How to solve hard problems?

Want:

- Fast algorithms
- Correct algorithms
- Algorithms solving hard problems

Exact methods

- Intelligent search (i.e. A^* algorithm)
- Pseudopolynomial algorithms
- Integer linear programming
- Parametric complexity

Approximation algorithms

Algorithm A is a **k-approximation** if for each input x
$A(x) \leq kOPT(x)$ (maximization problems),
or $A(x) \geq kOPT(x)$ (minimization problems).

Algorithm A is a **polynomial-time approximation scheme (PTAS)** if its running time is **polynomial w.r.t. $|x|$** and for each $\varepsilon > 0$ it is
$(1+\varepsilon)$-apx (maximization problems),
or $(1-\varepsilon)$-apx (minimization problems).

Algorithm $A(x, \varepsilon)$ is a **fully-polynomial-time approximation scheme (FPTAS)** if in addition its running time is **polynomial w.r.t $1/\varepsilon$.**

Hammers: Careful analysis of greedy algorithms

(minimization problems)

- **Want:** compare the result of a greedy algorithm with the optimal solution
- **Lower bound L(x)** on the optimal solution
- **Upper bound U(x)** on the greedy solution
- Bound on the approaximation factor k:

$$k = \frac{\text{GREEDY}(x)}{\text{OPT}(x)} \leq \frac{U(x)}{L(x)}$$

(similarly for maximization problems)

Hammers: Split the problem into important and unimportant parts

- **Important parts:**
  - Contribute **large amount** to the solution value.
  - Have special properties (i.e. large value, small number of different types, etc.)
  - Based on these special properties it is possible to find **the optimal solution efficiently** $\Rightarrow$ solution (*)
- **Unimportant parts:**
  - Only contribute **small amount** to the solution value.
  - Solution (*) does not change its value too much if we simply add them to the solution.

Hammers: Divide the problem into smaller easily solvabe subproblems

- **Easily solvable subproblems:**
  - Have special properties (i.e. small span, special positioning, etc.)
  - Based on these spacial properties it is possible to find **the optimal solution efficiently**
  - Each subproblem $\Rightarrow$ its own optimal partial solution
- **Combine partial solution:**
  - often simply put them together
  - May not be optimal (i.e. unresolved overlaps)
  - Prove that this does not create a **too large error.**

Hammers: Round the large numbers

- Assumption: there exist a pseudopolynomial algorithm
- "Lower" the values (divide and round)
- Rounding creates errors.
- Show that these errors are not too large.

Hammers: ILP relaxation

- Write the problem instance is integer linear program.
- **Relax:** integer conditions $x \in \{0, 1\}$ replace with $0 \leq x \leq 1$.
- The relaxed ILP yields a pseudosolution whose value is **the same or better** than the optimal solution
- **Round the non-integer values of variables**
- Show that the solution did not change too much.

Randomized algorithms

Algorithms that use **random numbers.**

Las Vegas algorithms.

- Always give a correct answer.
- Random numbers affect running time $\Rightarrow$ **expected running time**

Monte Carlo algorithms.

- Always run fast.
- Sometimes give inccorrect answer $\Rightarrow$ **probability of error** p
  - Single sided (i.e. "yes" always correct, "no" may be erroneous)
  - Two-sided errors

Important: Running time / error **does not depend on the input**, only on random numbers choice! (no consistently "bad" input)

Hammers (LV): Randomization of a deterministic algorithm

- Instead of a deterministic step requiring "balanced" choice, make a random choice.
- Trick for expected runtime analysis:
  - Split all cases into **good** and **bad**.
  - **Good cases** with a good upperbound $u(x)$ on running time happen with large probability
  - **Bad cases** with a lousy upperbound $U(x)$ do not happen too often

$$E[T(x,r)] = \sum_{r} \Pr(r).T(x,r) = \sum_{r \text{ is good}} \Pr(r).T(x,r) + \sum_{r \text{ is bad}} \Pr(r).T(x,r)$$

$$\leq \Pr(r \text{ is good}).u(x) + \Pr(r \text{ is bad}).U(x)$$

Hammers (LV): Problem kernelization

- Some instances can be solved efficiently (i.e. sparse graphs, low weights, . . .)
- Use **random choice** to (repeatedly?) transform the problem into a new instance which satisfies these conditions **with high probability.**

Hammers (LV+MC): Random walks

- Reduce the problem to a random walk.
- Use known results about random walks:
  - Expected time of a random walks
  - Distribution of random walk times
  - How far we are likely to get during the random walk
  - ...

Hammers (LV+MC): Markov inequality (and others)

Let X be a random variable, where $X \geq 0$ a $E[X] = \mu$. Then $\Pr(X \geq c\mu) \leq 1/c$

Example: If we have a random walk that takes on average k steps, then with probability $\geq 1/2$ we will finish the walk in $2k$ steps.

