Computers and Intractability
Micheal R. Garey
David S. Johnson

1. Computers, Complexity and Intractability.

1.1 Introduction. The primary application of the theory of NP-completeness is to assist algorithm designers in directing their problem solving efforts towards those approaches that have the greatest likelihood of leading to useful algorithms.

1.2 Problems, Algorithms and Complexity. *Algorithms* are general, step-by-step procedures for solving problems. *Efficiency* concentrates primarily on time as it usually dominates over other computing resources. The time requirements are expressed as the '*size*' of the problem referring to the amount of data required to specify the problem. The *time complexity function* for an algorithm expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size.

1.3 Polynomial Time Algorithms and Intractible Problems. An algorithm with time complexity which can be bounded by a *polynomial* (n^x) function (where n is the size of the problem) is said to have polynomial time complexity. Those that cannot be so bounded are called *exponential* (x^n) algorithms. Polynomial if in general far preferable though when n is small exponential can be faster and there are (rare) examples of exponential algorithms that are viable in practice. I problem is *intractable* if there is no polynomial time algorithm that can possibly solve it.

1.4 Provably Intractable Problems. A solution can be intractable because it necessitates an exponential algorithm or (less typically) because the solution itself is intractable. Turing demonstrated *undecidable* intractable problems in the 1930's and decidable intractable problems were identified in the 1970's. “All provably intractable problems known to date .... are either undecidable or *non-deterministically intractable*. However, most of the apparently intractable problems encountered in practice *are* decidable and *can* be solved in polynomial time with the aid of a nondeterministic computer. Thus none of the proof techniques developed so far is powerful enough to verify the apparent intractability of these problems.” pg 12

1.5 NP-Complete Problems. Stephen Cook (1971) laid the foundations for the theory of NP-completeness. Cook:

1. emphasized the significance of polynomial time reducibility. (that if there is a possible reduction from one problem to another known polynomial problem in polynomial time then there is a polynomial solution.)
2. He focused on the class NP of decision problems that can be solved in polynomial time by a non-deterministic computer. (a decision problem has a yes/no answer)
3. he proved that every NP problem can be polynomially reduced to the NP 'satisfiability′ problem. (Hence either all NP problems are either polynomial or intractable)

However, no-one has yet proven that NP-problems are intractable.

2. The Theory of NP-Completeness.

2.1 Decision Problems, Languages, and Encoding Schemes. The theory of NP-completeness is designed to be applied only to decision problems (yes/no answer) because it has a language which is suitable for study. Note, however, that so long as the cost function is relatively easy to evaluate, the decision problem can be no harder than the corresponding optimization problem. The correspondence between decision problems and languages is brought about by the encoding schemes we use for specifying problem instances whenever we intend to compute with them. However, the problems are independent of any particular encoding scheme and if we use a "reasonable" scheme the the polynomial nature of the problem is not impacted.

2.2 Deterministic Turing Machines and the Class P. A Deterministic Turing Machine DTM is defined:

Γ: finite set of tape symbols including subset $\Sigma \in \Gamma$ if input symbols and a blank symbol $b \in \Gamma - \Sigma$

Q: finite set of states including start-state q_0 and hold states q_Y and q_N.

δ: transition function.

A DTM program M solves and decision problem Π under encoding scheme if M halts for all input strings over the input alphabet and L_M=L[Π,e].

The time used in the computation of a DTM program M on an input x is the number of steps occurring in that computation up until the halt state is entered.

A decision problem Π belongs to P under the encoding scheme e if $L[\Pi, e] \in P$, that is, if there is a polynomial time DTM program that solves Π under encoding scheme e.

2.3 Non-Deterministic Computation and the Class NP. The Class NP is intended to isolate the notion of polynomial time "verifiability" - it does not imply polynomial time solvability. A non-deterministic is composed of two separate stages: a guessing stage and a checking stage (deterministic).

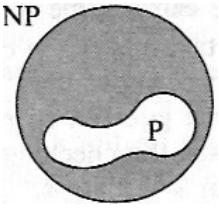
The Class NP is defined informally to be the class of all decision problems Π that, under reasonable encoding schemes, can be solved by polynomial time non-deterministic algorithms.

