =Last[i]−First[i].
(a) Compute $deg(v_i)$ for every i ($i \in \{1, \ldots, n\}$):

$$deg(v_i)\text{=[2,4,3,2,3]}$$

(b) Compute Last[i]−First[i] for every i ($i \in \{1, \ldots, n\}$):

$$\text{Last}[i]-\text{First}[i]\text{=[2,4,3,2,3]}$$

According to the three steps followed by algorithm (7), the graph G is an indifference graph.



$$\begin{array}{c c} & \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Figure 2.4: A graph $G = (V, E)$ and its adjacency matrix.

Chapter 3

Computational problems

Computational problems, as seen in previous chapters, arise from real-life situations and are mostly of practical nature. Moreover, indifference graphs are characterized by the existence of a special ordering of their vertices which leads to optimal greedy algorithms for a number of computational problems. There are four main types of computational problems as follows [57].

- Decision problems: acknowledge the existence of a solution, such problems are called binary as they require a "yes" or "no" answer;
- Search problems: acknowledge that the existence of a solution is not enough, it is required to be known;
- Optimization problems: acknowledge that a solution is not enough, it is required to be the best; and
- Counting problems: require the number of solutions of a given instance.

In this chapter, a couple of computational problems are presented accompanied with simple greedy algorithms particularly the graph coloring problem and the maximum matching problem. These problems are known to be NP-complete; however, for a particular instance, in the case of indifference graphs, they accept greedy algorithms to solve them.

3.1 Graph coloring

The graph coloring problem arise in a wide variety of applications amongst which the desire to give different colors to neighbor countries represented on a map is found. A graph $G = (V, E)$ is associated to a group of countries to color in order to proceed to a solving approach (figure 3.1). Graph coloring translates to coloring the vertices of G, where each vertex receives exactly one color, in such a way that if two vertices are joined by an edge, they get different colors (see figure 3.2). If such a vertex coloring can be executed using k colors, it is then said that G is k-colorable. The smallest possible k to exist for a given graph G is called the chromatic number of G [11, 45].

There are two variants of the graph coloring problem: the first investigates whether or not a proper $k - coloring$ $(k > 0)$ of G exists (a decision problem) while the second attempts to determine the chromatic number of G, $\chi(G)$ (an optimization problem). In this section, it's the latter that is studied. It is obvious that for a complete graph K_n, $\chi(K_n) = n$ where each vertex receives a different color as all vertices are pairwise adjacent. Furthermore for a cycle C_n, where n is even (alternative colors around the circuit), $\chi(C_n) = 2$; whereas when n is odd, $\chi(C_n) = 3$ [45].

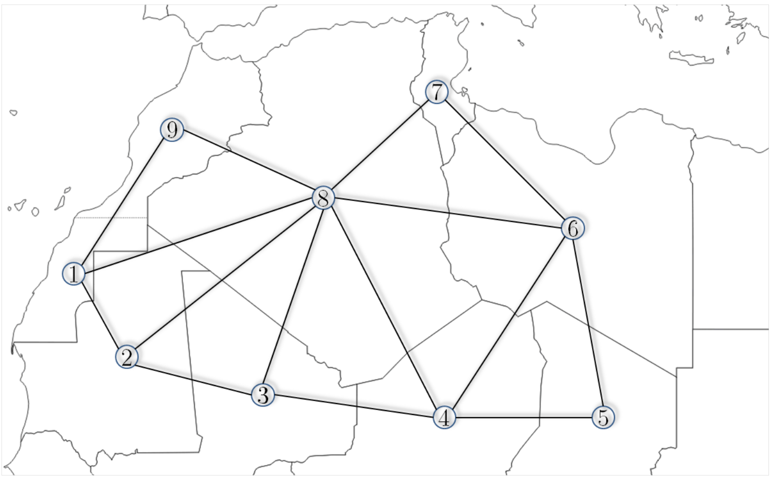
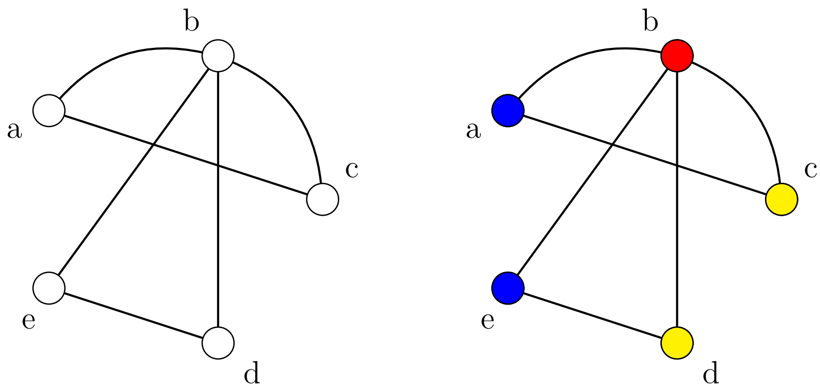


Figure 3.1: A graph associated to the map of some countries of Africa.



$\chi(G) = 3$

Figure 3.2: Graph coloring of a graph G.

The following algorithm (8) presents a simple, greedy approach to solve the graph coloring problem for indifference graphs. The specific linear order of the vertices of such graphs is exploited for the design of the optimal greedy algorithm. For this, the data structure used is a stack which contains $1, 2, \ldots, n$ colors in reverse order. The algorithm initiates by assigning color 1 to vertex v_1, i.e. v_1 is colored by popping the stack.

After v_i $(i \geq 1)$ has been colored, the algorithm proceeds to color v_{i+1}. First, it considers the set of colors assigned to the vertices

$$v_{First[i-1]},\ v_{First[i]+1},\ \ldots,\ v_{First[i]-1}.$$

It is essential to note that by corollary (1.14.2), all these colors must be distinct. In addition, none of these vertices are adjacent to v_{i+1}, and so we can reuse any of their colors on v_{i+1}. Next, it releases these colors by pushing them onto the stack and then proceed to color v_{i+1} by simply popping the stack.

Algorithm 8 Graph coloring

Input: $G = (V, E)$ an indifference graph

```
1: procedure COLOR(G)
2:     color(v_1) ← pop(stack);
3:     for i ← 2 to n do
4:         for j ← First[i − 1] to First[i] do
5:             push color(v_j);                    ▷ skips to the next color
6:         end for
7:         color(v_i) ← pop(stack);
8:     end for
9: end procedure
```

Output: an optimum coloring of G

Theorem 3.1. *[35] Let $G = (V, E)$ be an indifference graph with n vertices and m edges.* COLOR(G) *returns an optimal coloring of G in $O(n)$ time.*

Proof. It is important to show that the procedure COLOR(G) returns a proper coloring and that this coloring uses as few colors as possible in order to justify the correctness of the algorithm. First, the following assertion is essential.

$$\begin{gathered}\text{no vertex } v_j \text{ is adjacent to one of the vertices}\\ v_k \text{ such that First}[i-1] \leq k \leq \text{First}[i]-1,\ i \leq j.\end{gathered} \tag{3.1}$$

Note that if some vertex v_j with $i \leq j$ is adjacent to a vertex v_k with First$[i-1] \leq k \leq$ First$[i]-1$, then by (1.10) it must be that v_i and v_k are adjacent which contradicts $k <$First$[i]$.

