GTI

More Complexity

A. Ada, K. Sutner
Carnegie Mellon University

Spring 2018

Total Recall

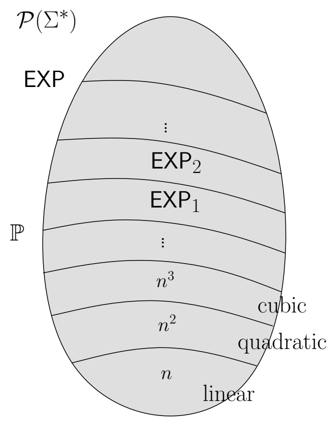
We are particularly interested in

$$\mathbb{P} = \mathrm{TIME}(\text{poly})$$

problems solvable in polynomial time. Alas, polynomial time is often elusive, and we may have to make do with exponential running times:

- $\mathrm{EXP}_1 = \bigcup \mathrm{TIME}(2^{c\,n} \mid c > 0)$, simple exponential time.
- $\mathrm{EXP}_k = \bigcup \mathrm{TIME}(2^{c\,n^k} \mid c > 0)$, kth order exponential time.
- $\mathrm{EXP} = \bigcup \mathrm{EXP}_k$, full exponential time.

Our World



Example 1: Shortest Paths

You are given a directed graph $G = \langle V, E \rangle$ with labeled edges $\lambda : E \to \mathbb{N}_+$.

Think of $\lambda(e)$ as the length (or cost) of edge e. The length $\lambda(\pi)$ of a path π in G is the sum of all the edges on the path.

The distance from node s to node t is the length of the shortest path:

$$\mathsf{dist}(s,t) = \min\big(\,\lambda(\pi) \mid \pi : s \longrightarrow t\,\big)$$

If there is no path, let's say $\mathsf{dist}(s,t) = \infty$.

Note that we only need to consider simple paths: no repetition of vertices.

Google Maps

The standard problem is to compute $\mathsf{dist}(s,t)$ for a given source/target pair s and t in G.

If you prefer decision problems write it this way:

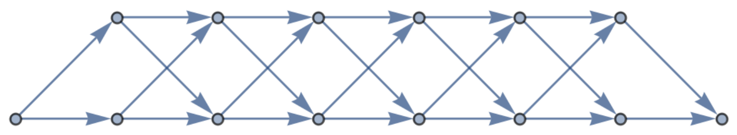
Problem:	**Distance**
Instance:	A labeled digraph G, nodes s and t, a bound B.
Question:	Is $\mathsf{dist}(s,t) \leq B$?

Note that the decision version and the function version are very closely related.

Upper Bound

A brute-force approach is to translate the definition directly into code: compute all simple paths from s to t, determine their lengths, find the minimum.

Straightforward, but there is a glitch: the number of such paths is exponential in general.



Dijkstra 1956

But one can get away with murder (read: a polynomial amount of computation) by acting greedy: at every step, always take the cheapest possible extension.

```
shortest_path( vertex s ) {
    forall x in V do
        dist[x] = infinity; add x to Q;

    dist[s] = 0;                        // s reachable

    while( Q not empty ) {
        x = delete_min( Q );                // x reachable
        forall  (x,y) in E  do
             if( (x,y) requires attention )
                dist[y] = dist[x] + cost(x,y);
    }
}
```

Dijkstra Contd.

Here Q is a priority queue (a specialized container data structure), and an edge (x, y) requires attention if

```
    dist[y] > dist[x] + cost(x,y);
```

Since all distance values are associated with an actual path, this means that the current value for y must be wrong.

It is a small miracle that updating only obviously wrong estimates at the cheapest node in a systematic manner is enough to get the actual distances in the end.

Example 2: Integer Multiplication

Multiplication of two k-bit integers seems to take $\Theta(k^2)$ steps:

$$12345 \cdot 6789 = 83810205 \qquad \begin{array}{cccccccc} 7 & 4 & 0 & 7 & 0 & & & \\ & 8 & 6 & 4 & 1 & 5 & & \\ & & 9 & 8 & 7 & 6 & 0 & \\ & & 1 & 1 & 1 & 1 & 0 & 5 \end{array}$$

After all, we have to write down all the digit-products and then perform a big addition at the end, right?

Divide-and-Conquer

Alas, one can try recursively attack the problem by breaking up the numbers into hi/lo-order bits. Let's say with use binary and $0 \leq x, y < 2^{2k}$.

$$x = x_1 \cdot 2^k + x_2$$
$$y = y_1 \cdot 2^k + y_2$$

Then

$$x \cdot y = x_1 \cdot y_1 2^{2k} + (x_1 \cdot y_2 + x_2 \cdot y_1) 2^k + x_2 \cdot y_2$$

Here $0 \leq x_i, y_i < 2^k$.

