CHAPTER I

INTRODUCTION

Two finite graphs are isomorphic if there exists a bijective map between the vertex sets of the two graphs which preserves adjacency. Determining whether two given graphs are isomorphic is a problem of both practical and theoretical interest, and there has been extensive work investigating whether graph isomorphism can be tested efficiently. Despite much work, there is to date no polynomial time test for graph isomorphism, nor is there a proof that no such test can exist. Nevertheless, there do exist algorithms which can test isomorphism of certain classes of graphs in polynomial time.

The two dominant lines of attack on graph isomorphism are topological and group-theoretic. In this monograph, we give a comprehensive development of the group-theoretic approach. Within a very short time, this approach has substantially broadened the class of graphs for which there exist polynomial time isomorphism tests and it has stimulated interest in a number of algebraic problems which had not been previously investigated for their computational complexity.

In the group-theoretic approach, one determines the group of all automorphisms of the graph. It is not too difficult to prove that knowledge of the automorphism group enables one to also test for isomorphism. At the same time, the translation of the topological question of testing graph isomorphism into the algebraic question of determining the symmetries of the graph allows the use of many new computational techniques. In particular, there are many algorithms for determining properties of permutation groups which become applicable for tests of graph isomorphism. The group-theoretic approach enables one to view the problem of graph isomorphism from a new perspective. Because of this, the algebraic approach has rapidly yielded significant new results.

Most of the algorithms developed in this monograph have been selected because of their bearing on the graph isomorphism problem. However, graph isomorphism can also be generalized, yielding a spectrum of algebraic problems of apparently greater difficulty. The generalizations are natural in the sense that they have the same "structure" as graph isomorphism, but they are also generalizations in the
computational sense that a polynomial time algorithm for the more general problem implies a polynomial time algorithm for graph isomorphism. We investigate those problems as well.

The algebraic generalizations of graph isomorphism differ from previous generalizations in at least two significant aspects: First, they are not themselves isomorphism questions, and second, while these problems are clearly in \( \text{NP} \), there is (just as in the case of graph isomorphism) some technical evidence that the generalizations are not \( \text{NP} \)-complete. Thus we conjecture that these generalizations are part of a hierarchy of increasingly more difficult problems within \( \text{NP} \).

1. **Graph Isomorphism**

Graph isomorphism has received considerable attention for a number of reasons, both practical and theoretical. For example, in combinatorial studies, one frequently wishes to generate a list of combinatorial objects in which each object occurs exactly once. This can be done in two phases: First, a list is constructed in which each object appears at least once. Then, multiple occurrences are deleted. Usually, these objects correspond in some natural sense to graphs, so recognizing multiple occurrence of an object requires testing isomorphism of graphs. As other applications of graph isomorphism, combinatorial designs, scene analysis, and chemical documentation are frequently cited.

Part of the theoretical appeal of graph isomorphism is its unknown complexity status. It is clearly a problem in \( \text{NP} \), it is not known to be in \( \text{P} \), and there are properties of the problem which seem to make it unlikely to be \( \text{NP} \)-complete. Thus, there is a good possibility that it is a problem of intermediate difficulty, i.e., a problem which is neither in \( \text{P} \) nor is \( \text{NP} \)-complete.

As mentioned above, the two dominant approaches to graph isomorphism are the topological approach and the group-theoretic approach. In the topological approach, one embeds the graphs onto a surface of minimal genus. Then one dissect the surface (and with it the graph) into planar components. With a careful study of the possible interconnection structure, one then reduces testing isomorphism of graphs embedded in a surface of nonzero genus to testing isomorphism of planar graphs.
The topological approach leads to the \textit{genus hierarchy}: For each genus $g$, there exists a polynomial $p_g$ of degree $f(g)$ such that isomorphism of graphs embeddable onto a surface of genus $g$ can be tested in at most $p_g(n)$ steps, where $n$ is the number of vertices of the graph.

Note that the problem of determining the graph genus is of comparable difficulty. That is, the algorithm for determining the genus of a graph requires time proportional to a polynomial whose degree grows with the graph genus.

In the group-theoretic approach to graph isomorphism, one seeks to determine a set of generating permutations for the automorphism group of the graph. It can be shown that testing graph isomorphism is polynomially reducible to determining generators for the automorphism group of graphs and that there are small sets of generating permutations. We prove this result in Chapter II. Typically, the group-theoretic approach is based on structural properties of the automorphism group which are the result of graph properties such as the valence of the graph. While the graph properties exploited are often trivial, the resulting group-theoretic properties may be quite complex.