Equation (3.1) implies that when a color assignment takes place in line 5 of algorithm (8), vertex v_i and the vertices that have received the same color prior to v_i are not adjacent. Thus, procedure COLOR(G) produces a proper coloring.

To argue about the optimality of this coloring, it is only needed to show that if procedure COLOR(G) uses a total of k colors, then G contains k pairwise adjacent vertices. This implies that no fewer than k colors can possibly be used to to properly color G. Consider the first vertex v_i that received the color k. Observe that the way the stack was initialized, along with line 5, guarantees that all the first $k-1$ colors, i.e. $1, 2, \ldots, k-1$, were in use when v_i was about to be colored. Now, (3.1) implies that these colors must have been used on the vertices v_k with First$[i] \leq k \leq i-1$. However, corollary (1.14.2) guarantees that G contains a set of k pairwise adjacent vertices, namely $v_{First[i]}$, $v_{First[i]}$, $\ldots$, v_i. This shows that the coloring produced is optimal.

As for the complexity, according to equation (3.1), the loop in lines 3-4 takes at most $O(n)$ time. Consequently, the running time of procedure Color(G) is bounded by $O(n)$. ■

The optimal coloring of the graph in figure (3.2) given by the previous algorithm would be as shown in table (3.1).

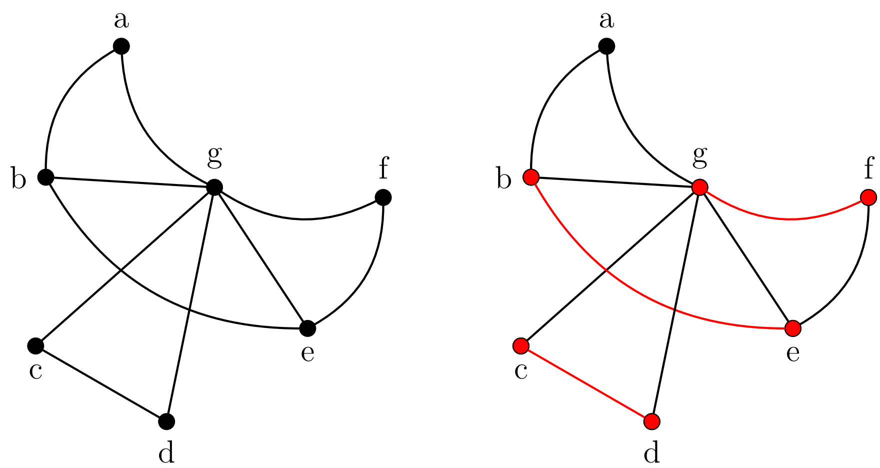
$$\textsc{Color}(G)=\{blue,\ red,\ yellow\}$$

Color	Vertices
Blue	a, e
Red	b
Yellow	c, d

Table 3.1: Optimal graph coloring of the graph G of figure (3.2).

3.2 Maximum matching

A matching of a graph $G = (V, E)$ is a set of independent edges of G, a set of pairwise nonadjacent edges. If $(u, v) \in E$ is an edge of a matching M of G, the incident vertices u and v of the edge are said to be matched by M [7]. A matching is maximum if it is the largest of all maximal matchings (no edge can be added to the set M of a maximal matching). Figure (3.3) shows an example of a maximum matching.



$$M = \{(b, e); (c, d); (f, g)\};\ |M| = 3$$

Figure 3.3: Maximum matching of graph G.

Matching problems have a wide-range of applications amongst which there are scheduling problems, flow problems, and covering problems. To have a better insight on such applications, refer to [36] and [27].

Fortunately, again, the canonical ordering of an indifference graph can be used to obtain a simple algorithm to compute maximum matching. The algorithm proceeds as follows.

Algorithm 9 Maximum matching

Input: $G = (V, E)$ an indifference graph

1: an ordering $<*$ of its vertices

2: **procedure** Max Matching(G,$<*$)

3: $M \leftarrow \varnothing$ ▷ M is a linked list containing edges of G

4: **for** $i \leftarrow 1$ to $n - 1$ **do**

5: **if** v_i and v_{i+1} are adjacent **then**

6: add the edge (v_i, v_{i+1}) to M (providing that v_i nor v_{i+1} is incident to an edge in M);

7: **end if**

8: **end for**

9: **return** M

10: **end procedure**

Output: M a Maximum Matching of G

Theorem 3.2. *[35] Let $G = (V, E)$ be a connected indifference graph with n vertices and m edges with a canonical order $<*$. Procedure* Max Matching (G,$<*$) *returns a maximum matching in G in $O(n)$ time.*

Proof. In order to address the correctness of the procedure, consider a connected component H of G and let

$$v_i < * \, v_{i+1} < * \, \ldots < * \, v_p$$

be the restriction of $<*$ on H. Note that according to corollary (1.14.2), v_j and v_{j+1} are adjacent to all values of $j = i, i+1, \ldots, p-1$. Therefore, the set M_H of edges returned by the procedure is a matching in H. To see that M_H is maximal, note that at most one vertex in H is exposed,which means that it is not incident to an edge in M_H. The conclusion follows from the observation that a maximum matching in G is the union of maximum matching in every component of G.

As for the complexity, once the ordering $<*$ is known, the loop in lines 4-6 runs in $O(n)$ time. ■

Chapter 4

Implementation

This chapter discusses the programming language used to code the program developed in the setting of this work. It also presents a case study that displays the performance of the program with the given results.

4.1 Application software

The Pascal language bears a good programming approach. It's widely used in the academic settings as it helps acquire solid notions that are found in all other programming languages. The latter is a code that the computer can interpret and execute. A program is a set of coding blocks that describes a set of instructions to solve a given problem in a programming language [31]. The application software developed in the context of this work uses Turbo Pascal 7.0 as a programming language.

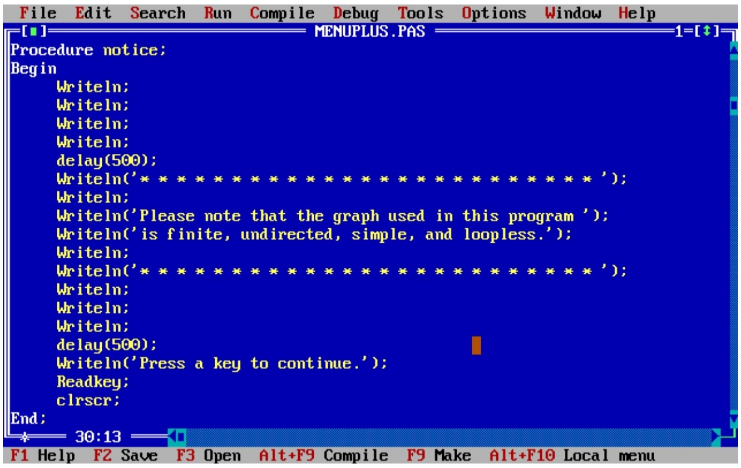


Figure 4.1: Turbo Pascal 7.0 interface.

