SE 3310b

Theoretical Foundations of Software Engineering

The Complexity Classes **NP-Hard**, **BPP**.

Aleksander Essex

The **NP-Hard** Class

The **NP-Hard** Class

Recall the **NP-complete** complexity class was defined as the set of problems that are:

- In **NP**, and,
- Evey problem in **NP** is polynomial time reducible to problems in this class.

The second property is interesting because it captures the idea that problems in this class are *hard to solve*.

Ok so what about this first condition? What if we just want to look at problems that are hard to solve, regardless of whether they are in **NP** or not?

The **NP-Hard** Class

So let’s consider the class of ”hard” problems, regardless of whether they’re in **NP** or not.

Definition 1 (**NP-Hard**).

A problem is **NP-Hard** if all problems in **NP** are polynomial time reducible to it.

Translation: **NP-Hard** is the set of problems that are *as hard or harder* than any**NP** problem.

Western Engineering

The **NP-Hard** Class

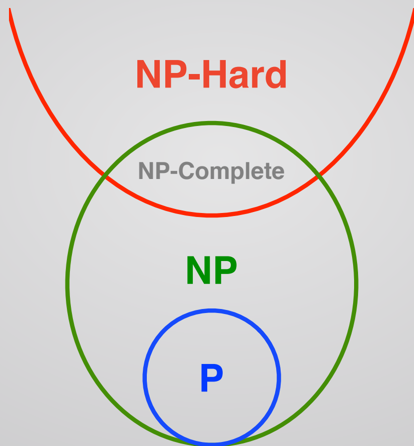
Let's examine a few consequences of this definition of the **NP-Hard** class:

- It contains the **NP-complete** class
- It includes problems that are not in **NP**
- It includes problems that aren't decision problems
- It even includes decision problems that are undecidable!

Western Engineering

The **NP-Hard** Class

Here's how **P**, **NP**, **NP-complete** and **NP-Hard** interrelate:



Western Engineering

Optimization Problems

NP-Hard contains **NP-complete** decision problems, but it's also well known for containing **optimization problems**, e.g., find the best/cheapest/fastest/smallest solution to a particular problem.

Here are just a few real-world situations where you might encounter hard optimization problems:

- Scheduling
- Data mining
- Selection
- Diagnosis
- Classification
- Process control
- Planning and resource allocation

Western Engineering

Example: The Traveling Salesman Problem

Problem: Given a list of cities and the distance between each city, find the shortest route that (a) visits each city once, and (b) returns to the starting point.

More generally: Given a wighted graph, find a Hamiltonian Cycle with the *lowest* weight.

Applications: Scheduling, route planning, etc

Traveling Salesman Problem

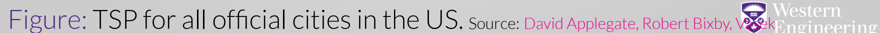


Figure: TSP for all official cities in the US. Source: David Applegate, Robert Bixby, Vašek Chvatal and William Cook

Example: Knapsack Problem

Problem: Given a knapsack with that can hold up to a certain weight, and a list of items, each with a weight and corresponding value, fill the knapsack with the highest value selection of items (without exceeding the knapsack's max load).

Applications: Resource allocation, financial planning, cryptography, etc.

Optimization Problems vs. **NP**

How do optimization problems relate to **NP**? Here's a natural question: is the Traveling Salesman problem in **NP-complete**? In it's basic form: no.

TSP ask to find the *best* solution, so its clearly not a yes/no decision problem, thus it's not in **NP**.

Ok so what if we re-stated it as a decision problem: here's a certificate (i.e., a claimed solution), is this the best? Is this in **NP**? For it to be in **NP** it must be checkable in polynomial time. But checking it's the best would mean being able to decide it it was less then all the other solutions.

Optimization Problems vs. **NP**

We can convert the basic TSP problem (which is **NP-Hard** but not **NP-complete**) into one that is **NP-Hard** *and* **NP-complete** as follows:

Given a weighted graph and a weight w, is there a Hamiltonian cycle below weight w?

The point is subtle, which is why you will see people refer to TSP as an **NP-complete** problem, even though it isn't *technically*.

Probabilistic Algorithms

Probabilistic Turing Machines

Recall a deterministic Turing machine (DTM) follows a single computational path whereas a non-deterministic Turing machine (NTM) can potentially follow many paths simultaneously. For *certain* types of problems (such as brute-force searches) NTMs provide a decisive performance advantage: they can explore exponentially more computational paths per unit time as DTMs (assuming, of course, $\mathbf{P} \neq \mathbf{NP}$).

Probabilistic Turing Machines

NTMs are a useful for helping us establish bounds on the hardness of various real-world problems. But as a real-world construct they have a major limitation: we can't build them (at least not to efficiently solve the really interesting problems)! But maybe there's a way we can borrow a page from the NTM playbook, so to speak.

Probabilistic Turing Machines

Recall for a problem to be in **NP**, any single computational path can be evaluated by a DTM in polynomial time. So in a sense a DTM could efficiently decide an **NP** problem if only it had an efficient way to decide which path to take. This would imply **P=NP**, which of course we conjecture to be false.

