Limits of Computation

21 - How to ease the pain... (of **NP**-completeness)

Bernhard Reus

The story so far

- we don't know whether **P** = **NP**
- we have seen that there are particularly hard problems in **NP**, the **NP**-complete problems
- we don't know any polynomial time solvers for **NP**-complete problems
- so are we in serious trouble?

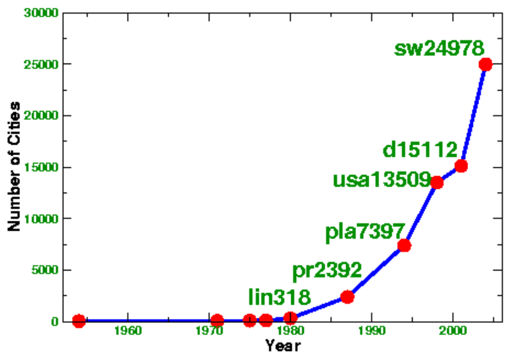
Getting around infeasibility?

THIS TIME

- how to attack **NP-** complete problems
  (various techniques)

- sometimes an **NP-** complete problem become essentially an asset

Progress in TSP solutions



www.math.uwaterloo.ca/tsp/methods/progress/progress.htm

Trying to ease the pain

- clever exact algorithms (*heuristic search)*
  that work well only for small(er) sizes
- approximation algorithms (*just not the optimum)*
- parallelism
- randomisation (use probabilities)
- harness natural phenomena *(the future?)*:
  - molecular (DNA) computing
  - quantum computing

Exact but for small(er) instances

- Use heuristic search (branch and bound, branch and cut, see AI)
- This will work well but only up to a certain size of input
- This is ok, if that's all you need.

Approximation

we focus on this (in context of TSP mainly)

Approximation guaranteed

function problem version

- for *optimization(!)*-version of **NP**-complete problems:
- instead of e.g computing the *shortest* TSP route, colouring with *fewest* colours, etc., produce a solution *in polynomial time* that is ***good*** (works for you) but is ***not*** *optimal*.
- Some optimization versions of **NP**-complete problems have good approximation algorithms, yet others do not. There is no uniformity!
- *Unfortunately, finding a solution of an* **NP***-complete problem with a* ***guaranteed*** *approximation is very often still* **NP***-complete.*

Approximation Algorithm

Definition 21.1 (α-approximation algorithm). An algorithm A for a maximisation problem is called *α-approximation algorithm* if for any instance x of the problem we have that

$$A(x) \leq OPT(x) \leq \alpha \times A(x)$$

where $OPT(x)$ is the optimal value for instance x of the problem. Analogously, an algorithm A for a minimisation problem is called *α-approximation algorithm* if for any instance x of the problem we have that

$$OPT(x) \leq A(x) \leq \alpha \times OPT(x)$$

We call α the *approximation factor*. In either case it holds that $\alpha \geq 1$.

Approximation Classes

NPO: optimisation, not decision problems now!

Definition 21.2. The complexity class **APX** (short for "approximable") is the set of optimization problems in **NP** that allow polynomial-time *approximation algorithms* where the approximation factor is *bounded by a constant*.

let's define something stronger:

Definition 21.3 (PTAS). The problem class **PTAS** is defined as the class of function optimisation problems that admit a **p**olynomial **t**ime **a**pproximation **s**cheme: this means that for any $\varepsilon > 0$, there is a polynomial-time algorithm that is guaranteed to find a solution which is at most ε times worse than the optimum solution. In other words, the approximation factor is $1 + \varepsilon$.

$$\mathbf{PTAS} \subseteq \mathbf{APX}$$

Why?

Scope of Approximation

For optimisation problems we have the following possibilities:

1. for every $\alpha > 1$ there is a polynomial time approximation algorithm with approximation factor α or better (so the problem is in **PTAS**) or

   0-1 Knapsack

2. there is a certain constant $\alpha > 1$, the *approximation threshold* such that an algorithm can produce a solution with approximation factor α or better in polynomial time, so the problem is in **APX**

   for many problems in APX it is unknown (up to P=NP) whether they are in PTAS

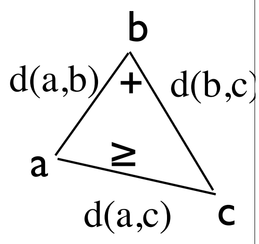
   MetricTSP, Graph Colouring

3. there is no $\alpha > 1$ that fulfills (2), so there is no approximation algorithm at all (unless $\mathbf{P} = \mathbf{NP}$) and thus the problem is not even in **APX**.

   TSP

Metric TSP

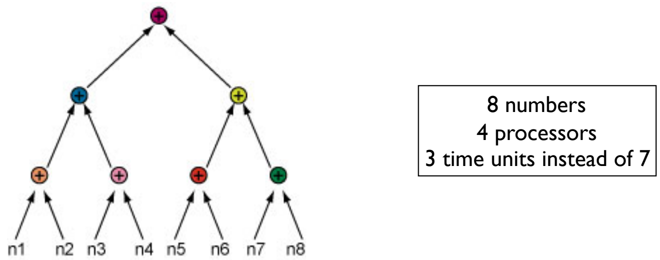
- The **Metric** version of TSP, where distances between cities satisfy the triangle inequality
- This does have a polynomial time algorithm that computes solutions approximatively at most 1.5 times the optimal length. (Christofides algorithm)



Other coping strategies

Parallelism

- To speed up time of adding *N* figures from linear to logarithmic time you need *N*/2 processors.