Turbo Pascal 7.0 has the ability to build Windows-compatible programs and it offers an enhanced variety of compatible built-in units compared to precedent versions.

Algorithms, as previously stated, are series of instructions describing a solving approach of a given problem. Those mentioned in this work constitute the core of the program designed for this project. The algorithms were converted into elementary operations before being translated to the Pascal language. In addition to these core algorithms, supplementary components were added to render the program user-friendly.

The algorithms and additional material composing the program are translated into Pascal language using a collection of elementary operations built by variables of various types: default and user-defined types. These help create data structures adequate to every problem. Data structures are discussed in the next section.

Several units build up the modules of the program providing an easy access and manipulation of code blocks. They fall into two categories: built-in and user-defined units. These units encase variables, types, and procedures that can be accessed by the main program and other units.

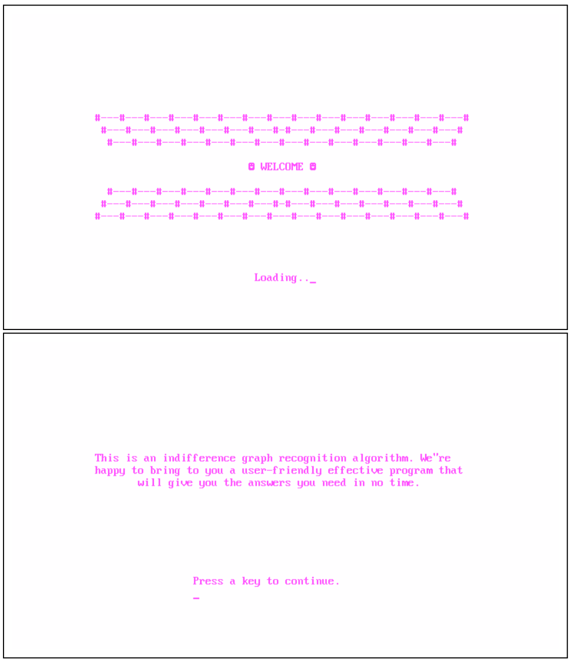


Figure 4.2: Welcome screen.

4.2 Data structures

Problem solving using an automated method requires an abstraction of reality in representing the essentials of the problem. The choice of the representation of data that is to model the real situation is seldom an easy task. This choice is guided by the problem itself as well as the representation tools available by the machine (considering that the automated method is carried by a machine, i.e. a computer). Another essential factor to be considered is the set of operations that are to be performed according to the chosen solving approach [59].

The developed program contains various data structures in agreement with the needs of the different algorithms that are implemented. Most of these data structures use dynamic allocation. The latter is useful in obtaining generic access to several similar variables and disposing a dynamic variable, whether it's an array, matrix, or a value, at any moment of the execution of the program. The table (4.1) illustrates the main data structures and their types.

Component	Type
Adjacency matrix	File of records
Ordering of vertices	Array of integers
Maximal cliques	Linked lists of records linked by records (fig.4.3)
Built sets A & B (LBFS)	Array of records and linked lists (fig.4.4)
Subslice[1] (LBFS)	Linked list of records (fig.4.5)

Table 4.1: Main data structures and their types.

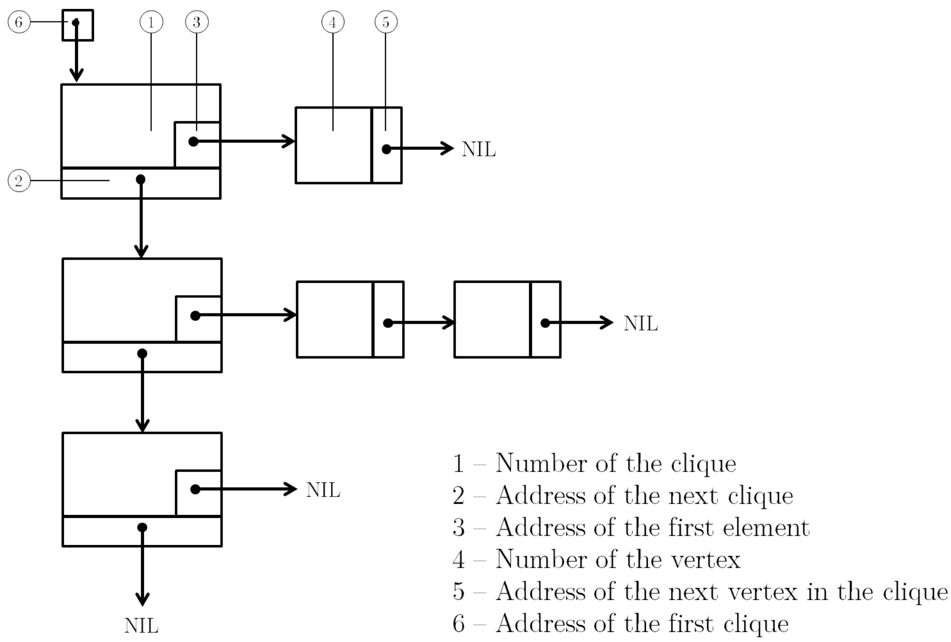


Figure 4.3: Data structure of maximal cliques.

[1]Set of vertices encountered in step ($\star$) of algorithm (6)

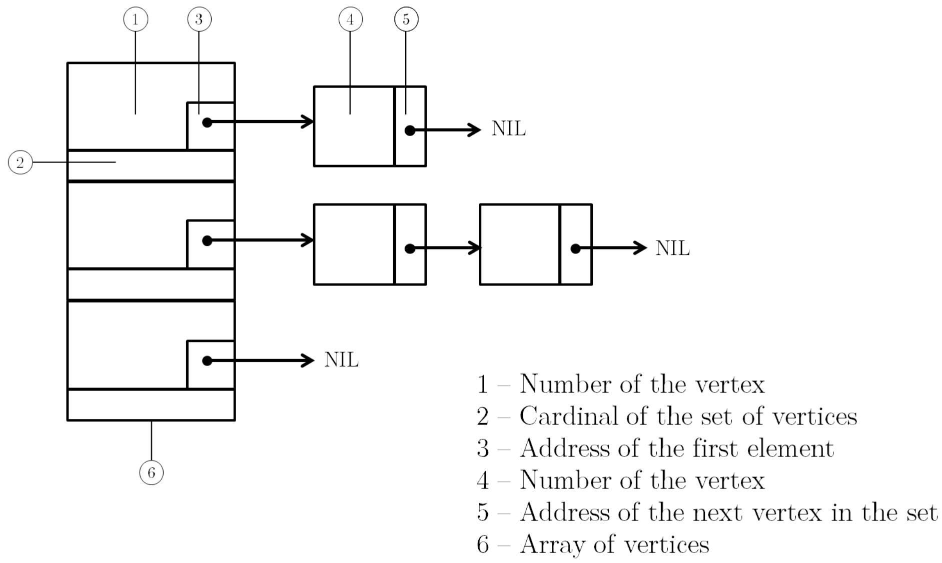


Figure 4.4: Data structure of built sets A & B.