As an illustration of the group-theoretic approach, consider determining the automorphism group of a connected trivalent graph $X$. Here one proves that the subgroup consisting of those automorphisms which fix an arbitrarily chosen edge in the graph is a 2-group; that is, it is a group each of whose elements has order a power of 2. It turns out that 2-groups have special properties which can be exploited in the design of algorithms to determine the intersection of two groups, one of which is a 2-group. These algorithms run in time polynomial in the number of graph vertices, despite the fact that the order of these groups might very well be exponential. Such an algorithm serves as the basis for an isomorphism test of trivalent graphs. We develop this approach in Chapter IV.

The group-theoretic approach leads to the \textit{valence hierarchy}: For each valence $d$, there exists a polynomial $p_d$ of degree $h(d)$ such that isomorphism of a graph $X$ of valence $d$ can be tested in $p_d(n)$ steps, where $n$ is the number of vertices of $X$. This result is presented in Chapter V. It is possible to broaden the class of graphs handled by this approach in polynomial time, and in Chapters IV and V we discuss which graph-theoretic properties are required.
At present, no concrete facts are known about the relationship between the topological and the group-theoretic approach to graph isomorphism. It is not known whether there is a technical connection, or even a more general technique of which both approaches are special instances.

It should be pointed out that the functions $f(g)$ and $h(d)$ which bound the complexity of the isomorphism tests in the two hierarchies grow so fast that the respective algorithms cannot be considered practical. Exceptions to this are the isomorphism test for planar graphs, which is linear in the graph size, and, possibly, the isomorphism test for trivalent graphs, which presently requires $O(n^4)$ steps, where $n$ is the number of graph vertices. We believe, however, that practical isomorphism tests for graphs of higher genus or of higher valence may yet be discovered.

2. Computational Complexity

There are many problems of interest which, for all practical purposes, cannot be solved by computer in any reasonable amount of time, even though they are in principle computable. The discovery of such intractable problems has given special motivation to the study of the inherent computational complexity of algorithmic problems in Computer Science.

Despite many successes in this research, it remains an open problem to demonstrate the exact relationship of the (worst case) complexity classes $P$, $NP$, and $coNP$. Much attention has been given to this question. The class $P$ includes all problems for which an algorithm exists which is practically efficient in all cases. On the other hand, the class $NP$ contains a large number of problems of practical interest for which it remains unknown whether they are truly intractable. The question of whether $P = NP$ is a long standing open problem usually referred to as the $P$ vs. $NP$ problem. We now informally explain the complexity classes $P$, $NP$ and $coNP$, and the concept of $NP$-completeness.

Let $\Sigma$ be a fixed finite alphabet, $\Sigma^*$ the set of all finite strings over $\Sigma$. A problem or a language is just a subset $L \subseteq \Sigma^*$, where it is understood that its solution is an algorithm for testing whether $x \in L$ for arbitrary strings $x \in \Sigma^*$. For example, the language $L$ might consist of all well-formed propositional formulae which are
satisfiable, or it might consist of all integer triples \((a,b,c)\) such that \(c\) is the product of \(a\) and \(b\).

A problem \(L\) is said to be in \(P\) (in \(NP\)) if there exists a deterministic (a nondeterministic) Turing machine \(M_L\) which accepts the language \(L\) and is such that, on each input \(x \in \Sigma^*\) of length \(n\), \(M_L\) terminates in at most \(p(n)\) steps, where \(p\) is a polynomial of fixed degree.

The notion of acceptance by a nondeterministic Turing machine \(M_L\) may be unfamiliar. First, \(M_L\) is a nondeterministic machine due to the presence of configurations in which there is a choice of possible next steps in the computation. This choice is not subject to probabilistic notions. Consequently, the machine \(M_L\) accepts \(x \in \Sigma^*\) if there exists a computation with input \(x\) which ends in an accepting final state. Acceptance of \(x\) does not preclude the possibility that other computations on \(M_L\) with input \(x\) do not terminate in an accepting final state. (Such other computations arise from differently chosen computation steps). Similarly, \(M_L\) rejects \(x \in \Sigma^*\) if every possible computation with input \(x\) fails to end with an accepting final state.

Finally, a problem \(L\) is in \(coNP\) if its complement \(\overline{L} = \{ x \in \Sigma^* \mid x \notin L \}\) is in \(NP\). For example, testing whether a propositional formula is a tautology is a problem in \(coNP\).