This is just a definition device to capture the notion of polynomial time verifiability rather than a realistic "solving" mechanism. Note that nondeterministic algorithms are not symmetric in the same way a DTM's.

A nondeterministic Turing Machine NDTM is defined identically to a DTM but with a guessing module used only at the start of the procedure.

2.4 The Relationship Between P and NP. $P \subseteq NP$ Every decision problem solvable by a polynomial time deterministic algorithm is also solvable by a nondeterministic algorithm.

Theorem 2.1 If $\Pi \subseteq NP$, then there exists a polynomial p such that Π can be solved by a deterministic algorithm having time complexity $O(2^{p(n)})$



There is widespread belief that P≠NP but it has not been proven.

2.5 Polynomial Transformations and NP-Completeness. A polynomial transformation from a language $L_1 \subseteq \Sigma_1^*$ to a language $L_2 \subseteq \Sigma_2^*$ is a function $f : \Sigma_1^* \to \Sigma_2^*$ that satisfies the following two conditions:

1. There is a polynomial time DTM program that computes f.
2. For all $x \in \Sigma_1^*$, $x \in L_1$ iff $f(x) \in L_2$

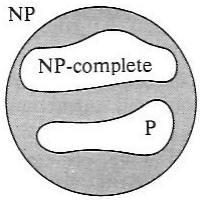
If there is a poynomial transform from L_1 to L_2, we write $L_1 \propto L_2$

Lemma 2.1 If $L_1 \propto L_2$ then $L_2 \in P$ implies $L_1 \in P$ (and equivalently, $L_1 \notin P$ implies $L_2 \notin P$)

Lemma 2.2 If $L_1 \propto L_2$ and $L_2 \propto L_3$ then If $L_1 \propto L_3$ (i.e. transitive)

The relation $\propto$ imposes a partial order on the resulting equivalence classes of decision problems. The class P forms the 'least' equivalence class under this partial order (ie easiest) and the class of NP-Complete form contain the 'hardest' decision problems in NP.

If any single NP-Complete problem can be solved in polynomial time then so to can all NP problems.



If any problem in NP-Compete is intractable, then so are all NP-Complete problems.

If P differs from NP, then there must exist problems in NP that are neither solvable in polynomial time nor NP-complete.

Lemma 2.3 If L_1 and L_2 belong to NP, L_1 is NP-Complete, and $L_1 \propto L_2$ then L_2 is NP-Complete.

To prove that Π is NP-Complete, we merely show that:

1. $\Pi \in NP$ and
2. some known NP-Complete problem Π' transforms to Π

2.6 Cooks Theorem. Let U={u_1,u_2,...u_m } be a set of boolean variables. A clause over U is a set of literals over U representing a disjunction of those literals and is satisfied by a truth assignment iff at least one of it's members is true. A collection C of clauses over U is satisfiable iff there exists some truth assignment for U that simultaneously satisfies all the clauses in C.

Theorem 2.1 (Cooks Theorem) SATISFIABILITY is NP Complete.

The proof proceeds by firstly realising that SAT is in NP because any guessed solution can be verified in polynomial time. The second step shows that a generic (polynomial time) transform can be created from any L into L_{SAT}. (i.e. very tricky)

3. Proving NP-Completeness Results.

A) Show that Π is in NP,
B) select a known NP-Complete problem Π',
C) construct a transformation f from Π' to Π, and
D) prove that f is a (polynomial) transformation.

3.1 Six Basic NP-Complete Problems. The following six problems were proved NP-Complete originally by Karp (1972) and have prove to be a popular and useful basis for equivalence proofs.

1. 3-SATISFYABILITY (3SAT)
2. 3-DIMENSIONAL MATCHING (3DM)
3. VERTEX COVER (VC)
4. CLIQUE
5. HAMILTONIAN CIRCUIT (HC)
6. PARTITION

The **3SAT** problem is just a restricted version of the SAT in which all instances have exactly three literals per clause. This restricted SAT is more useful for NP-Completeness proofs due to the simple structure.