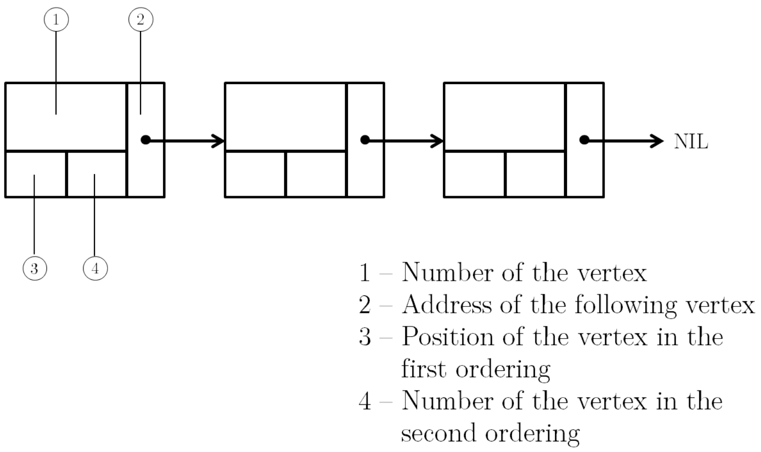


Figure 4.5: Data structure of subslices in LBFS.

4.3 Case study

When companies grow, they tend to evolve into a group of independent departments. Unfortunately, sometimes these departments turn out to accomplish identical assignments as well as recruit employees that possess the same set of skills. Eliminating redundancy between departments is one of the ways to reduce labor costs of a company whether it is through reducing staff or canceling certain tasks.

A company decides to consider the reduction of staff. For this, the CEO thinks of elaborating an inventory of the set of skills that each employee has. Then, he considers all the necessary sets of skills required by their respective departments in order to recognize a pattern of redundancy.

4.3.1 Modeling

To model this situation, let $G = (V, E)$ be an undirected, loopless and simple graph wherein the vertices represent the employees and an edge links two vertices whenever two employees share a set or more of skills. This means that the two employees, sharing a set of skills, are equally qualified for a given task. This example considers a small scale operation involving eight employees that supposedly work in one department. The table (4.2) shows the set of skills of the employees.

Jefri	Natalia	Slimane	Judith
Bilingual Management skills	Marketing skills	Communication skills Management skills	Marketing skills Communication skills Management skills Teamwork
Hayato	Omar	Steven	Nadia
Marketing skills Teamwork	Trilingual Management skills Teamwork	Marketing skills	Bilingual Communication skills Teamwork

Table 4.2: The set of skills of each employee.

Skills	Number of employees
Bilingual	$\vert\vert$
Trilingual	$\vert$
Marketing skills	$\vert\vert\vert\vert$
Communication skills	$\vert\vert\vert$
Management skills	$\vert\vert\vert\vert$
Teamwork	$\vert\vert\vert\vert$

Table 4.3: Number of employees having selected skills.

The following graph represents the eight employees with its vertices and its edges link two vertices when the two employees share a set or more of skills. It's important to note that even though two employees have more than a skill in common, they are linked only by an edge.

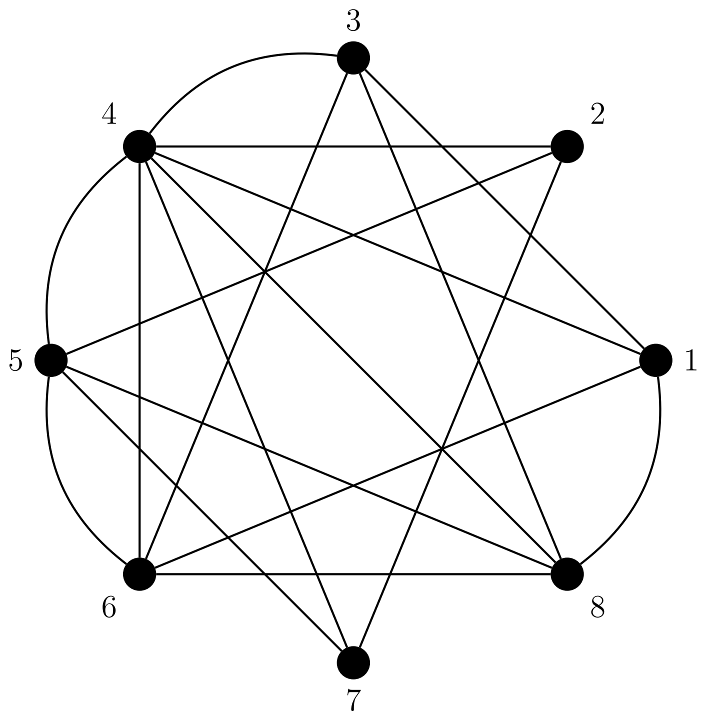


Figure 4.6: Graph $G = (V, E)$ represents the employees of the company.

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 4.7: The adjacency matrix of G.

4.3.2 Solving approach

In order to determine which necessary cutbacks are to be made, one of the suggestions is to attempt coloring the graph in an optimal manner. If the graph G is an indifference graph then optimal coloring will not be much of an obstacle. Thus, the optimal graph coloring showcases the following:

- The vertices of the same color represent the employees offering the company the same set of skills.
- The number of colors used represents the number of distinct sets of skills the company has.
- A vertex having a distinct color represents an employee that is obviously indispensable to the company.

The company determines how many individuals sharing the same skills it requires within a single department and/or in different departments according to its needs. Then, the CEO may decide which employees are indispensable to the company and those it can do without. A potential employee to lay off will be represented by a vertex that shares the same color as other vertices. This way the company can determine which employees have distinct sets of skills that are able to perform a wide range of tasks.

4.3.3 Results

The application software is able to investigate whether the graph G is an indifference graph and consequently submit an optimal coloring of the graph. After launching the program (figure 4.2), the software allows to choose from a variety of options in the menu (figure 4.8).