The class definitions are invariant with respect to the computing model, as long as reasonable programming systems are chosen. Instead of Turing machines, one may use, e.g., random access machines (RAM). Hence the reader can trust his intuitions about what constitutes a step in the computation.

\(P\) is clearly a nonempty class and is contained in the intersection of the classes \(NP\) and \(coNP\). It is not known whether this containment is proper. It is also unknown whether \(NP\) and \(coNP\) are different classes.

A problem \(L \in NP\) is called \(NP\)-complete if \(L \in P\) implies that \(P = NP\). Although there is no immediate intuition telling us that there should exist \(NP\)-complete problems, there are more than two thousand known \(NP\)-complete problems. Many of these problems are natural, i.e., they arise as computational problems of general practical interest. For example, finding an optimal length tour visiting every city in a set of \(n\) cities once is a natural \(NP\)-complete problem.

The reader unfamiliar with Computational Complexity may wonder how it is possible to prove \(NP\)-completeness of a problem. We sketch the proof that testing satisfiability of propositional formulae is \(NP\)-complete. Essentially, the proof is a
procedure to express the following sentence by a propositional formula: "For the machine \( M_L \) accepting \( L \in \text{NP} \), and for the input \( x \in \Sigma^* \), there exists a computation requiring time polynomial in the length of \( x \) on \( M_L \) which accepts \( x \)." For any fixed machine \( M_L \), the propositional formula constructed has length polynomial in the length of \( x \). If satisfiability of this formula can be decided deterministically in time polynomial in the length of the formula, it now follows that it can be decided deterministically in time polynomial in the length of \( x \) whether \( M_L \) accepts \( x \).

Once a problem \( P_0 \) (such as Satisfiability) has been shown to be \( \text{NP} \)-complete, it becomes simpler to show \( \text{NP} \)-completeness of another problem \( P_1 \). We now need only show that

1. \( P_1 \) is in \( \text{NP} \), and
2. that if \( P_1 \) were in \( \text{P} \), then so would be \( P_0 \).

Step (1) amounts usually to exhibiting a nondeterministic polynomial time algorithm for \( P_1 \), and is frequently straightforward. Step (2) involves a reduction of \( P_0 \) to \( P_1 \) requiring deterministic polynomial time. That is, one designs a deterministic algorithm for \( P_0 \) which uses an algorithm \( A \) for \( P_1 \) as subroutine. Except for the time required to execute \( A \), the algorithm for \( P_0 \) must be polynomial time.

If \( \text{P} \) is not equal to \( \text{NP} \), then one can show the existence of many problems of intermediate difficulty, i.e., problems which are in \( \text{NP} \) but are neither complete nor are in \( \text{P} \). These problems have been constructed artificially from \( \text{NP} \)-complete problems and are not natural. Furthermore, despite the large numbers of known natural \( \text{NP} \)-complete problems and of natural problems in \( \text{P} \), there seems to be a paucity of candidates for natural problems of intermediate difficulty. The only two commonly proposed candidates are testing graph isomorphism and testing primality.

These two problems seem to differ significantly in character: Clearly, one can recognize composite numbers in nondeterministic polynomial time; one guesses two factors and verifies that the number is their product. Furthermore, for every prime \( p \) there is a proof of its primality of length \( O((\log_2(p))^2) \). This proof can be verified in time proportional to its length, and so primality is a problem in \( \text{NP} \cap \text{coNP} \). However, since we do not know of short proofs of nonisomorphism of graphs, it is not clear that graph isomorphism is a problem in \( \text{NP} \cap \text{coNP} \).

There are other properties which distinguish graph isomorphism: Recall that a problem is a membership test in a language \( L \). We may call such a problem an existence problem since we ask for the existence of an accepting computation of the
machine $M_x$ with input $x$. Now with many existence problems, there is a naturally associated counting problem. For example, the counting problem for satisfiability is to determine the number of distinct assignments to the literals in a propositional formula $F$ which satisfy $F$. Clearly, if $F$ is not satisfiable, then this number is zero. The counting problem for graph isomorphism is to determine the number of isomorphisms between two graphs. It has been conjectured that for NP-complete problems the existence problem is easier than the associated counting problem. However, for graph isomorphism counting and existence have equal difficulty. That is, testing isomorphism of two graphs is polynomially equivalent to counting the number of isomorphisms between the graphs. This fact has been interpreted by a number of authors as technical evidence that graph isomorphism is not likely NP-complete.