A) Easy to see that 3SAT is verifiable in polynomial time.
B) SAT
C) Show how the arbitrarily long clauses in any SAT can be broken into collections of clauses with only 3 literals. There are fours cases for this procedure depending on how many literals are in each clause of the SAT problem: 1, 2, 3 or 3+.
D) Show that there is a polynomial procedure in each case. Case 1 & 2 are easy and there is nothing to do in case 3. Case 4 is a linear procedure of replacement.

3DM is a generalisation of the “marriage problem” but it arranges three-way matches acceptable to all three participants.

A) Easy to see that 3DM is verifiable in polynomial time.
B) 3SAT
C) Three disjoint sets W, X & Y are constructed of equal size. The set of ordered triples M must contain matches iff clause C in 3SAT is satisfyable. The set M is partitioned into three separate classes: “truth setting and fan-out”, “satisfaction testing”, or “garbage collection”. Each truth setting and fan-out component corresponds to a single variable and its structure depends on the total number of clauses in C. Each satisfaction testing component corresponds to a single clause in C. The construction is completed by one large “garbage collection” component G which guarantees that whenever a subset of M-G satisfies all of the constraints imposed by the truth setting and fan-out then that set can be extended to a matching

for M. The transformation can be completed in polynomial time.

D) .

VC and CLIQUE are really just different ways of looking at the same problem which is comparable in turn to INDEPENDENT SET.

A) It is easy the see that VC is verifiable in polynomial time.
B) 3SAT
C) d
D) c

HC

PARTITION

3.2 Some Techniques for Proving NP-Completeness. There are many ways to go about a proof but there are three general types of proofs that occur frequently and provide a framework for attempting a new NP-Completeness proof.

Restriction is the simplest approach which consists of showing that Π contains a known NP-Complete problem Π' as a special case and specifying the restrictions. Restriction focuses on the target problem and restricting away its 'inessential aspects' until a known NP-Complete problem appears. A extensive knowledge of known NP-complete problems becomes very valuable.

Local Replacement is also relatively simple and involves picking some aspect of a known NP-Complete problem to make up a collection of basic units in a uniform way, with a different structure. Each replacement must be a local modification of the structure which are essentially independent of one another (SAT -> 3SAT applied this approach). Sometime the replacement is purely based on the existing structure and sometimes additional structure (an enforcer) is required to provide certain additional restrictions. Enforcers can become complicated.

Component Design tends to be more complicated and involves using the constituents of the target problem instance to design certain 'components' that can be combined to 'realise' instances of the known NP problem. Two basic components are those the “make choices” and those that 'test properties'. These components are joined together in a target instance in such a way that the choices are communicated to the property testers, and the property testers then check whether the choices made satisfy the required restraints.

4. Using NP-Completeness to Analyze Problems.

Often it is not obvious that a problem is either polynomially solvable or NP-Complete. It is best to approach the problem from both ends as the insights are often instructive for either building a proof or an algorithm. Mapping the boundary of such problems leads to the introduction of a “pseudo-polynomial time algorithm” and “strong NP-completeness”.

4.1 Analyzing Subproblems. While an initial problem may be demonstrated to be NP-Complete, a subproblem obtained by placing additional restrictions on the allowed instances may have a different or more “natural” for the intended application. The frontier consists of the subproblems whose NP-Completeness is still an open question.

PRECEDENCE CONSTRAINED SCHEDULING is such a problem which is solvable in polynomial time when the prerequisite graph is either unlinked, a simple tree or has 3 or less parallel operations. The problem is NP-Complete for an arbitrary number o parallel operations but the situation with four or more parallel operations remains an open question.

Other examples	In P for D≤	NP-Complete for D≥
VERTEX COVER	2	3
HAMILTONIAN CIRCUIT	2	3
GRAPH 3-COLORABILITY	3	4
FEEDBACK VERTEX SET	2	3

The idea of “vertex substitution” is demonstrated in proving GRAPH 3-COLORABILITY. This involved designing a small graph G with a particular set of obvious properties and then substituting one vertex with a copy of G such that the properties are retained despite repeated substitution. The degree of ingenuity required may be substantial but it is worthwhile because the problems are common.

