Finance through algorithmic lens

Maxim Litvak

2017-04-22

Outline

Outline

1st half - build algorithmic lens

- We'll get the fast overview of fundmentals of algortihms
- Take-aways: 1 picture, 2 paradoxes, a few example algorithms

2nd half - look through it on finance

- 3 practical cases
- Some theoretical insights
- Blockchain
- Auctions
- Non-monetary algorithms

Example: find an atom in the universe

- How fast can we find a specific atom in the universe?
- The current estimate is that there are 10^{80} ($\approx 2^{270}$) atoms in the universe.
- Sounds like a needle in a haystack at least ...

Example: find an atom in the universe

Algorithm

- Split the universe in 2 halves: in which half is the atom we need?
- Repeat

Result

- In 270 operations we'll find it
- Could be done by hand in less than 5 minutes ...
- We've seen an algorithm being extremely fast on a large input

Example: Hanoi tower

Example: Hanoi tower

Task

Move all disks from one peg (tower) to another

Rules:

- 1. Move one disk at a time
- 2. Every disk must be bigger than the one below
- 3. You can move only the upper disk

Example: Hanoi tower

- 3 moves needed for 2 disks
- 7 moves are needed for 3 disks
- 15 moves are needed for 4 disks
- ... million moves for 20 disks
- Number of moves goes up exponentially with number of disks!

Example: Hanoi tower

- We've seen a task with an algorithm that is fast on huge input (atom searching)
- We've seen a task with an algorithm that is slow on small input (Hanoi tower)

Picture to take away

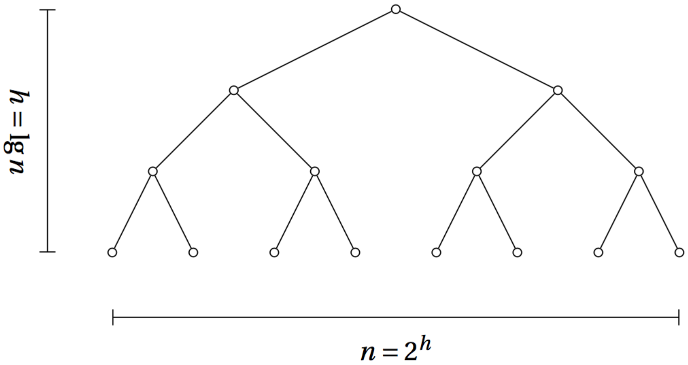


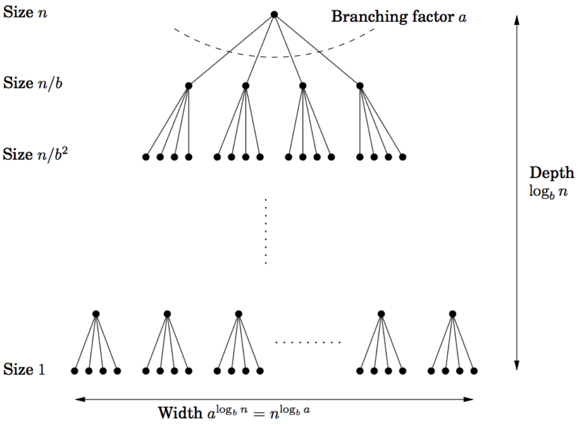
***Figure 3-3.** The height and width (number of leaves) of a perfectly balanced binary tree*

- The height is log(n) (fast), the width is $n = 2^h$, the number of nodes is $2n - 1 = 2^{h+1} - 1$ (slow)

Picture to take away

S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani 59

Figure 2.3 Each problem of size n is divided into a subproblems of size n/b.