This produces a recurrence $t(n) = 4 \cdot t(n/2) + n$ for the running time.

Quoi?

We need to solve $t(n) = 4 \cdot t(n/2) + n$.

One good method is repeated substitution: plug the thing into itself a couple of times, and look for a pattern:

$$\begin{aligned} t(n) &= 4 \cdot t(n/2) + n \\ &= 4^2 \cdot t(n/2^2) + (2^2 - 1)n \\ &= 4^3 \cdot t(n/2^3) + (2^3 - 1)n \\ &\dots \\ &= 4^k \cdot t(n/2^k) + (2^{k-1} - 1)n \end{aligned}$$

Assume for simplicity $n = 2^k$ and get $t(n) = \Theta(n^2)$, no better than the kindergarten algorithm.

Karatsuba 1960

But there is a trick to get rid of one multiplication:

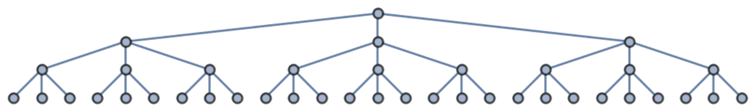
$$\begin{aligned} a &= x_1 \cdot y_1 \\ b &= x_2 \cdot y_2 \\ c &= (x_1 + x_2) \cdot (y_1 + y_2) \\ d &= c - a - b = x_1 y_2 + x_2 y_1 \end{aligned}$$

So we only need 3 multiplications total!

This produces a recurrence

$$t(n) = 3 \cdot t(n/2) + n$$

To solve the last recurrence, it is best to draw a call tree:



- The number of nodes at level k is 3^k (root is level 0).
- Each node at level k corresponds to $n/2^k$.
- So the total amount of work on level k is $n(3/2)^k$.
- There are $\log_2 n$ levels.

Calculemus

$$\begin{aligned} t(n) &= \sum_{i=0}^{\log_2 n} n(3/2)^i \\ &= n \sum_{i=0}^{\log_2 n} (3/2)^i \\ &= n \left((3/2)^{\log_2 n + 1} - 1\right) / (3/2 - 1) \\ &\approx 3 \cdot 3^{\log_2 n} \\ &= \Theta(n^{\log_2 3}) \end{aligned}$$

$$t(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$$

And Polynomials?

In many ways, polynomials behave much like integers (they form a ring).

Pleasant Surprise: Karatsuba's algorithm works just as well with polynomials.

The only difference is that we split the polynomial into high and low degree monomials. In terms of the coefficient list this comes down to a split in the middle.

Is This It?

One can try to subdivide the given number into smaller blocks (say, thirds) and get even better results, at least asymptotically.

But there is a much stronger result (Fürer, 2007):

Two n-bit integers can be multiplied in time

$$n \log n \, 2^{\log^\star n}$$

In the RealWorld$^{\mathrm{TM}}$, $\log^\star$ is constant.

Example 3: Matrix Multiplication

Matrix multiplication obviously can be handled in time $O(n^3)$:

$$A(i,j) = \sum_{k=1}^{n} B(i,k) \cdot C(k,j)$$

On the face of it, it looks like this bound is tight: matrix multiplication does seem to require $\Omega(n^3)$ operations since A has n^2 entries, and each costs n steps.

Small Examples

$$\begin{pmatrix}1 & 2 & 0\\ 2 & 0 & 1\\ 0 & 2 & 1\end{pmatrix} \times \begin{pmatrix}1 & 1 & 1\\ 1 & 0 & 1\\ 3 & 2 & 1\end{pmatrix} = \begin{pmatrix}3 & 1 & 3\\ 5 & 4 & 3\\ 5 & 2 & 3\end{pmatrix}$$

$$\begin{pmatrix}a & b & c\\ d & e & f\\ g & h & i\end{pmatrix} \times \begin{pmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{pmatrix} = \begin{pmatrix}c & b & a\\ f & e & d\\ i & h & g\end{pmatrix}$$

$\Omega(n^3)$ seems plausible, but it's quite wrong: one can use a divide-and-conquer strategy to break through the cubic barrier.