Another common restriction for graph problems is to planar graphs. There are two common approaches to a proof. The first is to use a planarity preserving transformation from another problem already known to be NP-Complete for planar graphs. The second, more basic technique is to use local replacement applied to the general problem, designing a “cross over” that can be used in place of any edge crossings that occur when a graph is embedded in the plane.

4.2 Number Problems and Strong NP-Completeness. Analyzing sub-problems is very important with problems involving numbers. There are problems that while are technically NP-Complete, only become intractable when the input becomes very large but remains tractable in

practical applications. Hence, the text explores subproblems obtained by placing restrictions on the magnitudes of the numbers occurring in the problem.

Two encoding dependent functions are required length and Max. Length is intended to map any instance I to an integer Length[I] that corresponds to the number of symbols used to describe I (under some reasonable encoding scheme). Max is intended to map any instance I to an Integer Max[I] that corresponds to the largest number in I.

Two Length functions are said to be polynomially related if there exists polynomials p and p' such that:

$$\text{Length}[I] \le p'(\text{Length}'[I]) \text{ and}$$
$$\text{Length}'[I] \le p(\text{Length}[I])$$

The pair of functions (Length, Max) are polynomially related to the functions (length', Max') if Lnght and length' are polynomially related and there exist two-variable polynomials q and q' such that:

$$\text{Max}[I] \le q'(\text{Max}'[I], \text{Length}'[I]) \text{ and}$$
$$\text{Max}'[I] \le q(\text{Max}[I], \text{Length}[I])$$

And there must be a polynomial time DTM to calculate Length and Max.

An algorithm that solves a problem Π will be called a ***pseudo-polynomial time algorithm*** for Π if its time complexity function is bounded above by a polynomial function of the two variables Length[I] and Max[I]

An NP-Completeness result does not necessarily rule out the possibility of solving Π with a pseudo-polynomial time algorithm.

A problem Π is a ***number problem*** if there exists no polynomial p such that Max[I] ≤ p(Length[I]) for all I in D_Π

If Π is NP-Complete and Π is not a number problem, then Π cannot be solved by a pseudo-polynomial time algorithm unless P=NP.

A decision problem Π is **NP-complete in the strong sense** if Π belongs to NP and there exists a polynomial p over the integers for which Π_p is NP-complete.

If Π is NP-complete in the strong sense, the Π cannot be solved by a pseudo-polynomial time algorithm unless P=NP.

The most straightforward way to prove that a number problem Π is NP-complete in the strong sense is simply to prove for some specific polynomial p that Π_p is NP-complete.

3-PARTITION is a number problem that is NP-complete in the strong sense and is more numeric than the other examples to date. 3-PARTITION is the 7th basic NP-complete problem.

3-PARTITION involves partitioning a set of items of various sizes into groups of three with a common group size. The proof proceeds in two steps: firstly establishing that the related 4-PARTITION is NP-complete in the strong sense.

A) Easy to see that 4-PARTITION is verifiable in polynomial time.
B) 3-DIMENSIONAL MATCHING
C) Show that 4-PARTITION is NP-complete even when restricted to instances with $\text{Max}[I] \le 2^{16} \cdot |A|^4$. The 4-PARTITION elements are setup to correspond to each member of a triple in the 3D MATCHING and another 4 PARTITION element for each triple in the 3D MATCHING. The size of these elements is defined (polynomials) in such a way that the bounds are met. Finally, the 4-PARTITION exists iff there is a matching
D) The transformation is a polynomial.

The second step establishes that 3-PARTITION is NP-complete in the strong sense.

A) Easy to see that 3-PARTITION is verifiable in polynomial time.
B) 4-PARTITION with $\text{Max}[I] \le 2^{16} \cdot |A|^4$.
C) The transformation involves a 3-PARTITION with 1 element for each element in the 4-PARTITION and 1 element for each pair of elements in the 4-PARTITION and filler elements. Again, the sizes of these elements are defined such that the requirements of the 3-PARTITION. and polynomial transformation are met.
D) The transformation is clearly polynomial.

It would be convenient if we could operate with transformations like this without needing to go into the details of the subproblems and the particular polynomials involved.