Here's an idea: what if we picked a computational **path at random**? That is, at each step that NTM branches, we let a coin-toss decide which path to follow. It's not really *non-deterministic* because it doesn't branch, but it's not really deterministic either since it follows paths by chance.

Western Engineering

Probabilistic Turing Machines

Definition 2.

Probabilistic Turing Machine A **probabilistic Turing machine** (PTM) is a Turing machine that has access to a source of random bits. when faced with more than one legal next-move, the PTM uses the random bits to pick *exactly one* path to follow.

Probabilistic Turing Machines

At first glance a PTM picking a random computational path leaves us no better off than a DTM brute force searching exponentially many paths: if there are e.g., 2^n computational paths, and only m of them accept (for some $m \ll n$), then we have a vanishingly small chance (i.e., $P = \frac{m}{2^n}$) of picking this path at random.

But what if we add another ingredient: the ability for the decision to be ***wrong*** sometimes? Well maybe it doesn't help us with exponentially growing brute-force searches, but maybe there is class of problems that *would* be faster to solve...

Western Engineering

The Class **BPP**

Definition 3 (The class **BPP**).

Bounded-error probabilistic polynomial-time (**BPP**) is the class problems solvable by a probabilistic Turing machine in *polynomial* time with an error probability of at most $\frac{1}{3}$.

The choice of $\frac{1}{3}$ as an error bound is mainly by convention. Ultimately any number < 0.5 would suffice; the idea is that **BPP** problems are one in which the associated PTM (a) halts in polynomial time, and (b) is right *most of the time*.

Western Engineering

The Class **BPP**

Stated more formally, $L \in$ **BPP** iff there exists a PTM M where for all strings x:

- M halts in polynomial time,
- If $x \in L$, $P(M$ outputs *accept*$) \geqslant \frac{2}{3}$
- If $x \notin L$, $P(M$ outputs *accept*$) < \frac{1}{3}$

Western Engineering

PRIMES ∈ **BPP**

One of the classic **BPP** problems is PRIMES:

$$PRIMES = \{x : x \text{ is a prime number}\}$$

The first major primality test is due to Fermat, however, a certain class of *composite* numbers (called Psuedoprimes) will always pass the test, i.e.,

$$P(\text{M accepts} \mid x \text{ is a composite pseudoprime}) = 1$$

Fermat's test therefore isn't in **BPP** since the error probability exceeds the bound for certain inputs. An extension due to Miller and Rabin provides an unconditional probabilistic algorithm, i.e., error is bounded across all inputs.

More trials = more confidence

The error of the Miller-Rabin test is one-sided (no false negatives), and the probability of false positives can be decreased through repeated trails:

- Small chance of false positives:
  P(Output=*"prime"* | input was composite) $\leqslant \frac{1}{4^k}$ for k trails
- No chance of false negatives:
  P(Output=*"composite"* | input was prime) $= 0$

So after only 20 trials if the M-R test claims the input was prime, there's a less than *one-in-a-trillion* chance the input was composite.

Western Engineering

PRIMES $\in$ **P**

Despite a small probability of error, the Miller-Rabin test is extremely fast and easy to implement, and as such remains the most widely used primality test for real-world applications (e.g., internet security).

For many years (1980-2006) it was generally thought that a probabilistic polynomial solution to PRIMES was the best we could do. Then in 2006, Agrawal et al. proved an important result:

$$\text{PRIMES} \in \mathbf{P}$$

The AKS primality test runs in polynomial time, and, unlike Miller Rabin, is deterministic, i.e.,

$$P(\text{Output=}\textit{"prime"} \mid \text{input was composite}) = 0$$

Western Engineering

So What?

The fact that PRIMES was found to be in **P** underlines the sometimes staggering difference between complexity *theory* and complexity *practice*. Consider an earlier test due to Adleman, Pomerance and Rumely (the APR test). Like AKS it is deterministic, but has the following (non-polynomimal) complexity:

$$O((\ln n)^{c \ln \ln \ln n})$$

That's right: 3 consecutive ln's. By definition it's not polynomial because the exponent is a function of n. But as Manindra Agrawal joked, "although $\ln \ln \ln n$ grows to infinity, emperically it has been found to be less than or equal to 4." So although AKS is more efficient in general, it's actually considerably ***slower*** for any conceivable practical practical purpose.

Western Engineering

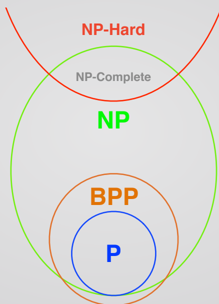
Does **P**=**BPP**?

Another major open question in complexity theory (in addition to **P** vs. **NP**) is whether **P**=**BPP**? This is somewhat of an interesting development, because when **BPP** was initially considered, it was thought that PTMs with bounded error *really could* solve more problems efficiently than DTMs.

Over the years, however, many **BPP** problems (like PRIMES) have been shown to be in **P** after all. In fact there are few known problems remaining in **BPP** that are not known to also be in **P** (one example is polynomial identity testing) leading us to conjecture **P=BPP**.

BPP vs. P and NP

Here's a depiction of the relationship between **P**, **NP** and **BPP**.



In the diagram **BPP** extends beyond **NP**, however $\mathbf{P} \stackrel{?}{=} \mathbf{BPP}$ remains an open problem in the theory of computation.

Western Engineering