Hammers (LV+MC): Finding witnesses

- If we would have an additional information, we would be able to solve the problem efficiently
  (i.e. partial order of elements, Fermat witness for a composite number, . . .)
- We call this information **a witness**.
- Use **a randomly generated witness** and verify!
- Show that we get a **bad witness** with **low probability**
  (witness leading to a long running time (LV) or bad answer (MC))

Hammers (MC): If it did not work, try it again

- Assume a MC algorithm with **one-sided error probability** p.
- If we run this algorithms k times, the probability that it makes a mistake in all runs is p^k
- One sided MC with $p = 1/2$,
  4 repeats: probability of correct answer $\approx 94\%$.
- One sided MC with $p = 0.9$
  20 repeats: probability of correct answer $\approx 88\%$

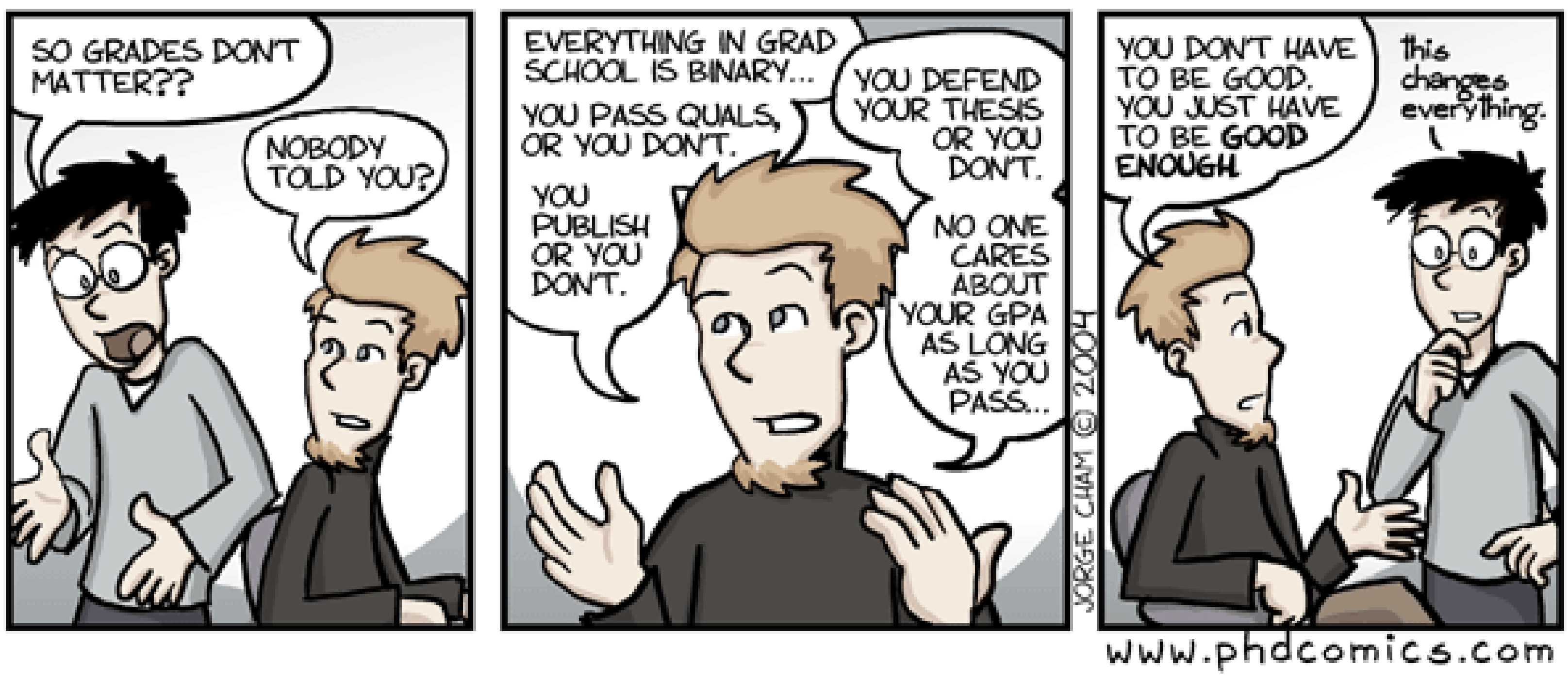
Hammers (MC): Fingerprinting

- Instead of comparing large objects bit-by-bit, compare only their **fingerprints.**
- Fingerprints should be short and easily comparable.
- Fingerprint computation depends on random numbers.
- Analysis: how often use of fingerprints can lead to incorrectly calling the identity?

PhD studies at FMFI UK

Doctoral studies content:

- 5% taking classes
- 20% teaching (tutorials, grading, etc.)
- 75% independent research



It is all about transformation...

Master degreee graduate:

- Can learn new things that somebody else came up with (typically from well-prepared books, manuals, or chat-GPT)
- Has demonstrated the (s)he can complete a project (master thesis) whose goals are set by somebody else (master thesis supervisor)

. . . PhD studies . . .

Successful researchers:

- Knows about newest advances in his/her field / is able to study half-baked research papers and fill the gaps.
- Can come up with new ideas that were not explored by others before.
- Can set up his/her research goals and judge if these goals are interesting for his/her colleagues or greater good.

What if I don't want to finish in academic research?

- Gain ability to work independently on new problems.
- Many of our graduates lead successful startups or work for prime employers such as Google, Facebook, etc.
- In the mean time: possibility to use a few years to explore your interests and call it a work (yes! it is paid!) . . . and have a (one last) chance to figure out what you really want to do with your life

Where to start? Find your future supervisor

Find supervisor: by mid-January, Application: by end of April

Computer graphics: prof. Ďurikovič, doc. Černeková, doc. Chalmovianský, doc. Ferko, doc. Madaras

Artificial intelligence: prof. Farkaš, Dr. Boža, doc. Markošová, doc. Homola, doc. Takáč

Theoretical computer science: prof. Královič, prof. Rovan, doc. Pardubská, prof. Škoviera, doc. Mačajová, doc. Mazák, doc. Lukoťka, doc. Jajcayová, doc. Guller

Distributed algorithms and computation: doc. Gruska, Dr. Dobrev (SAV)

Cryptology, information security: doc. Stanek, doc. Olejár

Bioinformatics: doc. Vinař, doc. Brejová

Software systems: doc. Polášek

Computer science education: prof. Kalaš, doc. Kubincová, doc. Tomcsányiová