Let Π and Π' denote arbitrary decision problems with instance sets D_Π and $D_{\Pi'}$, "yes" sets Y_Π and $Y_{\Pi'}$, and specified functions Max, Length, Max' and Length', respectively. A pseudo-polynomial transformation from Π to Π' is a function $f: D_\Pi \text{-} D_{\Pi'}$ such that

(a) for all $I \epsilon D_\Pi$, $I \epsilon Y_\Pi$ iff $f(I)$ in $Y_{\Pi'}$
(b) f can be computed in time polynomial in the two variables Max[I] and Length[I],
(c) there exists a polynomial q_1 such that, for all $I \epsilon D^\Pi$, $q_1(\text{Length}'[f(I)]) \ge \text{Length}[I]$
(d) there exists a two-variable polynomial q_2 such that, for all $I \epsilon D^\Pi$, $\text{Max}'[f(I)] \le q_2(\text{Max}[I], \text{Length}[I])$

This generalised approach is not as complicated as it seems. Conditions (a) thru (c) are not substantially

new. The heart of the definition lies in condition (d) which ensures that the largest number is not exponential in terms of Length and Max.

4.3 Time Complexity as a Function of Natural Parameters. There are a variety of ways in which the time complexity of algorithms can be "exponential", some of which might be preferable to others when the practical aspects of the real world problem are applied a bounds.

5. NP-Hardness.

5.1 Turing Reducibility and NP-Hardness Problems. Any decision problem, whether a member of NP or not, or search problem, to which we can transform an NP complete problem is said to be **NP-hard**, since it is, in a sense, at least as hard as the NP-complete problems.

A **search problem** Π consists of a set D_Π of finite objects called instances and for each instance $I \epsilon D_\Pi$, a set $S_\Pi[I]$ of finite objects called solutions for I. An algorithm is said to solve a search problem Π if, given as input any instance $I \epsilon D_\Pi$, it returns the answer "no" whenever $S_\Pi[I]$ is empty and otherwise returns some solutions s belonging to $S_\Pi[I]$.

The formal counterpart of a search problem is a string relation and the correspondence between the two is accomplished by means of encoding schemes.

A **polynomial time Turing reduction** from a search problem Π to a search problem Π' is an algorithm A that solves Π by using a hypothetical subroutine S for solving Π' such that, if S were a polynomial time algorithm for Π', then A would be a polynomial time algorithm for Π.

An oracle Turing machine (OTM) consists of a standard DTM augmented with an additional oracle tape and two additional distinguished symbols: oracle-consultation state q_c, and resume computation state q_r.

A search problem Π is **NP-hard** if there exists some NP-complete problem Π' that Turing-reduces to Π.

- All NP-complete problems are HP-hard.
- The complement (i.e. with yes/no reversed) of any NP-complete or NP-hard problem Π_c must also be NP-hard.
- An NP-completeness result for the decision problem can be translated into an NP-hardness problem result for the search problem.

We can use the notion of NP-hardness for analyzing the complexity of problems in much the same way as we use NP-completeness. All the types of questions discussed in Chapter 4 are also applicable to NP-hard problems, and we can proceed to consider the complexity of subproblems and such related issues as pseudo-polynomial time algorithms and "strong" NP-hardness (defined analogously to strong NP-completeness, with a search problem being NP-hard in the strong sense if it contains an NP-hard subproblem satisfying a polynomial bound on Max [I]). The only limitation on such results is that, whereas an NP-complete problem can be said

to be solvable in polynomial time if and only if P = NP, all we can say with certainty about an NP-hard problem is that it cannot be solved in polynomial time unless P = NP.

In the case of the Travelling Salesman problem; the decision problem is Turing reducible to the optimisation problem and the optimisation problem is truing redudible to the decision problem.

A search problem Π is **NP-easy** whenever there exists a problem Π'ϵNP for which $\Pi \propto_T \Pi'$. That is, an NP-easy search problem can be solvd in polynomial time if P=NP.

The restriction of the basic theory to decision problems has caused no substantial loss in generality, since most often the search problems whose decision problem counterparts have been proved to be NP-complete are themselves NP-easy and hence of equivalent complexity. The growing class of NP-complete problems is augmented by a much larger class of equivalent search problems: those that are both HP-hard and NP-easy – and which might be called **NP-equivalent**.