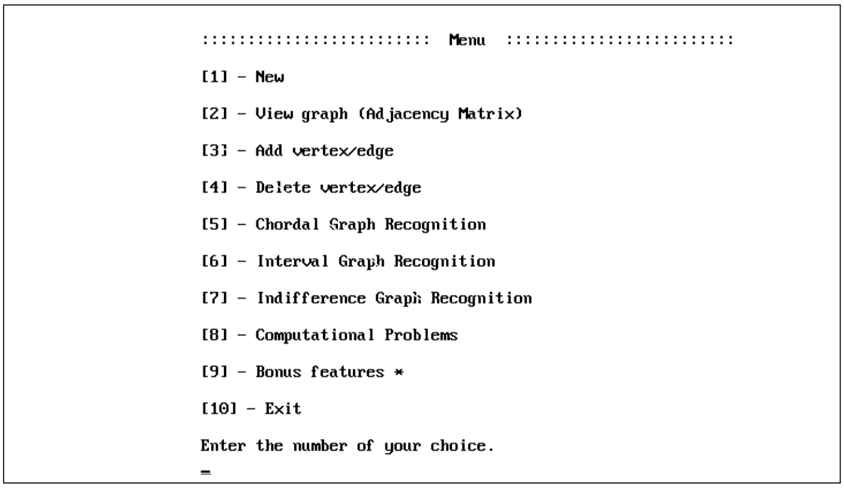


Figure 4.8: Application software menu.

Options 1 through 4 allow to manipulate the data representation of the graph G. In fact, they permit insertion, viewing, addition, and deletion of different elements of the graph.

Options 5, 6, and 7 offer to execute recognition algorithms on the graph G so one may know if the graph is chordal, an interval or an indifference graph.

Option 8 leads to another menu (figure 4.9) that permits to choose between the graph coloring problem and the maximum matching problem. It then executes one of their solving greedy algorithms according to the user's choice. Similarly, option 9 also leads to a secondary menu that contains extra features such as: maximum cardinality search, MCS, checking PEO, LBFS, and the generation of maximal cliques. However, this option is still under construction.

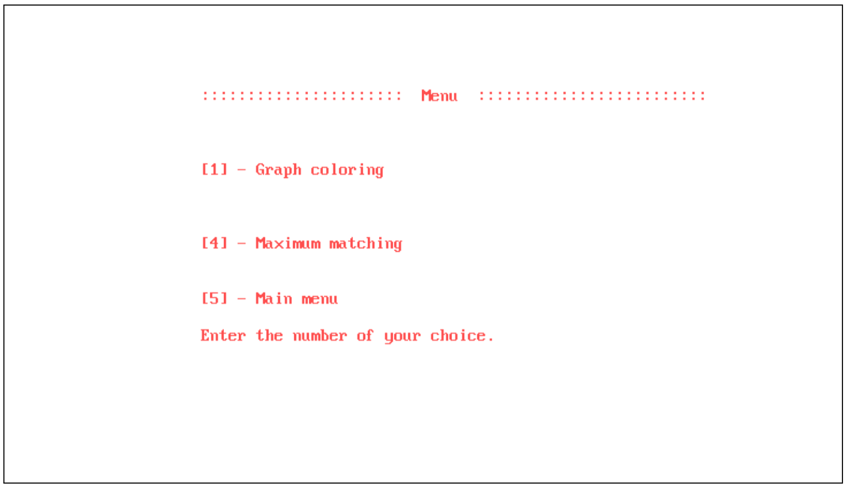


Figure 4.9: Computational problems menu.

The adjacency matrix of graph G (figure 4.6) is inserted into the program through option 1 (figure 4.10) after insertion of the size of the graph.

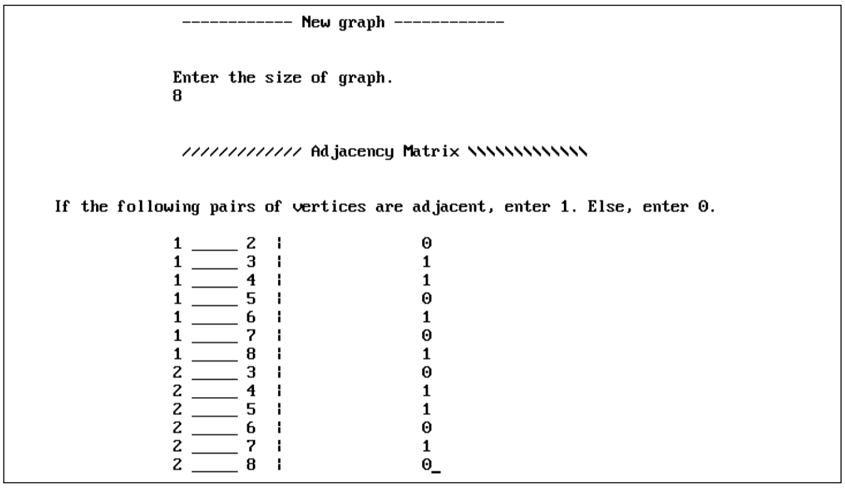


Figure 4.10: Creation of the adjacency matrix (4.7).

So that one may view the resulting adjacency matrix, option 2 must be chosen. The adjacency matrix of figure (4.7) is displayed in figure (4.11).

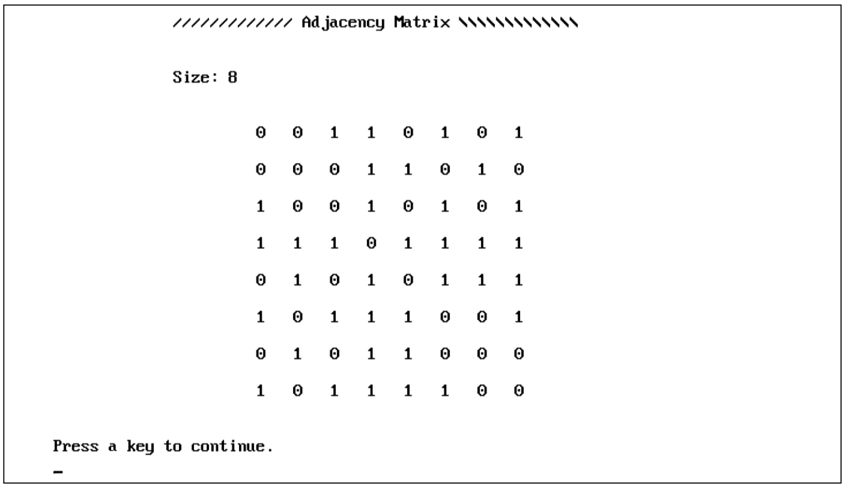


Figure 4.11: Creation of the adjacency matrix (4.7).

When selecting consecutively options 6, 7, and 8, the following results are obtained:

- Chordal Graph Recognition:
  - The Maximum Cardinality Search generates a Perfect Elimination Ordering (figure 4.12).