- You can use parallel processors to simulate the "nondeterminism" of NTMs for **NP**.

Parallelism (cont'd)

- Can **NP**-complete problems be solved in polynomial time on parallel processors?
- only in principle, **but not in reality** as
  - (1) *exponentially many processors* needed
  - (2) for them to communicate may/will again need *more than polynomial time!*
- Only ***constant*** *speed-up guaranteed* by ***constant*** number of processors in use.
- Still good to get speed-up: see Multi-Core Processors

Randomization: Vegas V Monte Carlo

- two approaches
  - Las Vegas:
    always correct but only *probably fast*

  - Monte Carlo:
    always fast, only probably correct;

RP: "yes-biased" Monte Carlo

randomised **p**olynomial-time

- return "yes" only if correct answer;
- error of *rejecting* correct answer has a *low probability p>0.*
- By (independently) repeating the run *n* times, probability of rejecting correct answer becomes p^n, i.e. as small as you like.
- Define **RP** as the class of problems decided by a Monte Carlo algorithm in polynomial time.

Class **RP**

- **P** ⊆ **RP** ⊆ **NP** Why?
- We don't know whether **P=RP** ? or **RP=NP** ? (other open problems)
- One can also define class **BPP** of problems decided by Monte Carlo algorithms that run in polynomial time and can give incorrect answers on *both* "Yes" and "No" instances with small probabilities.

  Bounded-error **p**robabilistic **p**olynomial-time
- **BPP** ⊆ **NP**? and **P=BPP**? are also open problems!

Problems in **RP**

- *Primality test (is a number n a prime number?)* can be solved *in polynomial time (in number of digits of n)* using *Monte Carlo:*
  [1975 Michael Oser Rabin (Turing Award Winner 1976) and Gary L Miller]

M.O. Rabin

- but the *Factorisation problem* (*what are the prime factors of a number?*) is not solved yet in polynomial time *even with randomisation.*

  Factorisation is a function problem but there is also an "up to polynomial time" equivalent decision problem
- non-randomised deterministic **P**-algorithm for *Primality test* in 2002

  Agrawal, **K**ayal, **S**axena (Kanpur)

Randomisation

- *Monte Carlo technique for* ***Optimisation*** *Problems:*
- **Simulated Annealing** (developed by Scott Kirkpatrick, Gelatt & Vecchi in 1983)
- a random walk between solutions, accept new solution with a certain probability, better solutions more likely but others possible; probability decreases during "cooling down" period of algorithm.

Scott Kirkpatrick
Turing Award winner 2011

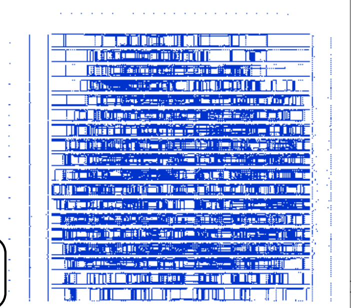
Return to Example: TSP

- There is an entire "industry" regarding the efficient solution of TSP.
- TSP is very common and easy to understand.
- TSP became a "benchmark" for any new discrete optimisation algorithm.

Example: TSP

- exact solutions:
  - branch and bound: about 40-60 cities
  - linear programming: ~86,000 locations
    In April 2006 an instance with 85,900 points on a VLSI chip was solved using *Concorde TSP Solver*, taking over 136 CPU-years, see Applegate et al. (2006). Simplex algorithm with cutting planes [Dantzig-Fulkerson-Johnson method dates back to 1954]

freely available C libraries



www.math.uwaterloo.ca/tsp

- heuristic/approximative
  - NearestNeighbour and many others: up to millions of cities but good solution only for (probable) cases (no guarantees)
- randomised: 700-800 cities

How “bad” complexity of a problem can become an asset!

When bad complexity is good

Rivest, Shamir, Adleman, MIT 1977; Turing-Award-Winners 2002

Public key cryptography ("RSA" algorithm):

- two primes p and q
- publish the public key computed from their product pq ...
- ... but not private data like p, q, $(p-1)(q-1)$ *that are all used to cipher and decipher* *(details in your favourite cryptography lecture/website)*

only secure if factorisation is infeasible!

Fatorisation problem

decision problem version

- We know that the Factorisation problem is in **NP**.
- We do ***not*** know whether it is **NP**-complete nor whether it is in **P** or in **RP**.
- But for security of RSA everybody implicitly relies on the fact that it is a hard problem.
- If **NP=P** then all of modern encryption (technology) could potentially collapse **if** the polynomial algorithm is indeed found and its runtime is not too bad.

END

Next Time: DNA computing