While testing isomorphism for arbitrary graphs may ultimately prove to belong to P, we suspect that it is neither in P nor NP-complete. Thus a theoretical motivation for studying the complexity of testing graph isomorphism is its candidacy for a problem of intermediate difficulty.

3. Group-Theoretic Algorithms

Most of the group-theoretic algorithms in this book have been selected primarily because of their relevance to graph isomorphism and to the P vs. NP question. However, the study of group-theoretic problems by computer has been pursued independently for almost 30 years, so the algorithms given here constitute only a small fraction of known techniques.

Roughly speaking, group-theoretic algorithms fall into one of two categories: Algorithms for (suitably specified) permutation groups, and algorithms for abstract groups presented by generators and relations fulfilled by them. Of course, there are some algorithms which combine methods from both categories.

We will be concerned exclusively with algorithms for finite permutation groups. Furthermore all group-theoretic problems we consider clearly have algorithmic solutions, although a number of them appear to be computationally intractable. (On the other hand, for abstract groups many problems are not just intractable, but are absolutely undecidable. For example, it is recursively undecidable whether a finitely presented abstract group is of finite order.)
Many of the group-theoretic algorithms in this book have been known and used extensively for many years. Nevertheless it appears that in the majority of cases they have not been formally analyzed for their asymptotic time complexity. In part this may be attributed to the exceedingly individual nature of the problems investigated in Computational Group Theory. For example, an algorithm may have been designed to investigate a fixed simple group and is not applicable to all simple groups. Hence it would not be meaningful to analyze its asymptotic behavior. Furthermore, in many group-theoretical applications space complexity is more often critical than asymptotic time complexity.

We will deal exclusively with algorithms applicable to infinite classes of groups and we will always analyze their time complexity. Space complexity is not analyzed, since it is usually apparent from the algorithm. Sometimes the timing analysis is not as refined as one might wish. In many cases not enough is known about the combinatorial interplay of the various factors affecting efficiency. An example may illustrate the situation.

Sims' algorithm determines the order (and a set of strong generators) of a permutation group of degree n from a given set of generating permutations (cf. Algorithm 3 of Chapter II). It is one of the most basic algorithms used for computing with permutation groups. Under the reasonable assumption that the initial generating set is small, the algorithm has an \( O(n^6) \) asymptotic worst case time bound. In Chapter II we give a proof that an \( O(n^5) \) bound can be achieved provided that one can pick a suitable set of strong generators in the course of the computation. (At present it is not known how to pick the generators correctly.) But it is also clear that the groups for which the worst case time bound is attained have a special structure and should be handled differently. Hence it appears that an even faster algorithm is possible.

In practice, one observes that Sims' algorithm is substantially improved by adding a coset enumeration procedure. Now coset enumeration (the Todd-Coxeter Algorithm) is a technique developed for abstract groups, and for certain finite group presentations it does not halt. Consequently, one incorporates an interruptible coset enumeration procedure and carefully balances the time spent enumerating cosets against the time spent forming and sifting pair products (the usual activity of Sims' algorithm). Empirically, one observes that a well-tuned implementation of this algorithm runs roughly linear in the size of the computed coset matrix. However, an
exact analysis of the algorithm is at present unavailable, and the contributing factors in the observed improvement are not understood. As a consequence, no worst case lower bounds are known.

For many group-theoretic algorithms this situation is typical; hence we have omitted a number of heuristics which, while leading to an observable improvement in performance, are not fully analyzable at this time. As a result, the algorithms given here usually have a very simple structure. Almost all of our polynomial time algorithms have a high degree bounding the worst case. It is therefore legitimate to ask whether these algorithms are really practical. As is the case with Sims' algorithm, we believe that this question cannot be fully answered at this time. Also, since a number of these algorithms are fairly recent, we expect better techniques to be found. Consequently we look at the results presented here as a stage in an ongoing historical process. Nevertheless, the empirical classification made by Computational Group Theory of these algorithms into "easy", "moderately difficult", and "hard" seems to correlate rather well with our complexity classifications into "polynomial of small degree", "polynomial of high degree", and "possibly not in P".