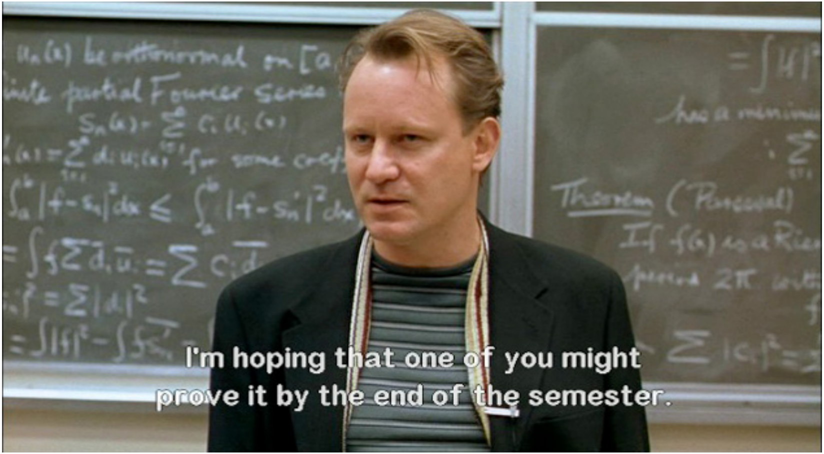


Paradoxes - Multiplication

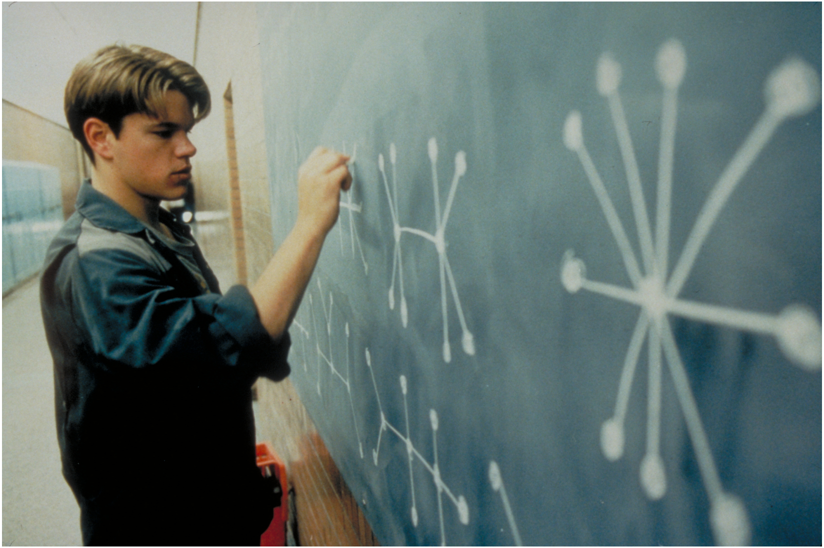
It's possible to multiply 2 numbers faster than the school method

- Andrey Kolmogorov in 1960 conjectured that there's no multiplication method faster than the school method (used by the humanity for thousands of years), and organized seminar to prove it
- A 17 years old student (Anatoly Karatsuba) found a faster method ...