5.2 A Terminological History.

- Cobham (1964) polynomial ≈ P
- Edmonds (1965) good algorithm ≈ P
- Edmonds (1965) good characterisation ≈ NP
- Cook (1971) introduced P and NP
- Cook (1971) P-reducability = polynomial time turing reductions (not polynomial transformations) – Cook reducability
- Cook (1971) SATIFYABILITY was not restricted to NP-Complete as it included HP-equivalent
- Karp (1974) reducability = polynomial transformability = Karp reducability or many-one reducability.
- Karp (1974) polynomially complete problem = NP-complete
- Cook-Karp class is a misnomer taken to mean Karp class
- Sahni (1974) P-complete = polynomially complete.
- Sahni (1974) P-hard
- Levin (1973) universal sequential search problem ≈ Sahni's P-complete
- Knuth formalised current usage for NP-complete, NP-hard and polynomial transformation.

6 Coping with NP-Complete Problems. One approach is to seek algorithms that, while achnowleging exponential ime conplexity, do better than a naive exhaustive search. These might avoid regions which cannot contain a solution or build on partial solutions (dynamic programming) amongst other approaches. A second approach, pertaining solely to optimisation problems, attempts a good solution in an acceptable amount of time (heuristics), but the approaches are often domain specific. In some cases it is possible to prove that solutions found by heuristic algorithms will never differ from optimal by more than a specific amount.

6.1 Performance Guarentees for Approximation Algorithms.

A **combinatorial optimisation problem** Π is either a minimisation pr a maximisation problem and consists of the following three parts:

1. a set D_Π of instances,
2. for each instance I ϵ D_Π, a finite set $S_\Pi[I]$ of candidate solutions for I; and
3. a function m_Π that assignes to each instance IϵD$_\Pi$, and each candidate solution σ ϵ $S_\Pi[I]$ a positive rational number $m_\Pi(I, \sigma)$, called the solution value for σ.

$OPT_\Pi(I)$ will be used to denote the value of $m_\Pi(I, \sigma)$ for the **optimal solution**. A(I) will be used to denote an **approximation algorithm** which returns a solution.

For example the bin packing problem can be solved with various approximation algorithms:

First Fit $\frac{17}{10}OPT(I)+1 \le FF(I) \le \frac{17}{10}OPT(I)+2$

First Fit Decendin $\frac{11}{9}OPT(I) \le FF(I) \le \frac{11}{9}OPT(I)+4$

The **absolute performance ratio** for minimisation problems and **asymptotic performance ratio** for maximisation problems are defined as:

$$R_A = \frac{A(I)}{OPT(I)} \qquad R_A^\infty(I) = \frac{OPT(I)}{A(I)}$$

A lower ratio (closer to 1) indicates better performance. Note that the ratios need not be equal.

For example consider the Travelling Salesman problem that obeys the triangle inequality (ie EITHER every direct path between cities is the shortest OR multiple visits are allowed).

Nearest Neighbour

$$\frac{1}{3}(\log_2(m+1)+\frac{4}{3})OPT(I) \le NN(I) \le \frac{1}{2}([\log_2]+1)OPT(I)$$

Twice round the min span tree $MST(I)<2\,OPT(I)$

Minimum matching $MM(I)<\frac{3}{2}OPT(I)$

Unfortunately there are a number of other NP-hard

problems for which no polynomial time approximation algorithms that perform even this well have yet been found. For example the graph coloring problem $A(G) \leq \frac{c|\dot{V}|}{\log|V|} OPT(G)$ appears to be even more difficult than either the bin packing or traveling sales person with triangl inequality. Finding a maximum set of independent vertices and the travelling salesman problem (without the triangle inequality) appear similarly hard in this respect.

A good approximation solution for one problem cannot in general be used to identify a good approximation solution for another problem with related polynomial transformations. Transformations do preserve optimal solutions but they do not preserve the ratio between the values of the optimal and sub-optimal solutions.

An **approximation scheme** for an optimisation problem Π is an algorithm A that takes as input both and instance I ϵ D_Π and and "accuracy measurement" ϵ>0, and that then outputs a candidate solution σ ϵ S_Π[I] such that $R_{A_\epsilon} \leq 1+\epsilon$. The term scheme is used because A provides a range of approximation algorithms.

