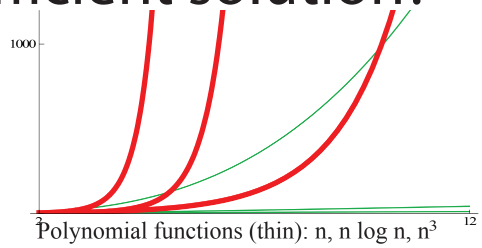
Can you tell if a problem will have an efficient solution?

If you can think of a polynomail time algorithm to solve the programming problem you are working on, then you know the problem is in P. But what if you can't think of one? Then it might be worth considering if the problem might be NP-Complete.



Polynomial functions (thin): n, n log n, n^3
Exponential function (thick): 2^n, n^n, n!

This graph illustrates why polynomial runtime is necessary for a solution to be practical. At small input size n = 10, the exponential algorithms are already slower than even than slow polynomial O(n). ³

Complexity Classes P, NP, and EXP

In theoretical Computer Science, problems are classified by existence of algorithms that solve them. We consider the issue of polynomial verses exponential runtime. Polynomial time algorithms (e.g. O(n) O(n log n) O(n3)) are the fastest. Exponential time algorithms are too slow to run on even moderate size data sets.

P

The set of all problems that have a polynomial time algorithm.

To prove a problem is in P you must write a polynomial time algorithm to solve it.

Almost every problem that computers are used for today has a polynomial runtime algorithm. It can be difficult to show that a problem is in P. When Computer Scientists find a efficient solution to a problem that previously was only efficiently verifiable, it shows that P was larger than we thought. If this was ever done for an NP-Complete problem, it would prove that P = NP.

Not every problem in NP is NP Complete. Factoring numbers in an example of a problem that has no known polynomial solution but no one has proved it is NP-Complete.

NP-Complete Problems

These are the hardest problems in NP. A solution to one of these problems would prove that P = NP. If P ≠ NP, these problems must all be in NP.

If you can prove your problem is NP-Complete, it suggests that your problem is likely to have no polynomial-time solution.

NP

The set of all problems that can be verified in polynomial time.

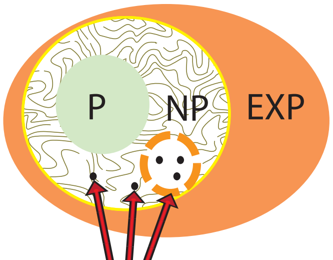
To prove a problem is in NP you must write an efficient algorithm to check a given solution.

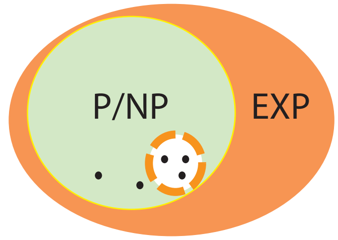
EXP

The set of problems that have an exponential time algorithm. To prove a problem is in set EXP you must develop an algorithm to solve it in exponential time (that is usually easy).

There are problems in EXP that cannot be solved in polynomial time. We know these problems are intractable. If P = NP, these are the only problems that are intractable. If P ≠ NP, some problems in NP are also intractable.

Does P = NP?





No polynomial-time solution?

The common wisdom is that all of the NP-Complete problems and some of the other problems in NP have no polynomial time solution. But this wisdom could be wrong.

It could be that that every problem in NP does in fact have a polynomial time solution. No one has ever proved that any polynomial time verifiable problems do not have a polynomial time solution.

This persistant question is known as the question of "Does P = NP?"

