Algorithms

Felix Dilke

A while ago, I was interviewed by Google

Didn’t get the job

It was still an amazing experience, spending a day at the “UK Googleplex” in Victoria, which featured:

- A whole room just about Google Earth with full-screen walls and a joystick
- Fully stocked micro-kitchens with celebrity chefs
- Developers loafing around in hammocks
- An interviewer who was like a friendly robot from the year 2111
- A (non-functional) rowboat with a full size Winnie the Pooh

When I got out, I pinched myself but the Google T shirt they gave me was still in my bag

Google are all about algorithms

They did tell me this and I should probably have done (even) more homework

You have to know all about trie trees, O(n log n), linked lists, bubble sort, the Fast Fourier Transform, etc

I had never really studied this stuff.

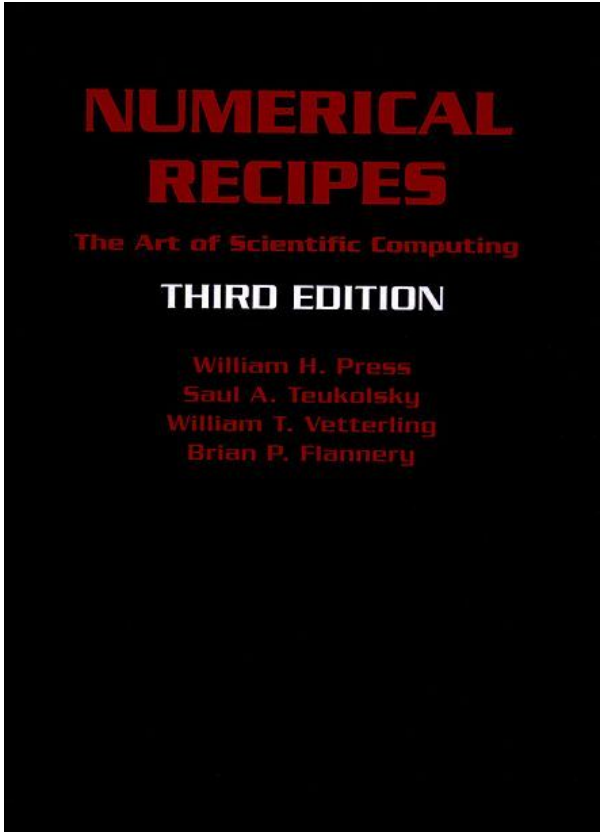
Also, it’s actually striking how LITTLE time we spend designing / optimizing algorithms as working software developers.

IMHO, the relevance of this to Springer Nature comes mostly from the ideas it develops about bottlenecks and optimization, particularly for distributed systems.

In my days of financial IT I did read this book though

It’s a bible of number crunching, and somehow manages to be witty and concise while imparting a huge amount of knowledge - rather like Kernighan and Ritchie, Kent Beck, and other classics of technical writing.

One of the most interesting algorithms is “simulated annealing” - of which more anon.



I was soon coming up with algorithms of my own

I worked for a company that wrote bond pricing software

This involved calculating bond yields, which involves solving “f(x) = 0” for a messy, but tractable function f

These were solved by the time honoured *Newton’s method* :

Given a candidate root x, repeatedly apply the correction x -= f(x) / f’(x)

My insight was that after two such corrections, and using some noddy algebra, you can make a pretty good guess at what the next correction will be

My colleagues called this The Newton Accelerator™. They were very impressed!

They shouldn’t have been

Like many would-be inventions I have come up with, this turned out to have been already done in 1877 but in a more sensible notation and with a proper theoretical grounding.

It’s not generally a good idea because Newton’s method already converges quadratically, i.e. very fast, so you’re mostly just making it (even) more unstable and likely to go off the rails and/or converge to the wrong root.

The “accelerator” does work for bond calculations, but only because the underlying function is so smooth and monotonic. Also, on an Intel 8086 with no coprocessor, and for impatient bond traders, the speedup is worthwhile.

So my algorithm did help, but for all the wrong reasons.