Strassen 1969

$$\begin{pmatrix}A & B\\ C & D\end{pmatrix} \times \begin{pmatrix}E & F\\ G & H\end{pmatrix} = \begin{pmatrix}AE+BG & AF+BH\\ CE+DG & CF+DH\end{pmatrix}$$

Some "random" auxiliaries:

$$\begin{aligned}
q_1 &= (A+D)(E+H)\\
q_2 &= D(G-E)\\
q_3 &= (B-D)(G+H)\\
q_4 &= (A+B)H\\
q_5 &= (C+D)E\\
q_6 &= A(F-H)\\
q_7 &= (C-A)(E+F)
\end{aligned}$$

Assembly

Looks wild, but costs only 7 $n/2$-multiplications, plus 14 $n/2$-additions. The actual product then looks like

$$\begin{pmatrix}q_1+q_2+q_3-q_4 & q_4+q_6\\ q_2+q_5 & q_1-q_5+q_6+q_7\end{pmatrix}$$

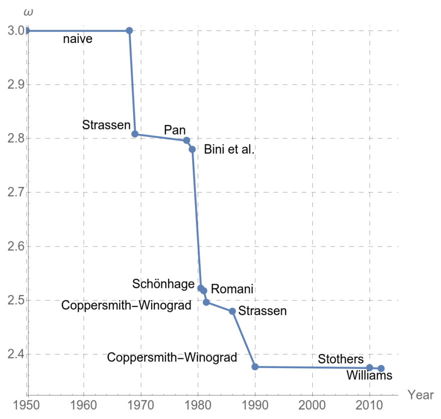
Recurrence now has form

$$T(n) = 7\,T(n/2) + n^2.$$

Solution: $O(n^{\log_2 7}) \approx O(n^{2.81})$.

Even Faster

Currently the best methods are a tiny bit better than a classical result by Coppersmith, Winograd from 1990 $O(n^{2.376})$



Example 4: Primality Testing

Suppose you have a k-digit number n, let's say $k = 1000$.

How do you check whether n is prime?

One can easily get rid of potential small divisors (like 2, 3, 5, 7, 11) in linear time using finite state machines, but that does not help much: we need to check factors up to $\sqrt{n} \approx 2^{500}$.

So this is one of the cases where we really have to use the logarithmic cost function, there is no point pretending that, say, a multiplication takes $O(1)$ steps.

Agrawal-Kayal-Saxena 2002

In 2002, a miracle happened: Agrawal, Kayal and Saxena showed that primality testing of k-bit numbers is in time polynomial in k.

Amazingly, the algorithm uses little more than high school arithmetic.

The original algorithm had time complexity $O(n^{12})$, but has since been improved to $O(n^6)$.

Alas, it seems useless in the RealWorld$^{\text{TM}}$, probabilistic algorithms are much superior.

Small Witnesses

The surprising algorithm uses polynomial arithmetic. Suppose wlog that an integer $n \geq 2$ is not a perfect power. Then n is prime iff for some suitable number r (called a witness for n):

$$(x+a)^n = x^n + a \pmod{x^r - 1}$$

for all $1 \leq a \leq \sqrt{r} \log n$ where all arithmetic is with modular numbers mod n.

This looks like a mess, but one can implement all the necessary operations in time polynomial in $\log n$, given the fact that the least witness r is not too large.

Pretty Good

We can use fast exponentiation (repeated squaring) to compute $(x+a)^n$. At each step, we reduce modulo $x^r - 1$.

So suppose we have some intermediate result

$$q(x) = \sum_{i<r} c_i x^i.$$

Squaring we get

$$q^2(x) = \sum_{i<2r} d_i x^i.$$

Reducing modulo $x^r - 1$ produces

$$\sum_{i<r} (d_i + d_{i+r}) x^i.$$

Unfortunately, it seems the actual running time, though polynomial, is so horrendous that this method will never be practical.

The Message?

In many cases, a problem description carries its own solution: the solution is clearly computable.

But getting an efficient solution is often much, much harder and requires additional insights (maybe even theorems).

The fast algorithms are often much harder to prove correct.

Unfortunately, we have reached the point where there is a race for better and better asymptotic running times, without anyone every implementing the algorithms, see Knuth.

Designing Hard Problems

Where is a good place to look for difficult decision problems?

Not quite as hard as Halting, we would like a problem that is still decidable but does not admit any fast algorithms.

A generic decision problem asks a specific question (with parameters) and the answer to the question may be hard to come by in some cases.

Brilliant Idea:
How about asking a lot of questions simultaneously?
Maybe even all possible questions?

Quoi?

"All possible questions" is obviously way too much, but maybe we could ask all questions in a certain domain? All questions that can be asked in some formal manner?

Problem:	**Entscheidungsproblem**
Instance:	A sentence Φ in some formal language.
Question:	Is Φ true?

For example, the sentence could be a statement in arithmetic:

$$\forall x \, \exists y \; (x < y \land \mathsf{prime}(y) \land \mathsf{prime}(y+2))$$

Syntactically fairly simple, but is it true?