A is a **polynomial time approximation** if for each fixed ϵ>0 the derived approximation algorithm A_ϵ is a polymomial time algorithm. A is a **fully polynomial time approximation scheme** if the time complexity of A itself is bounded by a polynomial function of Length[I} and 1/ϵ.

6.2 Applying NP-Completeness to Approximation Algorithms. The observed differences in approximability among NP-hard problems are inherent.

The best that one might hope for is |A(I)-OPT(I)| ≤ K for a fixed constant K. However this can be ruled out for the KNAPSACK and MAXIMUM INDEPENDENT SET problems by assuming that there is an approximation algorithm and then multiplying all instances (and hence solutions) by K+1. This implys that the approximation algorithm is in fact a optimization algorithm.

If there exists a two variable polynomial q such that for all instances I ϵ D_Π, OPT(I) < q(Length[I], Max[I]) then the existance of a fully polynomial time approximation algorithm for Π implies the existance of a pseudo-polynomial time optimisation algorithm. Further more, if Π is an interger valued optimisation problemand HP-hard in the strong sense, then Π cannot be solved by a fully polynomial time approximation algorithm unless P=NP.

This approach holds for many problems of interest.

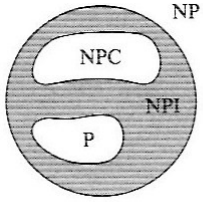
It is a bit more difficult to rule out approximation schemes that are polynomial but not fully polynomial.

Techniques to prove lower bounds are considerable more limited in power.

6.3 Performance Guarantes and Behaviour "In Practice". Approximation algorithms often behave significantly better in practice than their guarantees would suggest. However, choosinging typical test problems for emperical testing in troublesome.

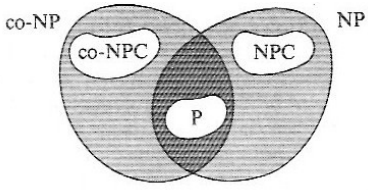
7. Beyond NP Completeness.

7.1 The structure of NP. There must be problems NPI=NP-(P U NP-Complete) and open problems can be viewed as candidates. In particular GRAPH ISOMORPHISM, COMPOSITE NUMBERS and LINEAR PROGRAMMING. (Note LP was proved polynomial in 1979 after this book was published)



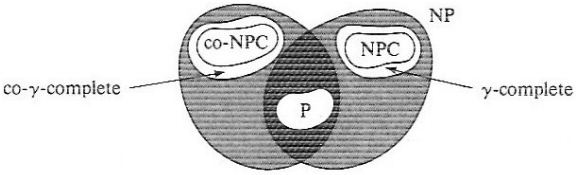
The world of NP, reprised (assuming P≠NP)

Many problems in co-NP (complementary problems have their answers reversed) do not seem to be in NP so it can be conjectured that NP≠co-NP. Neither COMPOSITE NUMBERS or LINEAR PROGRAMMING can be NP-complete unless NP=co-NP.



The world of NP and vicinity (assuming P≠NP and NP≠ co-NP). It may or may not be the case that P = NP∩ co-NP.

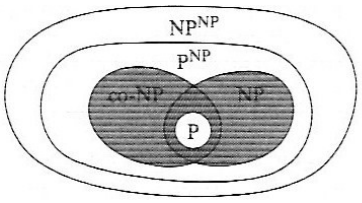
ɣ-reducability provides a way of proving that a problem is intractable under the assumption that NP≠co-NP.



7.3 The world of NP, once more revised (assuming that both P≠NP and NP≠ co-NP).

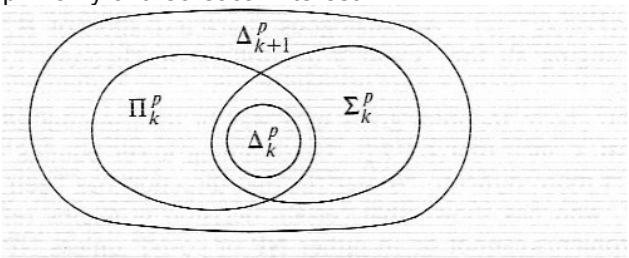
7.2 The polynomial heirachy. The difficulty of NP-hard problems can be arranged in a heirachy.