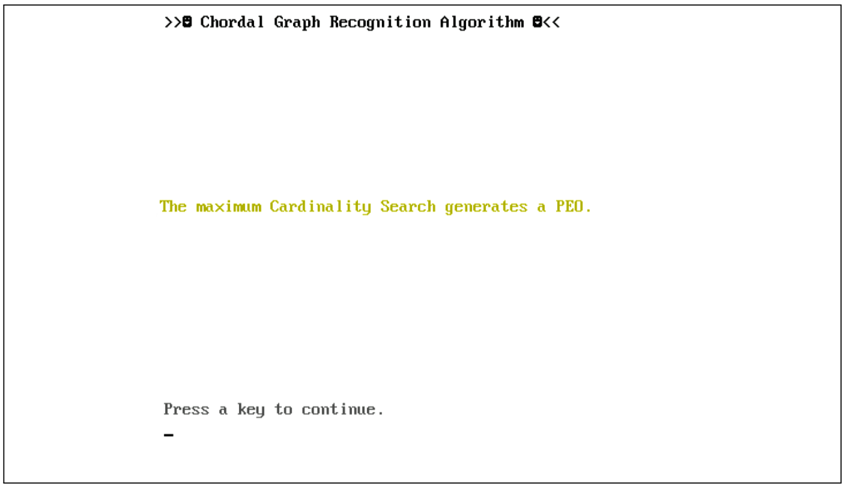


Figure 4.12: Maximum Cardinality Search on G (4.6).

  - The graph G is a chordal graph (figure 4.13).

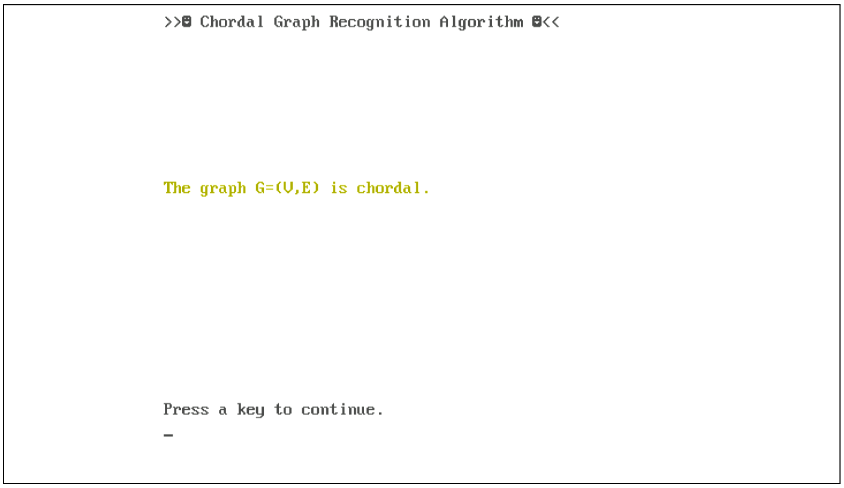


Figure 4.13: Chordality result of G (4.6).

- The maximal cliques of the graph G (figure 4.14) are:
  $C_1 = \{5, 4, 6, 8\}$;
  $C_2 = \{7, 2, 4, 5\}$;
  $C_3 = \{8, 1, 3, 4, 6\}$.

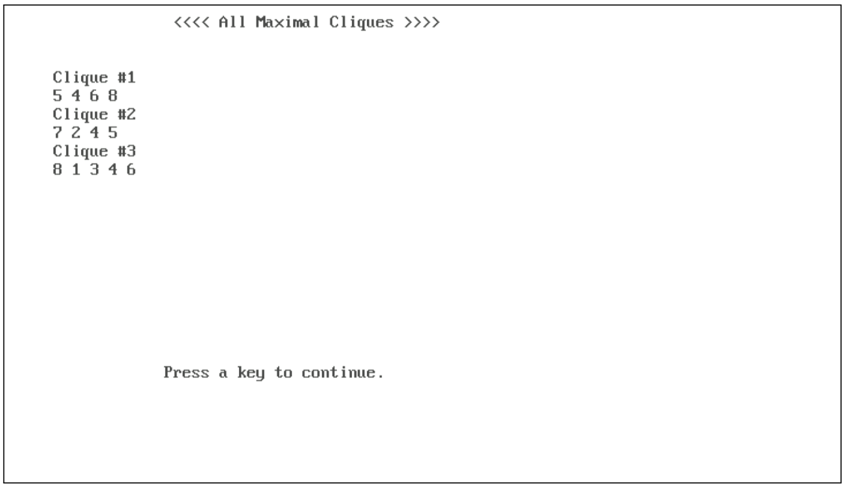


Figure 4.14: Generation of the maximal cliques of G (4.6).

- Interval Graph Recognition:
  - The graph G is an interval graph (figure 4.15).

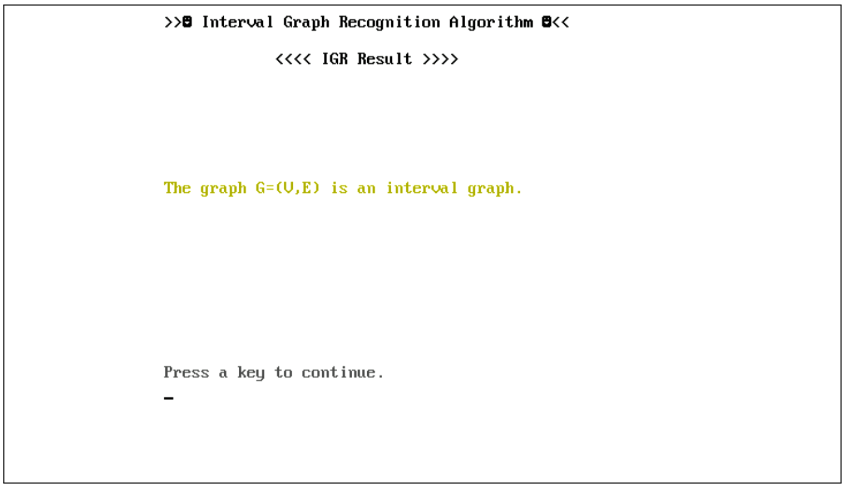


Figure 4.15: Interval graph recognition result of G (4.6).

- Indifference Graph Recognition:
  - The graph G is an indifference graph (figure 4.16).

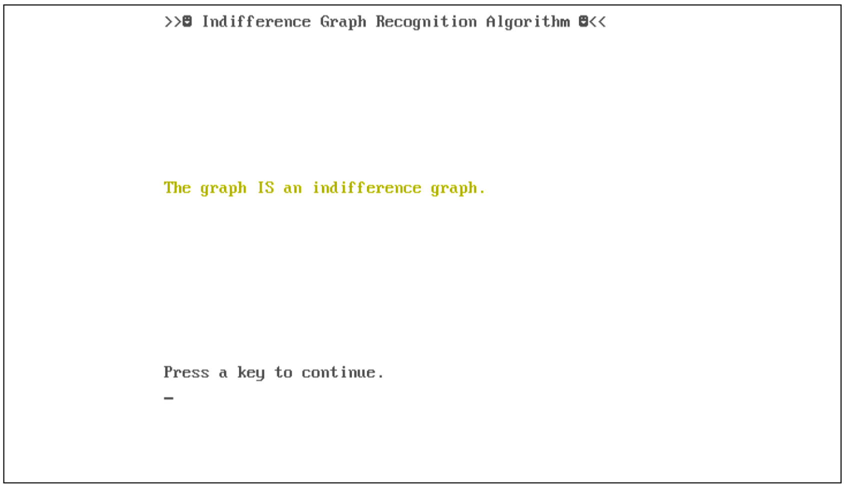


Figure 4.16: Indifference graph recognition result of G (4.6).