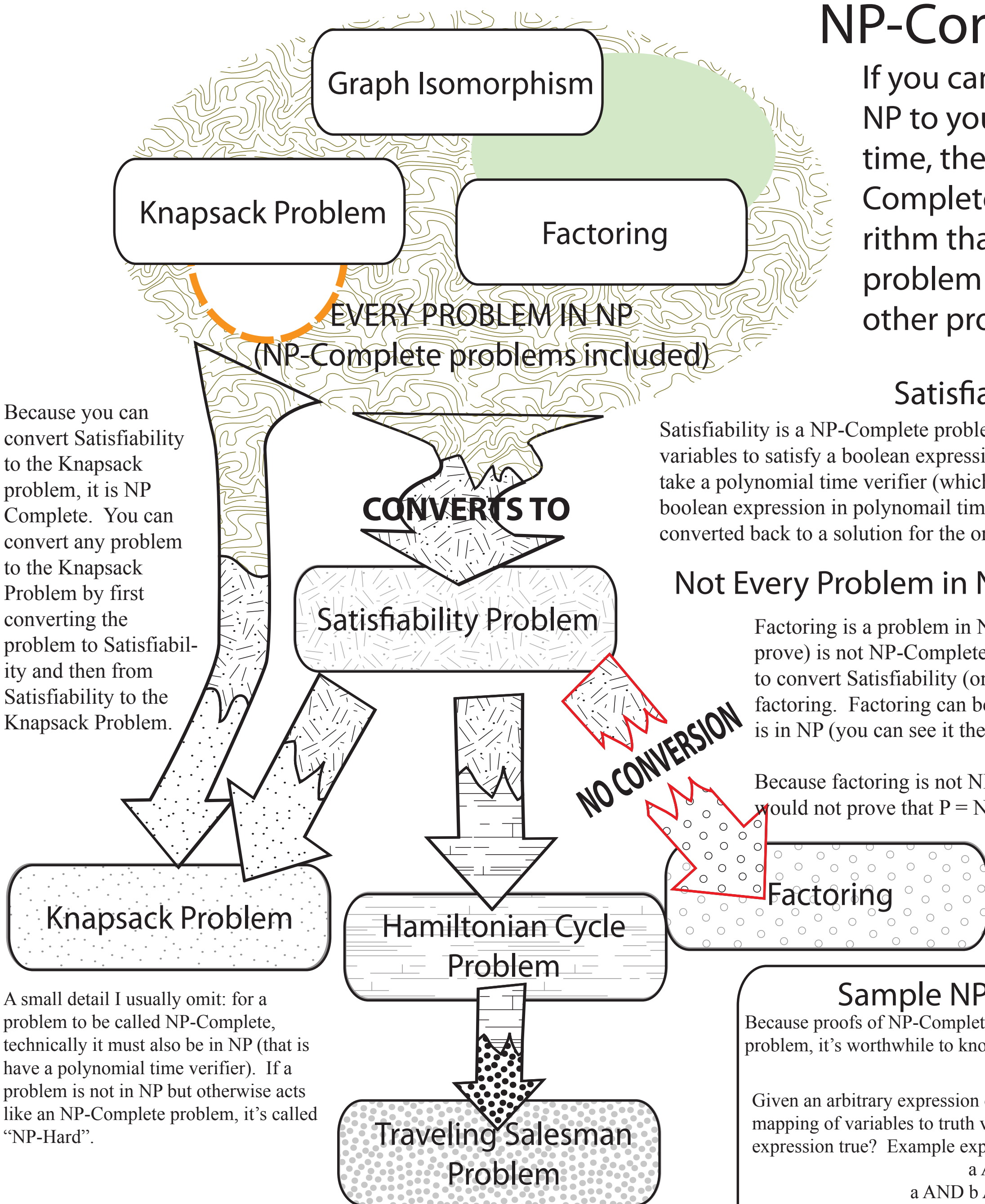
Key Questions

How do you prove a problem is in set P?
How do you prove a problem is in set NP?

What would you need to prove that P = NP?
What would you need to prove that P ≠ NP?

NP-Complete Problems

If you can convert every problem in NP to your problem in polynomial time, then your problem is NP-Complete. A polynomial time algorithm that solved an NP-Complete problem would also solve every other problem in NP.



Because you can convert Satisfiability to the Knapsack problem, it is NP Complete. You can convert any problem to the Knapsack Problem by first converting the problem to Satisfiability and then from Satisfiability to the Knapsack Problem.

A small detail I usually omit: for a problem to be called NP-Complete, technically it must also be in NP (that is have a polynomial time verifier). If a problem is not in NP but otherwise acts like an NP-Complete problem, it's called "NP-Hard".

Satisfiability

Satisfiability is a NP-Complete problem about finding mappings for true/false variables to satisfy a boolean expression. Cook's Theorm provides a algorithm to take a polynomial time verifier (which every NP problem has) and convert it to a boolean expression in polynomail time. The true/false mappings can then be converted back to a solution for the original problem in polynomial time.

Not Every Problem in NP is NP Complete

Factoring is a problem in NP that (as far has any one has been able to prove) is not NP-Complete. That means that no one has ever been able to convert Satisfiability (or any other NP-Complete problem) TO factoring. Factoring can be CONVERTED TO Satisfiability because it is in NP (you can see it there in the big oval at the top).

Because factoring is not NP-Complete, a polynomail time solution to it would not prove that P = NP.

If you can convert a NP-Complete problem TO your problem, your problem is NP Complete. This is the usual way you prove a problem is NP Complete.

It follows from the definition of NP-Completeness that any problem that an NP-Complete problem can be converted TO must be NP-Complete as well (see the text over the Knapsack problem for an example). Usually when you want to prove a problem is NP complete, you simply prove an existing known NP-Complete problem can be converted to it.

No one has ever found a polynomial time algorithm to solve an NP-Complete problem. For this reason, most Computer Scientists use "NP-Complete" as a shorthand for "no polynomial time solution".

If P ≠ NP, then no NP-Complete problem can be in P. This is not a proof that any particular NP-Complete problem has no polynomial time solution - if one is found it proves P = NP. But because no one has ever found an efficient solution to an NP-Complete problem, intution suggests that no solution may be possible. So if you can prove your problem is NP-Complete, it should be a warning that no efficient solution is likely.

Sample NP-Complete Problems

Because proofs of NP-Completeness involve converting a known NP-Complete problem, it's worthwhile to know a few.

Satisfiability

Given an arbitrary expression of variables NOTs ANDs and ORs. What is a mapping of variables to truth values (e.g. a → true, b → false) that makes the expression true? Example expresses:

a AND (b OR (NOT a))

a AND b AND ((NOT a) OR (NOT b))

Knapsack Problem

Given a set of items with different weights and values, what is the maximum value you can get in a "knapsack" with a maximum weight W?

Hamiltonian Cycle

Given a set of cities with roads between them, is there a path that visits each city exactly once and returns to the start?

Traveling Salesman Problem

Given a set of cities with roads between them, what is the path of minimum distance that visits each city exactly once and returns to the start?

Exam Scheduling Problem

Given a list of courses, a list of conflicts between them, and an integer k; is there an exam schedule consisting of k dates such that there are no conflicts between courses which have examinations on the same date?

Key Questions

If a solution were developed for an NP-Complete problem, what would that mean?

If you can convert an NP-Complete problem into your problem, what does that mean about your problem?