Complexity Theory and NP-Completeness

Alan W. Black
Coventry (Lanchester) Polytechnic

June 1984

This paper gives a general description of modern day complexity theory and NP-completeness. Within modern complexity theory problems can be classified as follows.

1. Unsolvable - such as halting problem, "busy beaver" problem etc.
2. (a) Tractable - solvable in sensible time
   (b) Intractable - solvable, but requires so much time /space that practically they are too difficult to solve

To look closer at the tractable and intractable problems, researchers looked at how the numbers of steps in an algorithms (or say a turing machine) varies with the size of the problem.

If an algorithm takes time $T(n)$ and there exists a polynomial $p(n)$ such that the maximum time $T(n) \propto p(n)$, then we say the algorithm has polynomial time complexity, and the algorithm is a polynomial time algorithm. If an algoritm is not a poly time algorithm then it may be something bigger e.g. exponential. That is a problem which takes time a^n where n is the size of the problem.

Exponential time functions are much harder (take more time) than poly time algorithms e.g. say each step takes 1 micro seconds on a computer when n=60, poly time algoritms (say cubic) 60^3 take about 0.216 seconds while in the exponential case 3^{60} is 1.3×10^{13} centuries (a very long time).

The boundary between poly time complexity and non-poly time complexity is thought to be the distinction betwen tractable and intractable problems.

We try to classify problems into distinct classes, two such classes are P and NP. If there is a poly time algoritm (a deterministic turing machine) that can solve a problem X, then X is in class P. If there exists poly time algorithm on a *non* deterministic machine that solves a problem X, then X is said to be in class NP (non-deterministically polynomial). **NOTE** a non-deterministic machine is a machine with *unlimited* parallelism.

An example of a problem in NP is the travelling salesman problem. This problem is, given a number of cities, and the distances between them, is there a route through them such that the total distance is less than some bound.

The way to solve this problem is to guess a route and then test its length against the required. On a non-dterministic machine we could guess all possabilities at once. The guessing and checking can be done in poly time, i.e. it is non-deterministically polynomial and hence is in class NP.

When we refer to problems we can can specify them as decision problems, i.e. problems that have either an answer yes or no.

Formally we can say a decision problem Π has a set of instances that give yes and a set that give no. An instances is some encoding (see later) of a problem in a language Σ. Strings in Σ fall into thre categories.

1. invalid encodings of the problem
2. valid encodings that give answer yes
3. valid encodings that give answer no

Strings of Σ in category b) are refered to as

$$L(\Pi, e)$$

Instances of Π, under some encoding that give the answer yes. A machine M is said to solve a decision problem Π if M halts for all inputs and

$$L(M) = L(\Pi, e)$$

The encoding of a problem is difficult to formalise as we have many different types of problem. All that need be said is that the encoding of the problem should be reasonable (i.e. can be done in poly time). Obviously it is "cheating" to specify that, in the travelling salesman problem, to order of specifying the cities must be such that it specifies the shortest route through them. The other point to mention here is the operation speed of the machine that is to be used to solve the problem. The speed of the machine can only increase the time required to solve the problem polynomially. That is, speeding the machine up will not make NP have less parallelism required, only the spped of the poly part.

Completeness

A problem is said to be complete, within a class, if all other problems in the class can be reduced to it. (by some poly time algorithm). Formally $L(C)$ is complete within class C iff

$\forall L(i) \in C$

$L(i) \propto L(C) \;\; (L(i)\ can\ transform\ to\ L(C)\ by\ \ some\ p\,t\,a)$

NP-Completeness

L is NP-complete iff

1. $L \in NP\ and,$
2. $if\ L' \in NP\ then\ L' \propto L$

Note that if there is an efficient algorithm to solve an NP-Complete problem then there is an efficient algorithm to solve all other problems in NP.

Cook (1971) showed that the satisfiability of a boolean expression is NP-complete. Given a collection of clauses C, each clause contains some number of terms u_i or $u_i' \in U$, variables within each clause are ORed, the result of each clause is ANDed; is there some assignment of values to each member of U that makes the expression true (i.e. satifiable). This is referred to as the SAT problem.

The travelling salesman is another NP-complete problem. There are some 100-200 currently known NP-Complete problems (see Computers and Intractability, Garey and Johnson; H. Freeman, San Francisco). At one time many of these problems were thought to be distinct. This means if an efficient algorithm is found for solving an NP-Complete problem there exists an efficient algorithm to solve all NP-Complete problems (by efficient it is meant polynomial (well probably)). The can be done by

1. convert a problem to an encoding of the solvable one.
2. solve it.

To show that 3-SAT is NP-complete (3-SAT is a special case of SAT where each clause contains three and only three variables). We know SAT is NP-complete so if we can convert any instance of SAT to 3-SAT, 3-SAT must be NP-complete, by definition.

We must convert each cluase of SAT to 3 members without changeing the solvability (i.e. have the same true interpretations). And show that the transformation can be done in a reasonable time.

In converting there are three cases

$$k < 3 \;\; k = 3 \;\; and \, k > 3$$

Where k is less then 3 (1 or 2 variables in a clause).

$k = 1$: $C_j = \{z\}$ we introduce two more variables
$U_j' = \{y_{j1}\ ,\ y_{j2}\}$
$C_j' = \{z\ ,\ y_{j1}\ ,\ y_{j2}\}\ \{z\ ,\ y_{1j}\ ,\ y_{j2}'\}\ \{z\ ,\ y_{j1}'\ ,\ y_{j2}\}\ \{z\ ,\ y_{j1}'\ ,\ y_{j2}'\}$
this does not change the satisfiability of C_j

$k = 2$: $C_j = \{z_1\ ,\ z_2\}$
same as above add one variable to give C_j' with two clauses

$k = 3$: null case

$k > 3$: $C_j = \{z_1\ ,\ z_2\ ,\ z_3\ ,\ \dots\ ,\ z_k\}$
we need to add $k - 3$ variables
$U_j' = \{y_{j1}\ ,\ y_{j2}\ ,\ \dots\ y_{jk-3}\}$
$C_j' = \{z_1\ ,\ z_2\ ,\ yi_{j1}\}\ \{y_{j1}'\ ,\ z_3\ ,\ y_{j2}\}\ \{y_{j2}'\ ,\ z_4\ ,\ y_{j3}\}$
...
$\{y_{jk-4}'\ ,\ z_{k-2}\ ,\ y_{jk-3}\}\ \{y_{jk-3}'\ ,\ z_{k-1}\ ,\ z_k\}$

The original clause C_j and the modified one C_j' will be true under the same assignments to the z's.

Finally we have to show that the transformation is not too hard, intuitively we can see this is true. The number of variables added is not exponentially related to the instance of SAT.

$$SAT \propto 3 - SAT$$

If $\Pi \propto$ SAT then we can say $\Pi \propto$ 3-SAT because $\propto$ is obviously transitive