- Graph Coloring: (since the cardinal of the largest clique is equal to five; the chromatic number of G is five)
  - Let the colors $\{blue, red, green, yellow, purple\}$ be represented respectively by the color codes (whole numbers) $\{1, 2, 3, 4, 5\}$
  - The following table shows the optimal coloring of graph G as well as figure (4.17):

Vertex	Employee	Color code	Color
1	Jefri	2	red
2	Natalia	1	blue
3	Slimane	3	green
4	Judith	4	yellow
5	Hayato	2	red
6	Omar	5	purple
7	Steven	3	green
8	Nadia	1	blue

Table 4.4: Table of color assignments.

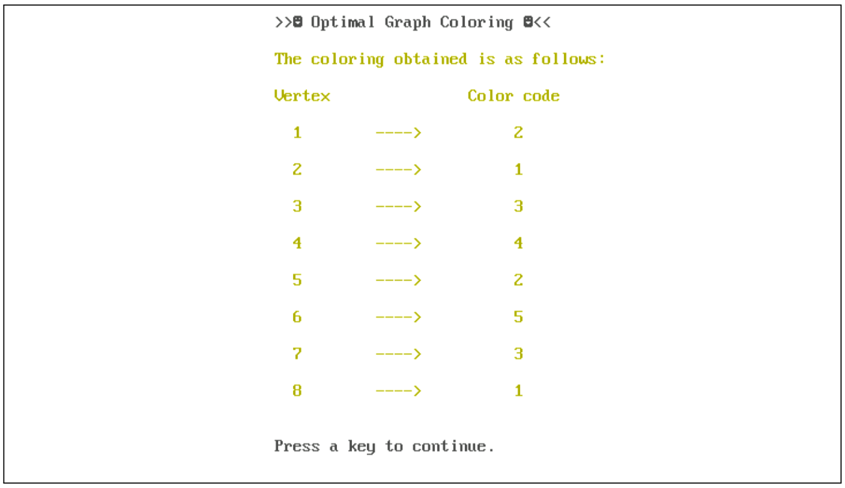


Figure 4.17: Graph coloring result of G (4.6).

The table (4.4) shows that both *Omar* and *Judith* have distinct colors compared to the others. This indicates that these two employees are essential to the company as they have a diverse set of skills, that enables them to accomplish a wider range of tasks than the other employees.

It also shows that *Jefri* and *Hayato*, *Natalia* and *Nadia*, as well as *Slimane* and *Steven* have respectively the colors red, blue and green (table 4.5 and figure 4.18). In these cases, the CEO can choose between employees sharing a same color. This is due to the fact that employees sharing the same color are capable of carrying out the same range of tasks. So, if the CEO decides to lay off employees he should choose from the ones sharing the same colors but leaving at least an employee of each color.

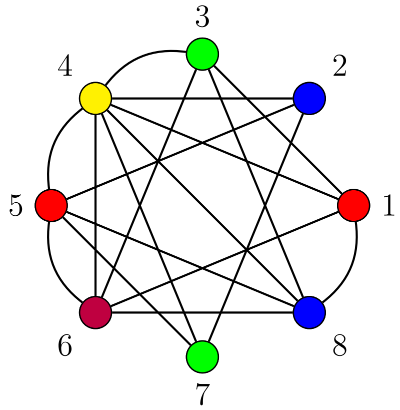


Figure 4.18: Coloring of graph G.

The CEO makes his choice according to both the departments' and the company's needs. For instance, he decides to keep *Slimane*, *Judith*, *Hayato*, *Omar*, and *Nadia*.

Color code	Color	Employees
1	Blue	Jefri and Hayato
2	Red	Natalia and Nadia
3	Green	Slimane and Steven
4	Yellow	Judith
5	Purple	Omar

Table 4.5: Assignment of colors to the employees.

Conclusion

Nowadays, many research fields require the introduction of new concepts of measurability such as the notions of preference and indifference. This work emphasizes on the concept of indifference and its major utility in disciplines such as seriation, political sciences, marketing, and many more. It attempts to define the structural properties of indifference and determine essential tools to solve problems tightly related to it.

Seeing that the use of precise, graph theoretical reasoning provides tools to help in making decisions on problems arising from real-life situations or of practical nature; this work, adopts graph theory as an adequate choice to study the notion of indifference through a class of indifference graphs.

Indifference graphs are part of a larger family of graphs namely chordal and interval graphs. Therefore they form a crucial element for the study of indifference graphs. Even more so when it comes to building a greedy algorithm with a recognition perspective. For this, a particular characterization of indifference graphs was pinpointed, that resides in a special ordering of their vertices, is elementary to the greedy algorithm.

It is important to note that the greedy algorithms contained in this work are subject to present and future improvements. This could take effect through the introduction of local optimization procedures to improve the obtained results. Within the context of future perspectives of this work, it is interesting to consider the study of the choice of the positive δ value used to define and measure the notion of indifference.

By means of summarizing the contents of this work, the first chapters introduced one of the measurability theory tools which is indifference and illustrates how graph theory offers adequate tools of representing such as relations between objects or alternatives. Moreover they describe the theory of computational complexity and its importance. More importantly, they present greedy recognition algorithms to recognize chordal graphs, interval graphs and indifference graphs.

The following chapters presented a couple of computational problems accompanied with simple greedy algorithms. The last includes a description of the program developed in the setting of this work along with a case study summarizing the concepts and notions of this dissertation.

The purpose of this work was to investigate the class of indifference graphs and its algorithmic properties. A specific characterization of indifference graphs in terms of a linear order on their sets of vertices has been presented along with illustrating that this linear order offers important elements to build optimal greedy algorithms to solve computational problems such as graph coloring and maximum matching.

Bibliography

[1] Borland: Pascal units. http://programmersheaven.com/categories/pasprog. Accessed: 23-05-2016.

[2] *Turbo Pascal*. Borland International, USA, 1992.

[3] Karen I. Aardal, Stan P. M. van Hoesel, Arie M. C. A. Koster, Carlo Mannino, and Antonio Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, 2007.

[4] Stuart Armstrong. Utility indifference. Future of Humanity Institute, Oxford University, Technical Report., 2010.

