The test generation problem: an NP-Complete problem?

Definition of an NP-Complete problem (Wikipedia)

In computational complexity theory, the complexity class NP-complete [1] [2], also known as NP-C or NPC, is a subset of NP ("non-deterministic polynomial time"); they are the most difficult problems in NP in the sense that a deterministic, polynomial-time solution to any NP-complete problem would provide a solution to every other problem in NP (and conversely, if any one of them provably lacks a deterministic polynomial-time solution, none of them has one). Problems in NP-complete are known as NP-complete problems.

A decision problem C is NP-complete if:

1. C is in NP, and
2. Every problem in NP is reducible to C.

C can be shown to be in NP by demonstrating that a candidate solution to C can be verified in polynomial time.

A problem L is reducible to C if there is a polynomial-time many-one reduction, a deterministic algorithm which transforms instances \( l \in L \) into instances \( c \in C \), such that the answer to \( c \) is YES if and only if the answer to \( l \) is YES. To prove that an NP problem A is in fact an NP-complete problem it is sufficient to show that an already known NP-complete problem reduces to A.

Note that a problem satisfying condition 2 is said to be NP-hard, whether or not it satisfies condition 1.

A consequence of this definition is that if we had a polynomial time algorithm for C, we could solve all problems in NP in polynomial time.

List of NP-Complete problems


Solving NP-Complete problems

At present, all known algorithms for NP-complete problems require time that is superpolynomial (for example, exponential) in the input size, and it is unknown whether there are any faster algorithms.

Many techniques can be applied to solve these types of problems. For instance:

- **Randomization**: Use randomness to get a faster average running time, and allow the algorithm to fail with some small probability.
- **Heuristic**: An algorithm that works "reasonably well" on many cases, but for which there is no proof that it is both always fast and always produces a good result. Metaheuristic approaches are often used.

**Fault Detection Problem: an “NP-Complete problem”**

In [4], the authors look at several variations of the single fault detection problem for combinational logic circuits and show that deciding whether single faults are detectable by input-output (I/O) experiments is polynomially complete, i.e., there is a polynomial time algorithm to decide if these single faults are detectable if and only if there is a polynomial time algorithm for problems such as the traveling salesman problem, knapsack problem, etc.

Recently [5], Fujiwara and Toida analyze the computational complexity of fault detection problems for combinational circuits. Although major fault detection problems have been known to be in general NP-complete, they were proven for rather complex circuits. The authors show that these are still NP-complete even for simple circuits.

**Our Test Generation Problem**

Our new software testing approach has been proposed to automatically verify that a software system conforms to its functional requirements [6] (therefore, free of bugs). Our approach is based on modelling the software functional requirements (A unique simulated model to represent these requirements has been developed [7]) and on generating automatically test cases from this model. To guide the selection of test cases, coverage criteria are used. A coverage criterion can be seen as a set of items (relations between inputs and outputs) in the requirements model to be covered [8]. Therefore, our test generation problem can be formulated as a reachability problem.

A Test Case is a set of ((TS1,C1), (TS2,C2), (TS3,C3), ... (TSn,Cn)), where TSi is a Test Step and Ci is the coverage contribution performed by TSi. Ideally, this set should be reduced so that the total coverage ΣCi is not changed, and the length of the test case, i.e., Σ|TSi| is minimized. The remaining Test Steps TSi can then be used as the test case. However and as it was shown below, selecting a subset of Test Steps with this property is an NP-Complete problem.

**Covering the Input/Output relationships of a software system: an “NP-Complete problem”**

In [9], Seroussi and Bshouty prove that the design of an optimal exhaustive test case for an arbitrary logic circuit is an NP-complete problem. In fact, they demonstrate that finding the minimal test case (and its size) which covers the logic circuit is an NP-complete problem. In order to do this, they first show that the problem can be solved by a nondeterministic algorithm in polynomial time. Then, they use the standard technique of reduction to prove that the problem is NP-complete: for a given problem Q (the graph coloring problem) known to be NP-complete, they show that if our problem is solvable in deterministic polynomial time, then so is Q.

In [10] and [11], the authors consider the problem that arises in software testing: “generating small test case where the combinations that have to be covered are specified by input-output
parameter relationships of a software system”. They prove that finding optimal test cases is an NP-Complete problem. Indeed, they reduce the test generation problem to the set-covering problem.
References