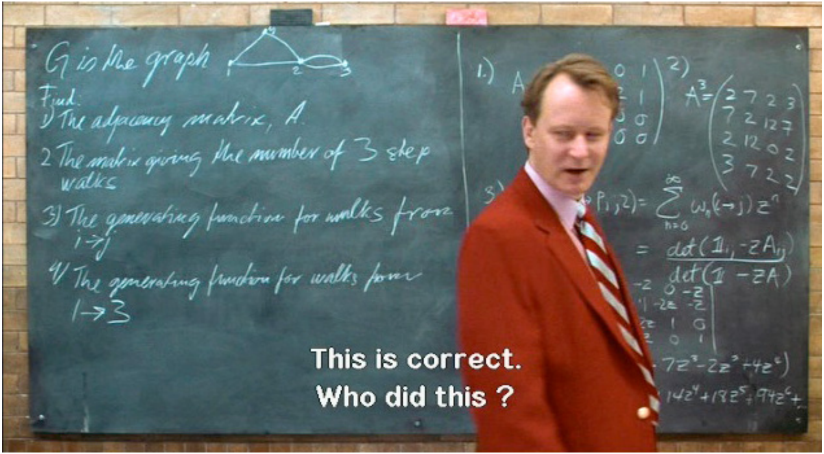
Paradoxes - Multiplication



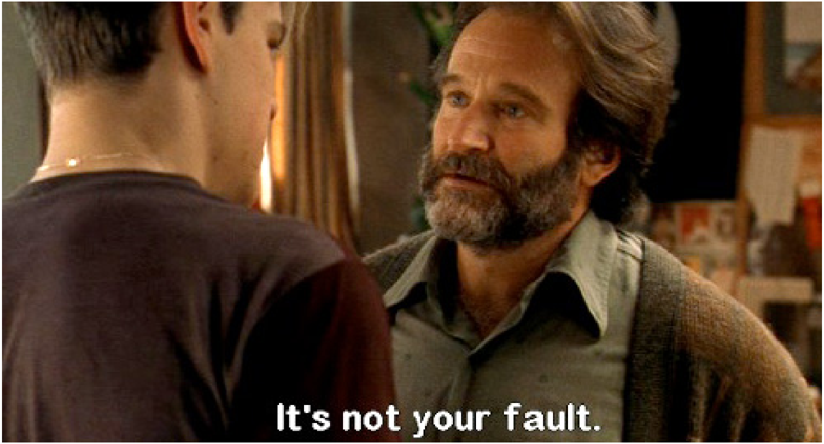
Paradoxes - Multiplication



Paradoxes - Multiplication



Paradoxes - Multiplication



Paradoxes - Multiplication

School

```
 25
x
 17
 ---
 175
+
 25
 ---
 425
```

$$(2\times10+5)(1\times10+7) = (2\times1)\times100+(2\times7+5\times1)\times10+(5\times7)\times1 = 425$$

Paradoxes - Multiplication

Karatsuba

```
(2*10 + 5)(1*10 + 7) = x*100 + y*10 + z*1 = ...
----------------------
x = 2*1 = 2
z = 5*7 = 35
y = (2 + 5)*(1 + 7) - (x + z) = 56 - 37 = 19
----------------------
... = 2*100 + 19*10 + 35*1 = 425
```

Paradoxes - Multiplication

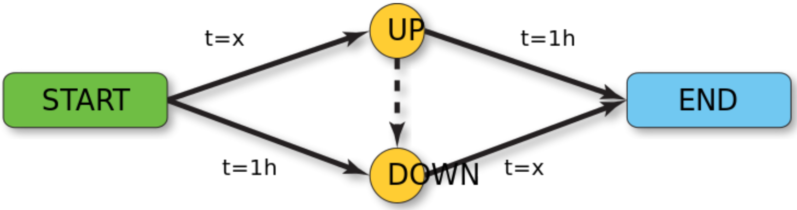
Difference

- School method - 4 multiplications and some additions
- Karatsuba method - 3 multiplications and some more additions
- Karatsuba is faster because the recursion can be deployed on multiplication

Paradoxes - Braess paradox

- Decreasing the number of opportunities might increase the efficiency
- In option pricing - adding an opportunity can't reduce the price

Paradoxes - Braess paradox

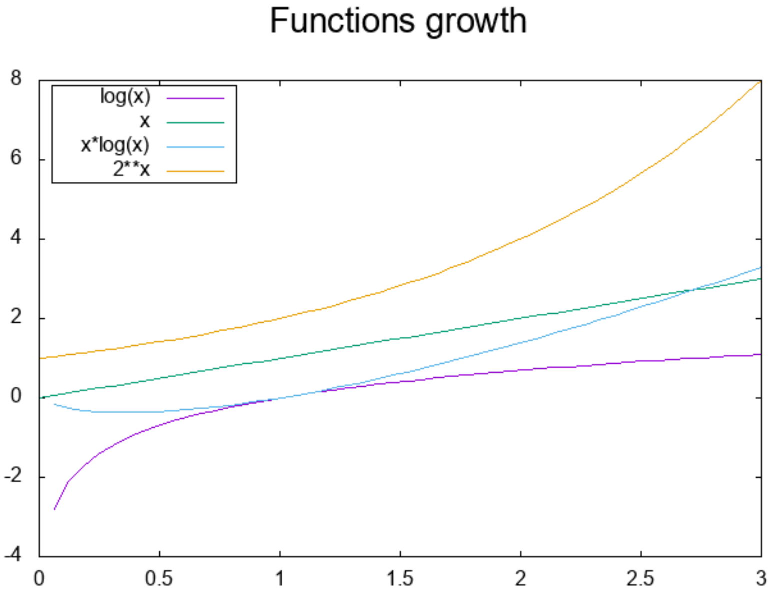


- The roads START-DOWN and UP-END take 1 hour, the roads S-up and Down-t depend on traffic (100% cars on the road - 1 hour, 50% - 30min)
- Initial equilibrium - 1.5h trip time (one half goes the upper way)
- Adding high-speed UP-DOWN highway (with 0h time to cross it) increases the time to 2h