4. Background

Of necessity the following chapters contain a lot of group-theoretic material. In fact, the group-theoretic verification of a number of algorithms seems far more sophisticated than the algorithmic techniques employed. For a reader without extensive algebraic background, we have developed and proved all necessary results from Group Theory by completely elementary arguments. There are, of course, a number of good texts available containing the majority of these results. However, since they are often presented from a different perspective and to a different purpose, we hope the reader will find our presentation useful.

Most of the algorithmic techniques we use are quite elementary, and none go beyond what a good undergraduate course in algorithms should teach. We assume that the reader has a basic familiarity with these techniques, e.g., with the disjoint set union/find algorithm or the tree isomorphism algorithm. Almost no graph-theoretic concepts are needed.
We analyze the running time of all algorithms which we present. As is usual in the analysis of asymptotic behavior, one need not be very concerned with the underlying model of computation. However, we use as underlying model of computation the familiar random access machine with uniform cost of its instructions, since this model corresponds well with intuitive notions of computation steps.

We will use the term efficient algorithm in a technical sense to mean a polynomial time algorithm (as opposed to, e.g., an exponential one).

In order to give an uninterrupted and fluid presentation, we defer a discussion of the origin of the material until the final section of each chapter.

Lemmata, theorems, examples, etc., are sequentially and separately numbered beginning with 1. Theorems (lemmata, etc.) of the same chapter are simply referred to as "Theorem x". Theorems of other chapters are always referred to as "Theorem x of Chapter y". Chapters are subdivided into sections, which, in turn, may be subdivided into subsections.

5. Notes and References

The topological approach to graph isomorphism begins with algorithms for testing planarity and with algorithms for testing isomorphism of planar graphs. See, for example, Hopcroft and Tarjan [1973, 1974], and Hopcroft and Wong [1974]. The approach was fully developed for surfaces of higher genus by Filotti [1978], Filotti, Miller and Reif [1979], Filotti and Mayer [1980], and Miller [1980].

One of the earlier papers studying graph isomorphism with a group-theoretic approach is Miller [1979]. More recently, the group-theoretic approach has been taken in Babai [1979], Hoffmann [1980a], Furst, Hopcroft and Luks [1980a], Luks [1980], and Hoffmann [1981b].

Read and Corneil [1977] gives a survey of some of the older approaches to testing graph isomorphism, such as vertex classification schemes and proposed invariants to characterize isomorphism classes. Booth and Colbourn [1979] contains a comprehensive list of problems polynomially equivalent to graph isomorphism. Colbourn [1978] gives an annotated bibliography for graph isomorphism and its applications.

Most textbooks on algorithms will contain a more or less detailed treatment of the complexity classes $P$, $NP$, and $coNP$. A discussion of the usual choices of models
of computation can be found in Aho, Hopcroft, and Ullman [1974].

The definition of NP-completeness is due to Cook [1971]. Garey and Johnson [1979] give a good introduction to the class NP and to NP-completeness. It also contains a comprehensive list of NP-complete problems. Other books discussing the classes P, NP, and coNP include Aho, Hopcroft and Ullman [1974], Baase [1978], Horowitz and Sahni [1978], Lewis and Papadimitriou [1981], and Reingold, Nievergelt, and Deo [1977].

Problems of intermediate difficulty have been investigated by Ladner [1975]. Under the assumption that $P \neq NP$, Ladner shows that there must exist problems of intermediate difficulty, and that some of these form dense hierarchies with respect to polynomial time reduction. A proof that primality is in $NP \cap coNP$ has been given by Pratt [1975].

The earliest group-theoretic algorithm seems to be the Todd-Coxeter Algorithm for enumerating cosets of a subgroup of a finitely presented abstract group. A good account of the basic method can be found in Coxeter and Moser [1957]. A discussion of an implementation of the algorithm and empirical experience with it can be found in Cannon, Dimino, Havas and Watson [1973]. The incorporation of coset enumeration into Sims' algorithm is discussed by Sims [1978a], Butler [1979], and Leon [1980].

Previously analyzed group-theoretic algorithms are, e.g., the primitivity test of Atkinson [1975] and the centralizer algorithm of Fontet [1977].

There are many good textbooks and monographs on Group Theory. See, e.g., Hall [1959], Huppert [1967], and Kochendörffer [1970]. Wielandt [1964] treats exclusively finite permutation groups. A bibliography of Computational Group Theory is given in Felsch [1978].

The tree isomorphism algorithm and the disjoint set union/find algorithm are described and analyzed in detail in Aho, Hopcroft, and Ullman [1974]. A less compact description of the disjoint set/union find algorithm can be found in Baase [1978].