MINIMUM EQUIVALENT EXPRESSION is a problem which is NP-hard but apparently not NP-easy as it has not been shown that it can be solved in polynomial time with an oracle machine. However MEE can be solved in polynomial time with a nondeterministic oracle machine (NOTM). An NOTM with an oracle for problem Π corresponds to a nondeterministic algorithm with a subrouting for Π. Note that P^{NP}=NP-easy and NP^{NP} contains MEE.



Containment relationships between classes of languages, including the new classes P^{NP} and NP^{NP} (assuming P≠NP).

Finally it is observed that the process of defining new classes on the bases of old ones could be extended indefinitely. However this appears to be primarily of theoretical interest.



Containment relationships within the polynomial hierarchy

7.3 The Complexity of Enumeration Problems. Enumaration problems are natural candidates for being intractable even if P=NP. These problems ask how many solutions there are to an instance in contrast to a search problem which asks for a solution.

An enumeration problem Π beongs to #P if there is a nondeterministic algorithm such that for each I ϵ $D_Π$, the number of distinct guesses that lead to acceptance of I is exactly $|S_{Π,}[I]|$ and such that the length of the longest accepting computation is bounded by a polynomial in Length[I].

#P contains at last a large fraction of the enumeration problems one might wish to consider and certainle many apparently difficult ones. The notion of #P-completeness for #P is once again used to capture the notion of a "hardest" problem in the class.

There is a variation on polynomial transformability called parsimonious transformation (which contains the former) and automatically gives rise to a Turing reduction between the associated enumeratino problems.

Some (but not all) enumeration problems are #P-complete even when the associated search problem can be solved in polynomial time.

7.4 Polynomial Space Completedness. Any problem solvable in polynomial time can necessarily be represented in polynomial space.

The corresponsing notions of PSPACE, and

PSPACE-complete can be defined.

The fact that a problem id PSPACE-complete is an even stronger indication that it is intractable than if it were NP-complete; we could have P=NP even if P≠PSPACE. It is also considered evidence that the problem is not in NP or even in the polynomial heirachy.

Combinatorial Games have been a particularly rich source of PSPACE problems.

NPSPACE can be defained and is perhaps surprisingly equal to PSPACE.

Theorem 7.12 If L can be recognized by an NDTM program in space bounded by T(n), where T(n) ≥ log n for all n ≥ 1, then L can be recognized by a DTM program in space bounded by $T^2(n)$.

The most famous open problem in complexity theory before the P vs. NP question arose is known as "the LBA problem. "Let us call a DTM program that obeys a space bound of n + 1 a deterministic linear bounded automaton (DLBA) and an NDTM program obeying the same bound a nondeterministic linear bounded automaton (NLBA) . The set of languages recognizable by NLBAs is precisely the set of "context sensitive languages". By Theorem 7.12 all such languages can be recognized by DTM programs obeying a space bound of $(n+1)^2$. The question is, can every such language be recognized by a DLBA? (Or, even more strongly, can the exponent of 2 be removed from Theorem 7.12?) This question involves the same issues of determinism vs. nondeterminism as does the P vs. NP question, albeit at a more detailed level, and it has remained open since it was first posed in the mid-1960's. For a thorough" discussion of this question, its consequences, see Hartmanis and Hunt 1974.

7.5 Logarithmic Space.

7.6 Proofs of Intractability and P vs. NP.

Further Reading

- Lin, S. [1975] "Heuristif programming as an aid to network design" Netorks 5, 33-43 (6.0)

[GP11] NxN GO (*)
INSTANCE: Positive integer N, a partition of the "points" on an NxN Go board into those that are empty, those that are occupied by White stones and those that are occupied by Black stones, and the name (Black or White) of the player whose turn it is.
QUESTION: Does White have a forced win from the given position in a game of Go played according to the standard rules, modified only to take into account the expanded board?
Reference: [Lichtenstein and Sipser, 1978]. Transformation from PLANAR GEOGRAPHY.
Comment: PSPACE-hard.

\