What is "fast"

- From algorithmic point of view, under the "speed" is considered the asymptotic behaviour of the algorithm on input
- For a certain input, algorithm A might be faster than B, but "slower" in the asymptotic sense, i.e. with input big enough B outperforms A
- Big-O notation: $f(n) = O(g(n))$ if (up to a constant) for some n, $g(n)$ catches up or outperforms $f(n)$

What is "fast"



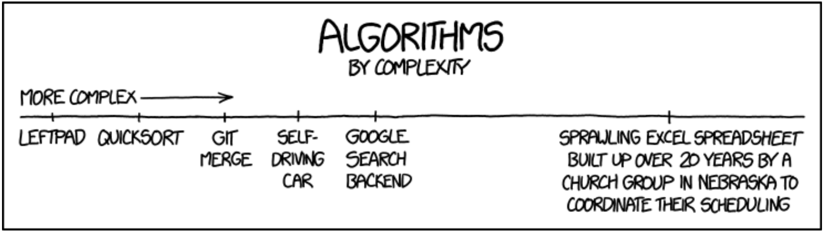
What is "hard" to compute

- As a rule of thumb, the algorithms that can be computed in polynomial time are considered as "easy" to compute
- ... and algorithms which computation time can't be bound by a polynom (e.g. exponential) are considered to be "hard"
- Sometimes computational "hardness" is a good thing, if you don't want something to be easily computed (e.g. cryptography)
- A lot of "hard" algorithms are considered to be "hard" since no faster solution is known (includes $P = NP$ problem)

Examples of algorithm's speed (current stand)

- Sorting numbers - O(n log(n))
- Binary search - O(log(n))
- Satifiability (if some logical statement is true or not) - exponential
- Integer number factorization (important in cryptography) - exponential

Non-standard examples on complexity . . .



- xkcd.com

Algorithmic topics in finance

- Cases in practice of quantitative finance
- Theoretical insights
- Blockchain
- Auctions
- Non-monetary mechanisms

Case: generate normal r.v.

- We can generate a normally distributed r.v. in a few ways, not all of them are equal computationally

(Bad solution)

- Generate 1000 uniformly distributed r.v. and use the CLT
- Their sum (centered by mean and normalized by standard deviation) is (approximately) normally distributed
- However, it's computationally expensive to do it this way

Case: generate normal r.v.

Another solution:

- use inverse c.d.f.
- $\xi = \Phi^{-1}(\eta)$, s.t. $\eta \in U_{0,1}$
- Next question: how expensive is it to compute inverse c.d.f.?

Subproblem: computation of inverse c.d.f.

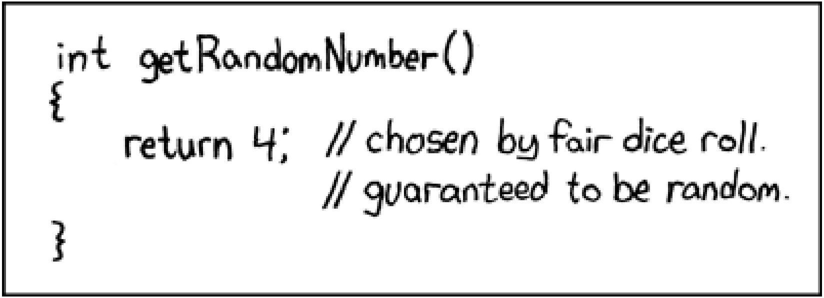
- Option: calculate using the root-finding procedure (on average 5-6 iterations, however, in this case the c.d.f. itself might be costly to compute)
- Option: use the Taylor expansion (well explored for normal distribution, might not be the case for other distributions)

Case: generate normal r.v.

