Prospectus
NP-Completeness: Theory and Applications

Teofilo F. Gonzalez
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
teo@cs.ucsb.edu
July 6, 2007

1 Introduction

Even before the 1970s a distinction was made between problems that could be solved efficiently (polynomial time) from those that did not seem to have such algorithms. Among the early researchers that made this distinction were von Neumann, Cobham, and Edmonds. At that time, all the problems that could not be solved efficiently (or computational difficult problems) seemed for the most part different from each other. In the early 1970s a class of (apparently) intractable (or computationally difficult) problems called NP-Complete was formally introduced. Cook established this class of problems, and satisfiability (SAT) was the first problem to be proven to be NP-Complete. He also showed that 3-SAT and subgraph isomorphism belong to this class of problems, and conjectured that there were other problems that shared this property. A year later Karp redefined this class of problems and showed that several well-known problems were also NP-Complete. These problems were decision version of classical optimization problems from several research areas. From then on problems in virtually all (quantitative) research areas have been shown to be NP-Complete and Karp’s definition became the standard. Working independently in the Soviet Union, Levin established around the same time a practically equivalent class of problems. He showed that many combinatorial problems can be proven to be as hard as the tiling problem which, he had known for a some time, was universal, i.e. at least as hard as any search problem.
Before we go further let us consider the following problem. Suppose that we have a set \( Q \) of \( n \) people who have interacted with each other for some time. These interactions have resulted in a list of pairs of incompatible individuals \( L = (l_1, l_2, \ldots, l_m) \), i.e., the two people in each pair are incompatible.

The compatible meeting (CM\(_3\)) problem is given any set of people \( Q \) and any list \( L \) of pairs of incompatible individuals, determine whether or not each person in \( Q \) can attend one of three meetings in such a way all the people attending each meeting are compatible\(^1\). This is a decision problem, meaning that its answer is simply “yes” or “no”. A yes-instance (resp. no-instance) is an instance whose answer is “yes” (resp. “no”). Computationally the CM\(_3\) problem is time consuming to solve. Each individual may be assigned to any of the three groups, therefore there are \( 3^n \) possible assignments of individuals to the three meetings. Checking all the possible assignments for a valid one takes in the worst case \( O(n^3) \) time. With the worst case being when the answer to the problem is “no”. It is simple to see that many of the assignments are repeated, so there are algorithms that check fewer assignments and are considerably faster than the “naive” algorithm. But there is no known algorithm that takes polynomial time with respect to \( n \), e.g., \( O(n^k) \) for some fixed constant \( k \). Table 1 lists the actual and estimated times taken by the “naive” \( O(n^3) \) time algorithm for the CM\(_3\) problem. For \( n = 33 \) this algorithm takes about 30 years to compute the answer to a no-instance. An “improved” algorithm can be made to run much faster, but for \( n \) around 50 it would take more than a century to solve a no-instance. Even on a machine that is 10000 times faster the algorithm would take 1% of a year (about 3.65 days). But then for \( n \) about 70 it would take a century. For \( n \) equal to about 5000, it would take enormous hardware improvements to speed-up the running time of the “improved” algorithm so that it takes only one century to solve a no-instance.

The CM\(_2\) problem is defined as the CM\(_3\) problem, except that the problem is to partition the set of people into two meetings such that all the people assigned to each meeting are compatible. The CM\(_2\) problem can be solved in polynomial time even though it is very similar to the CM\(_3\) problem and there are \( 2^n \) possible assignments of individuals to the two meetings. The simplest polynomial time algorithm for this problem is based on depth-first

---

\(^1\)This problem can be viewed as the problem of coloring the vertices of a graph with at most three colors. Each person is represented by a vertex in a graph which is called the incompatibility graph. The incompatibility between two individuals in this graph is represented by an edge joining the two corresponding vertices. The problem is to color the vertices with one of three colors so that no two adjacent vertices are assigned the same color. A 3-coloring for the incompatibility graph corresponds to the meetings when the colors are viewed as labels for the meetings.
Table 1: Running times for solving the CM3 problem via the “naïve” algorithm that enumerates all assignments of people to meetings. The values less than an hour are actual times and the ones larger than 1 hour are estimated.