[5] Wallace E. Armstrong. The determinateness of utility function. *The economic journal: the journal of the Royal Economic Society*, 49:453–467, 1939.

[6] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, first edition, 2009.

[7] R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. Springer, 2012.

[8] Granville Barnett and Luca Del Tongo. *Data Structures and Algorithms: Annotated Reference with Examples*. DotNetSlackers, 2008.

[9] Padraic Bartlett. Perfect graph theory. Lecture 3: Chordal Graphs. *Mathcamp*, 2011.

[10] Claude Berge. Färbung von Graphen, deren sämtliche bzw, deren ungerade Kreise starr sind (Coloring of graphs with rigid odd circuits). *Zeitschrift der Martin-Luther-Universität Halle-Wittenberg*, pages 114–115, 1961.

[11] Claude Berge. *Graphes et Hypergraphes*. Dunod, Paris, 1973.

[12] Jean R. S. Blair and Barry W. Peyton. An introduction to chordal graphs and clique trees. *The IMA Volumes in Mathematics and its Applications*, 56(1):1–9, 1993.

[13] Kenneth P. Bogart and Douglas B. West. A short proof that 'proper=unit'. https://pdfs.semanticscholar.org/3a14/7274b23cc04dbc3d3ca62aeaec827695e555.pdf. Accessed: 04-08-2016.

[14] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[15] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes*. SIAM, Philadelphia, 1999.

[16] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The ultimate interval graph recogntion algorithm? In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 175–180, Philadelphia, 1998.

[17] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM Journal of Discrete Mathematics*, 23(4):1905–1953, 2009.

[18] Franz Dietrich and Christian List. Where do preferences come from? *International Journal of Game Theory*, 42(3):613–637, 2013.

[19] Gabriel A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1):71–76, 1961.

[20] Peter C. Fishburn. Intransitive indifference in preference theory: A survey. *Operations Research*, 18(2):207–228, 1970.

[21] Delbert R. Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[22] Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. *Graph-Theoretic Concepts in Computer Science*, 1017:358–371, 2005.

[23] Philippe Gambette. Cours d'Algorithmique des Graphes du M.P.R.I. par Michel Habib, 2009.

[24] P. C. Gilmore and Alan J. Hoffman. A characterization of comparability graphs and interal graphs. *The Canadian Journal of Mathematics*, 16:539–548, 1964.

[25] Martin C. Golumbic. *Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[26] Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. Lexbfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1):59–84, 2000.

[27] Nora Hartsfield and Gerhard Ringel. *Pearls in Graph Theory*. Dover Publications, New York, 2003.

[28] Wen Lian Hsu. A new test for interval graphs. *Institute of Information Science, Academia Sinica, Taiwan, Republic of China*, 1993.

[29] David S. Johnson and Michael Garey. *Computers and Intractability*. W. H. Freeman and Company, USA, 1979.

[30] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[31] Chahid Khichane. *Turbo Pascal*. Editions El Maarifa, Bab El Oued, 2005.

[32] Keith Lehrer and Carl Wagner. Intransitive indifference: The semi-order problem. *Synthese: Action Theory*, 65(2):249–256, 1985.

[33] C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.

[34] Innar Liiv. Seriation and matrix reordering methods: An historical overview. Published online: <www.interscience.wiley.com>, March 2010.

[35] Peter J. Looges and Stephan Olariu. Optimal greedy algorithms for indifference graphs. *Computers and Mathematics with Applications*, 25(7):15–25, 1992.

[36] Duncan R. Luce. Semiorders and the theory of utility discrimination. *Econometrica*, 24(2):178–191, 1956.

[37] George S. Lueker and Kellogg S. Booth. A linear time algorithm for deciding interval graph graph isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.

[38] Alexander L. Macfie. Choice in psychology and as economic assumption. *The Economic Journal*, 63(250):352–367, 1953.

[39] Karl Menger. Probabilistic theories of relations. *Proceedings of the National Academy of Sciences of the United States of America*, 37(3):178–180, 1951.

[40] Norbert Korte and Rolf H. Möhring. An Incremental Linear–Time Algorithm for Recognizing Interval Graphs. *SIAM Journal of Computing*, 18(1):68–81, 1989.

[41] Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21–25, 1991.

[42] Linda L. Price, Eric J. Arnould, and George M. Zinkhan. *Consumers*. McGraw-Hill, New York, 2002.

[43] Fred S. Roberts. *Indifference Graphs.* Proof Techniques in Graph Theory, Academic Press, 1969.

[44] Fred S. Roberts. Indifference and seriation. *Annals of the New York Academy of Sciences*, 328(1):173–182, 1979.

[45] Frederic S. Roberts. *Graph Theory and its Applications to Problems of Society.* SIAM Press, Philadelphia, 1978.

[46] Donald J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematcial Analysis and Applications*, 32(3):597–609, 1970.

[47] Donald J. Rose, Robert E. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing*, 5(2):266–283, 1976.

[48] Murray N. Rothbard. *Economic Controversies*. Ludwig von Mises Institute, Alabama, 2011.

[49] Leon G. Schiffman and Leslie Lazarkanuk. *Consumer Behavior*. Pearson, New Jersey, 2004.

[50] George F. Schumm. Transitivity, preference and indifference. *Philosophical Studies*, 52(3):435–437, 1987.

[51] Dana Scott and Patrick Suppes. Foundational aspects of theories of measurement. *The Journal of Symbolic Logic*, 23(2):113–128, 1958.

[52] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wasley Professional, fourth edition, 2011.

[53] Halldén Sören. *On the logic of "etter"*. C. W. K. Gleerup, Lund, and Ejnar Munksgaard, Copenhagen, 1957.

[54] Sven Ove Hansson and Till Grüne-Yanoff. Preferences. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. `http://plato.stanford.edu/archives/win2012/entries/preferences/`, winter 2012 edition, 2012.

[55] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13(3):566–579, 1984.

[56] Nazli Temur. Study of LexBFS algorithm and its applications. `https://academia.edu/15721202/LEXBFS_Algorithm_and_Its_Applications`, 2015. Research Project Final Report.

[57] Lucas Trevisan. Computational complexity. Handout *Stanford University*, 2010.

[58] Aaron Wildavsky. Choosing preferences by constructing institutions: A cultural theory of preference formation. *The American Political Science Review*, 81(1):3–22, 1987.

[59] Niklaus Wirth. *Algorithms and Data Structures*. Prentice Hall, 1985. eBook version (2012).

[35] [15] [25] [23] [34] [45] [48] [33] [24] [14] [37] [21] [36] [3] [11] [**?**] [9] [31] [1] [2] [47] [55] [22] [46] [19] [12] [10] [16] [26] [17] [28] [40] [59] [29] [8] [6] [52] [58] [30] [44] [50] [54] [39] [53] [5] [4] [51] [7] [18] [38] [42] [49] [43] [13] [41]