Hilbert

Herr Hilbert was hopeful in the 1920s that the Entscheidungsproblem might be solvable for large areas of mathematics.

In 1930 Gödel took a big step in the direction of making Hilbert's dream come true: he showed that there is a nice system of logic (first-order logic) where

- all "true" sentences are provable, and
- the collection of proofs is decidable, and the collection of provable sentences is semidecidable.

In other words, proofs are fairly simple syntactical objects. There is no magic anywhere.

"Truth"

The "true" indicates that this requires a slightly different definition of truth than the one ordinarily used by mathematicians (and humans in general): it's really semantic consequence.

Suppose you have a system Γ of axioms and a particular formula Φ.

Suppose further that any structure that satisfies all the axioms also satisfies Φ: in any world described by Γ, the assertion Φ is true.

Then Φ is already provable from Γ: our logic knows about Φ.

Gödel Incompleteness

And then he ruined it all in 1931 by showing

- there is true sentence of arithmetic,
- that is not provable in any reasonable system.

Unfortunately, this means, among other things, that the Entscheidungsproblem (for arithmetic) is undecidable.

To be clear: Gödel's theorem does not clobber one particular system of arithmetic, it clobbers all possible systems. It's a feature, not a bug.

And Matiyasevic

In 1970 Matiyasevic proved that even a simple formula like

$$\exists x_1, x_2, \ldots, x_n \; P(a, x_1, \ldots, x_n) = 0$$

where P is a polynomial with integer coefficients cannot be tested for truth in general (Diophantine sets are semidecidable, but not decidable in general).

So one does not need very complicated sentences to ruin decidability for arithmetic.

The Upside

Without Gödel's Theorem, life would be endlessly boring.

And no one would need to hire a ToC expert. Just turn the crank if you need another theorem. Awful.

More importantly, one can now look for more limited domains of discourse and try to find a solution for the Entscheidungsproblem in the particular domain.

Presburger Arithmetic

How about weaker arithmetic, with fewer operations?

Realistically, the only useful choice is to drop multiplication. This yields Presburger arithmetic:

$$\mathfrak{N}_0 = \langle \mathbb{N}; +, <, 0 \rangle$$

Since multiplication is missing one cannot describe polynomials in this setting, only linear combinations.

So the problem of Diophantine equations disappears.

Admissible Operations

Full multiplication is absent, but multiplication by a constant is available; for example

$$y = 3 * x \iff y = x + x + x$$

We can also do modular arithmetic with fixed modulus:

$$\begin{aligned} y = x \bmod 2 &\iff \exists\, z\, (x = 2 * z + y \wedge y < 2) \\ y = x \operatorname{div} 2 &\iff \exists\, z\, (x = 2 * y + z \wedge z < 2) \end{aligned}$$

A non-trivial example of a valid Presburger formula:

$$\exists\, x\, \forall\, y\, \exists\, u, v\, (x < y \Rightarrow y = 5 * u + 7 * v)$$

Presburger's Theorem

Without multiplication arithmetic is much less complicated.

Theorem (M. Presburger 1929)

Presburger arithmetic is decidable.

Presburger's original algorithm is based on quantifier elimination: a formula is translated into an equivalent formula that has one fewer quantifier.

Unfortunately, it turns out that the complexity of Presburger arithmetic is pretty bad:

$$\Omega(2^{2^{cn}}) \qquad \text{and} \qquad O(2^{2^{2^{cn}}})$$

Aside: John von Neumann

Physics Nobel laureate E. Wigner was once asked why Hungary produced so many geniuses early in the 20th century.

He replied there was only one genius, John von Neumann.

There are lots of depressing anecdotes about him, among others the old dog-and-bicycle story, designed to flummox mathematicians.

Two bicycles, 20 km apart, moving at 10 kmh each, dog zigzags back and forth at 20 kmh till they meet.

How far did the dog run?

Neumann and Gödel

When Gödel first announced his incompleteness result at a conference in Königsberg in 1930, von Neumann was the only audience member who understood what just had happened.

Which is doubly remarkable since, up to this point, he had been working on Hilbert's program trying to find a decision algorithm for math.

And he independently discovered the second incompleteness theorem (a system cannot prove its own consistency).

Another Gödel Bombshell

It was generally assumed that Gödel never spent a second thinking about digital computers or complexity theory.

Surprisingly, in the 1980s a letter from Gödel to John von Neumann from 1956 surfaced, in which he outlines some fundamental complexity questions.

Unfortunately, von Neumann was dying of cancer, and Gödel apparently did not pursue the matter any further–a huge missed opportunity.