<table>
<thead>
<tr>
<th>n</th>
<th>18</th>
<th>21</th>
<th>24</th>
<th>27</th>
<th>30</th>
<th>33</th>
<th>36</th>
<th>39</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes-instance</td>
<td>.06s</td>
<td>.6s</td>
<td>5.7s</td>
<td>56.8s</td>
<td>9.4m</td>
<td>1.6h</td>
<td>16h</td>
<td>6.7d</td>
</tr>
<tr>
<td>no-instance</td>
<td>1.5m</td>
<td>47m</td>
<td>23.5h</td>
<td>11.8d</td>
<td>1y</td>
<td>30y</td>
<td>9c</td>
<td>270c</td>
</tr>
</tbody>
</table>

s: seconds, m: minutes, h: hours, d: days, y: years, and c: centuries.
On the other hand, if for one of these problems one can show it cannot be solved in polynomial time, then none of the NP-complete problems can be solved in polynomial time. Let us now define this notion in a more formal setting.

The class of problems $P$ (easy to solve) are all decision problems that can be solved in polynomial time. The CM$_2$ problem falls into this category ($P$). The class of problems $NP$ (easy to check) are all decision problems for which yes-answers can be verified in polynomial time. In other words, each problem $S$ in $NP$ has a verification algorithm $A$ and the property that for every instance $I$ of $S$ there is a “fixed” structure $F(I)$ such that algorithm $A$, operating in polynomial time with respect to the length of $I$, can determine whether or not the structure $F(I)$ “proves” that the instance $I$ is a yes-instance. The structure $F(I)$ must be able to represent at least one feasible solution for each yes-instance $I$. The CM$_3$ and CM$_2$ problems defined above are clearly in $NP$. But the CM$_3$ problem is not known to be in $P$. In other words, for the CM$_3$ problem one can verify a yes-answer in polynomial time, but it is not known whether it can be solved in polynomial time.

Clearly, every problem that can be solved in polynomial time can be verified in polynomial time, i.e., every problem in $P$ is in $NP$ (e.g., the CM$_2$ problem). But it is not known whether the converse is true. Therefore, $P \subseteq NP$. The questions is whether or not $P = NP$. This is an important question. Now suppose that $P = NP$ and we have a polynomial time algorithm to solve an NP-Complete problem. Then every problem for which a yes-answer can be verified in polynomial time can be solved in polynomial time. Furthermore, the theory of NP-completeness will provide us with one such algorithm from a polynomial time verification algorithm. This would simplify programming considerably because automatically one will generate an efficient algorithm to solve a problem from an efficient algorithm to verify a yes-answer. This seems to good to be true, and the conjecture is that $P \neq NP$. I.e., there are some problems in $NP$ that cannot be solved in polynomial time. The next question is whether or not the CM$_3$ problem is one of this computationally intractable problems (every algorithm to solve this problem takes exponential intractable time with respect to the input length).

The class of NP-Complete problems is the set of “hardest” problems in $NP$. A problem $Q$ in $NP$ is said to be among the hardest in $NP$ if a polynomial time algorithm for $Q$ implies there is a polynomial time algorithm to solve every problem in $NP$. On the other hand, if one can show that an NP-complete problem cannot be solved in polynomial time, then clearly $P \neq NP$. The conjecture is that $P \neq NP$. The $P$ versus $NP$ question has been a central problem in CS for several decades and it has been referred to as the most important open problem in CS. This problem is also one of the seven
so called “millenium problems” identified by the Clay Mathematics Institute CTI. To stimulate research this institute has offered a million dollar award to any person or group of people who solve any of these millenium problems. Though, a “really fast” algorithm to solve NP-Complete problems is worth considerably more than that.