Another fun algorithm: Simulated annealing

This addresses the problem of “Find x in some large navigable set S with f(x) minimal”

The idea is to let x wander around in S, occasionally choosing suboptimal values, but slowly nudging the random walk in the direction of lower f-values, so that more of the space is explored with time, and a more optimal solution found.

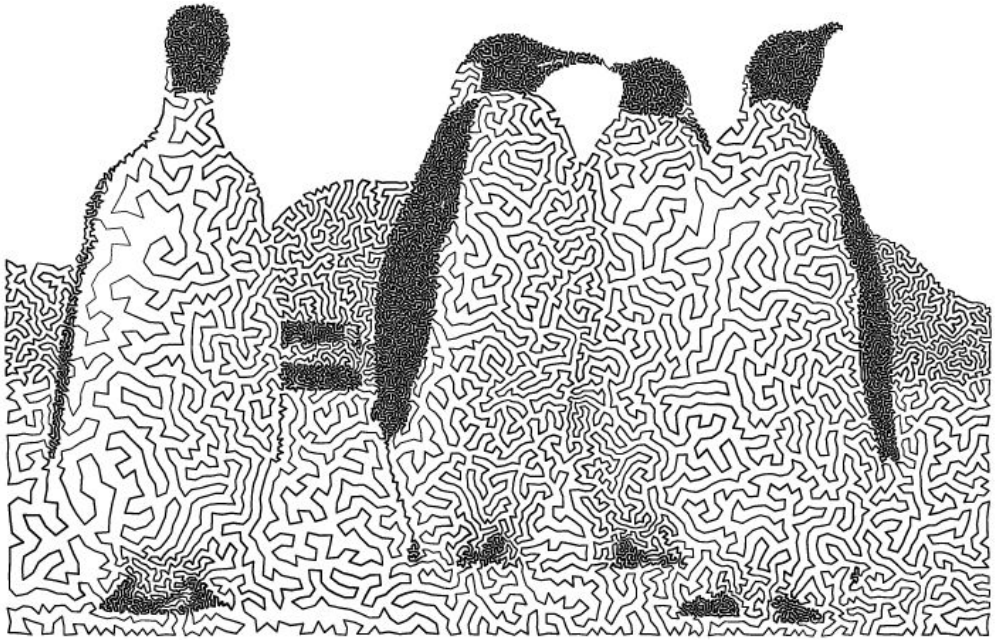
This is based on an analogy with the thermodynamics of what happens when molten metal cools down. The atoms end up in a more optimal (low-energy) configuration as a result of controlled random walk between possible sites.

This has essentially solved the Traveling Salesman Problem so now we can have TSP art

Problem is to find the shortest cyclic route between a bunch of points on a map

Simulated annealing finds an almost always good enough solution

So, strictly speaking, it’s a heuristic: You are not guaranteed to find the absolute optimum.



I hope this nicely illustrates...

...some of the philosophy and practice of algorithms, which I've been learning about in this book:

"The Algorithm Design Manual", Steven Skiena

This was on a list of recommended homework for the Google interview. It's quite readable and gives lots of ideas and examples.

Some insights from the first few chapters

Often, the business problem is solved by an informal, not very rigorous heuristic which turns out to be “generally good enough”

When it isn’t, you need a proper algorithm…

...and a proof that it gives the correct answer

So, we have to understand the algorithm well enough to PROVE that it works.

In other words, we have to do math :(

Modelling the problem

You need an abstract version of the problem, expressed in terms of standard data structures (combinatorial and geometric): Graphs, trees, lists, etc

It helps to know your way round these and understand their basic properties

Often, these structures are *recursive* - reflecting the fact that you can express the problem nicely in terms of a simpler version of itself

This leads to:

Recursive algorithms

Recursive, or inductive, proofs

Working with abstract algorithms

Ideally, algorithms should be

- Provably correct
- Efficient
- Straightforward to implement

To properly understand them and evaluate their efficiency, *as an exact science*, you have to do this at the right abstract level. Some tools that help:

- The RAM model of computation (an idealized VM for theoretical purposes)
- Asymptotic worst case analysis

More next time! THANK YOU