The $CM_3$ problem defined above is an NP-complete problem. So if $P \neq NP$, then there does not exist a polynomial time algorithm to solve the $CM_3$ problem. On the other hand, if there is a polynomial time algorithm to solve the $CM_3$ problem, then there is a polynomial time algorithm to solve all the problems in $NP$. The $CM_3$ problem, and every NP-Complete problem, is well defined and all its input parameters are known even before attempting to solve any problem instance. The output for every problem instance is simply yes or no. However, there does not seem to be any way, or we do not know how, to solve these problems efficiently.

Another important issue is the meaning of computational tractability. One normally equates an algorithm that takes polynomial time with an “efficient” algorithm. But this is not really true in the sense that an algorithm that takes $O(n^{100})$, where $n$ is the input size, is not really an efficient one. The interesting point is that there is no known “efficient” algorithm to solve any NP-Complete problem even under this very relaxed notion of “efficiency”. Tractability has been equated to “efficiently solvable”. The non-existence of such algorithms has been equated to computational intractability. NP-Complete problems are said to be intractable problems since there is no known polynomial time algorithm for their solution and it is likely none exists (i.e., $P \neq NP$).

The key component in the theory of NP-Completeness is that of polynomial transformation (or reduction). This notion had been used before the 1970s. After Cook’s Theorem, to establish that a problem $Q$ is NP-complete one needs to show that $Q$ is in $NP$ and that there is an NP-Complete problem that polynomially transforms to problem $Q$. A transformation from problem $Q_1$ to problem $Q_2$ is an algorithm that for every instance $I$ of $Q_1$ constructs an instance $f(I)$ of $Q_2$ in such a way that $f(I)$ is a yes-instance of $Q_2$ if, and only if, $I$ is a yes-instance of $Q_2$. A transformation is polynomial if the algorithm takes polynomial time with respect to the input size (i.e., $|I|$).

The immediate implication of a polynomial time transformation from a problem $Q_1$ to a problem $Q_2$ is that any algorithm that solves problem $Q_2$ can be used to solve $Q_1$. The time complexity of the resulting algorithm is the time required to perform the transformation plus the time required to solve the resulting problem instance. On the other hand, if one can show that problem $Q_1$ cannot be solved in polynomial time, then $Q_2$ cannot be solved in polynomial time. Suppose we could polynomially transform the
CM₃ problem to the CM₂ problem, then we would have a polynomial time algorithm to solve the CM₃ problem. The polynomial time algorithm is given an instance \( I \) of the CM₃ problem and constructs in polynomial time an instance of the CM₂ problem which is then solved by a polynomial time algorithms for the CM₂ problem. The answer to this latter problem is the solution to the original problem. But remember that there is no known polynomial transformation from CM₃ to the CM₂ problem.

On the other hand, it is very simple to polynomially transform the CM₂ problem to the CM₃ problem. The idea is to add a new person, that is incompatible with all the other people. Clearly, if the resulting problem instance is a yes-instance of the CM₃ problem if and only if the original instance of the CM₂ problem is a yes-instance. However, this polynomial time reduction does not add any new information because we are reducing an “easy” problem to a problem that may be “easy” or “hard”. So a polynomial time reduction from a problem \( Q₁ \) to a problem \( Q₂ \) does not tell us anything new about \( Q₂ \) when \( Q₁ \) can be solved in polynomial time. Also, if \( Q₂ \) cannot be solved in polynomial time, there is again no implication for \( Q₁ \). In this case, we are transforming an “easy” or “hard” problem to solve, to a problem that is “hard” to solve. Clearly, polynomial transformations are not symmetric. However, if one polynomially transforms \( Q₁ \) to \( Q₂ \) and \( Q₂ \) to \( Q₁ \), then both \( Q₁ \) and \( Q₂ \) are computationally equivalent problems in the sense that \( Q₁ \) is in \( P \) if and only if \( Q₂ \) is in \( P \).

During the 1970’s the number of known NP-Complete problems grew tremendously as researchers showed that hundreds of problems were NP-Complete. These problems included the following traditional problems: traveling salesperson, quadratic assignment, knapsack problem, timetable problems, bin packing, graph colorability, independent set, vertex arrangements, generalized spanning trees, Steiner trees, graph partitioning, generalized flow problems, set cover, partition, scheduling parallel machines, (open, flow and job) shop scheduling, inequivalence of finite automata, dynamic storage allocation, code generation, inequivalence of programs with arrays, logic problems, etc. During the 1980’s there was growth in the number of NP-Complete problems arising from emerging application areas. The number of NP-Complete problems has been increasing steadily since then.

Because of the many new applications, there are quite a few new problems being identified every day. Efficient exact and approximate solutions to these problems make these applications run smoothly. NP-Completeness plays an important role in this process, by identifying which problems can and cannot be currently solved efficiently. Due to the extensive literature on NP-Complete problems, publication of such results is becoming more difficult with the passage of time. Because of this many authors do not even attempt...
to publish such results or just publish a small summary of the result. In some cases authors just indicate that it is very likely that a given problem is NP-Complete. Sometimes this is because a simple reduction and its correctness proof do not seem possible. The aim of this handbook is to compile a large number of reductions so that the readers can identify the different transformation patterns which can then be used to simplify the process of establishing simple polynomial time transformations.

An important research aspect is to identify the simplest versions (with respect to problem parameters) of a problem that remain NP-Complete. It is interesting to note that as we vary problem parameters the problem jumps rather quickly from tractable to intractable (under the assumption that $P \neq NP$). This may provide us with some insight as to what makes a problem computationally intractable. For example, the CM$_2$ problem is in $P$, but the CM$_3$ problem is NP-Complete. Increasing the number of meetings from two to three (in the problem) makes the problem intractable. Many problems exhibit similar characteristics. Another reason for identifying “easy” and “hard” subproblems is that there are many approximation algorithms based on restriction. This means that an optimal or suboptimal solution to a restricted version of a problem may be used as a suboptimal solution for the original problem. NP-Completeness helps us identify the difficulty of solving restricted versions of problems and thus guide us through the process of designing provably good approximation algorithms. Approximation algorithms, metaheuristics, and randomized algorithms have been used with varying degrees of success to generate optimal and suboptimal solutions quickly for instances of many optimization problems whose corresponding decision problems are NP-Complete. However, none of these algorithms can generate in polynomial time an optimal solution all of the time. But for many problem instances they may provide optimal solutions quickly, and for all problem instances they may provide good solutions quickly.

In principle one may transform any NP-Complete problem to any other NP-complete problem. However, some of the transformations are more natural or simpler than others. All the known polynomial transformations follow a set of patterns. In the first portion of the handbook we will provide examples of the basic transformations. For example SAT (satisfiability) is a common source for reducibility. Other good sources are partition and 3-partition for scheduling problems; planar-SAT and planar vertex cover for planar, 2-dimensional and geometry problems; etc. The latter part of the handbook will include chapters for different application areas. The main idea is to summarize the NP-Completeness results in research areas and present the basic techniques used to establish NP-Completeness results for problems in different application areas.
A good portion of the NP-Complete problems have only theoretical value due to the limited, or non-existent, direct applicability for the problem at hand. However, even these results are important because such problems may be used to show that other problems arising from real-life applications are also NP-Complete. Our goal for this handbook is to collect a wide portion of the NP-Completeness results in as many areas as possible, as well as to introduce and explain in detail the different approaches used to show that these problems are NP-Complete. There will also be a chapter discussing the most important open problems in established as well as in emerging application areas.

2 Features

This handbook will be a collection of a wide range of information about NP-Completeness for problems in many different research areas. This volume will be partitioned into two parts: theory and applications. In order to facilitate the access to the handbook there will be a roadmap to the handbook in the introduction. This chapter will also include basic notation. It will also include basic NP-Completeness results as well as inapproximability results, and complexity classes beyond \( \text{NP} \). There will be chapters on strong NP-Complete problems, historical background and open problems.

We propose to include one or more chapters for application areas (or problems) including: packing (including bin-packing and its variants), traveling salesperson (and its variants), Steiner trees (and its variants), clustering, scheduling, partitioning, graph problems, bioinformatics, network routing (computer, optical, wireless or sensor), CAD, computational geometry, data bases and mining, Internet applications, etc.

3 Intended Audience

This handbook will be an excellent reference book for a large audience. Established researchers will be able to find in this handbook results in areas they have not had the opportunity to explore in the past. Graduate students as well as newcomers, who may or may not intend to pursue research in theoretical aspects of computation, will find an accessible body of work that will enable them to understand NP-Completeness at a fundamental level and will be able to use it as a reference when trying to establish NP-Completeness results for problems in computer science, computer engineering, operations research, applied mathematics as well as for problems arising in several other
disciplines. Increasing numbers of the mathematics, science and engineering graduate students and almost all the computer science, computer engineering and operations research graduate students will have to deal with NP-Completeness more than once in their professional careers. Our handbook will provide them with a strong starting point. To fully benefit from this handbook one needs a solid background in discrete mathematics and algorithms.

Our goal is to provide a comprehensive view of NP-Completeness, its theory and applications. This handbook will provide different approaches to establish NP-Complete results which have been successfully applied to several problems. The practitioners will be able to find different possible reductions that may apply to the problem at hand.

4 Market Information

The main book for NP-Completeness problems is the classical book by Garey and Johnson published in 1978. Over the years there have been 25 journal columns by Johnson which may be considered as extensions of the book. Almost all of these columns appeared before 1992. Every computational complexity and almost every algorithms textbook has one or two chapters on NP-Completeness. These chapters include basic results and simple transformations. The proposed handbook will have extensive information and details for problems in many application areas. Undergraduate and graduate CS (as well as some mathematics and operation research) courses have covered basic NP-Completeness results. As we said before, we are aiming this handbook at computer science, computer engineering and operations research graduate students as well as those in mathematics, science, and engineering. Advanced undergraduates will be able to understand the basic material. To fully benefit from this handbook students need a solid background in discrete mathematics and algorithms. The handbook will be a reference book for years to come.

Previous Books.

- Algorithms books like the one by Horowitz, Sahni and Rajasekaran (Computer Algorithms in C++, CS Press); Sedgewick (Algorithms); Kleinberg and Tardos (Algorithm Design); Dasgupta, Papadimitriou and Vazirani (Algorithms); and numerous other textbooks normally contain one or two chapters dedicated to NP-Completeness. This is perfect for undergraduate courses since NP-Completeness is just one of the course components.
5 Table of Contents (Tentative)

Each entry in the first part will be a chapter. The second part (applications) will includes several chapters for each application area/problem.

- Theory
  - Introduction / Notation / Handbook Roadmap - T. Gonzalez (UC Santa Barbara)
  - Historical perspective and recent advances
  - Reducibility and Cook’s Theorem
  - Basic Reductions
  - Typical Reductions
  - Strong NP-Completeness and Its Implications.
  - NP-hard problems
  - Planar and 2D Reductions
  - Efficient Transformations
  - Approximation classes
  - Inapproximability
  - Related Complexity Classes
  - Open Problems

- Applications
  - Packing
  - Routing (TSP, Vehicle, etc)
  - Trees (Steiner, spanning, etc.)
  - Scheduling
  - Graph Problems
    * Selecting
    * Partitioning
    * Arrangements
    * Embedding
    * Miscellaneous
  - SAT
– Flow Problems
– Bioinformatics
– Superstring and Supersequence Problems
– Message Routing (computer, optical, wireless and sensor)
– VLSI & Rectilinear Steiner Trees
– Image Processing
– Logic
– Computational Geometry
– Code Generation and Register Allocation
– Protocol and Security
– Data Bases, Query Languages, and Mining
– Internet
– Games and Puzzles
– Miscellaneous