- Other methods are known (Box-Miller procedure etc)
- However, the very 1st question: how costly is it to compute pseudo-random uniformly distributed r.v.?
- (guess) r.v. generators with better properties might be harder to compute

Case: generate normal r.v.

- Fast, but not so efficient generator . . . (xkcd.com)



Case: generate normal r.v.

- Example: middle square method
- Simple, but has different flaws
- Was enough for the simulations for the 1st nuclear bomb

middle square method

- Start with some n-digit number
- Square it
- Take the middle as the next number in the sequence
- Example: 6234^2 -> $38\underline{8627}56$, i.e. next number is 8627
- Problem: 7600^2 -> $57\underline{7600}00$, i.e. generator got stuck at the same number
- For 4 digits maximal period is 2^{12} (one of modern generators has the period 2^{19937})

Case: credit pricing

- A big bank re-calculated for controlling purposes each month its whole portfolio of credits (10^6 credits)
- For a lot of credits you need to do the same calculation (e.g. credits with the same conditions, but different principals)

Credit was priced (simplification) based on the following parameters:

- loan-to-value (LTV)
- r (interest rate)
- T (time to maturity)
- R (rating of the client)
- K (quality category of the principal)

Case: credit pricing

Idea

- why not to split all parameters into intervals (10 on average)
- pre-calculate all the possible situations (10^5)
- Then assign to each credit its price based on its parameters
- Rational: searching in the DB is faster than calculating

Case: credit pricing

- Some time later, a few improvements to the price engine were introduced
- instead of collateral category, there were introduced the mean and standard deviation of its value
- an option for extra payment (e.g. maximally 5% p.a. of the collateral)
- type of credit (most of the credits were annuities, but some were bullet loan credits etc)
- now, there were 10^8 credits in the database
- with a few other propositions to extend the number of parameters down the way . . .

Case: credit pricing

- time complexity (linear on number of credits and polynomial on number of parameters) was exchanged for the space complexity (exponential on number of parameters)
- at some point of time exponential growth outperformed the polynomial ...

Case: search for the best regression

- The frequent problem: given a number of factors, pick some of them, s.t. the dependend variable is explained in the "best" way
- The criteria can be e.g. some information criteria (e.g. Akaike)
- ... or some combined criteria (R^2 is high enough, p-values for coefficients are at least x%, p-value for normality test in residuals at least y% etc)

Case: search for the best regression

- Problem: the number of all combinations $\sum_{k=0}^{n} C_n^k = 2^n$, i.e. grows exponentially on number of factors
- e.g. for 50 factors, we would need to consider $2^{50} \approx 10^{15}$ regressions - too much

Case: search for the best regression

Attempt solution:

- constrain the number of possible factors (who needs regression with 100 factors?)
- e.g. for 50 possible factors, consider regressions with at most 4 factors, we'd need ca. 250 tsd. regressions - doable ...
- ... doable, but not scalable. Consider, we'd like also to consider the 1st lags (doubling the number of factors), then for 100 factors, we'd need to go through ca. 4 mln regressions

Case: search for the best regression

- The flaw of the attempt solution: it's faster, but it still grows exponentially
- p.s. $O(\sum_{k=0}^{k_{max}} C_n^k) = O(2^{nH(\frac{k_{max}}{n})})$, where H() is the binary entropy function

Case: search for the best regression

- Solutions in practice
- As the criteria (e.g. AIC) is chosen that assigns a number to a regression
- Deploy an optimal solution search method
- It usually has a polynomial complexity (however, not guaranted to find the global optimum)

Algorithms in practice of quantitative finance

- Different tasks require different speed
- Calculation of capital can easily take a few hours (or even days) longer
- With a client sitting in front of you, you want the program to calculate the conditions in 10-20-30 seconds
- In high-frequency trading you need to deploy the fastest algorithm, in the most efficient language on the fastest infrastructure

Theoretical insights

- Usual question in economic theory: does the equilibrium (e.g. price) exist?
- The algorithmic approach: how fast it can be computed?
- Nash-equilibrium is "hard" to compute
- One of ideas: markets are efficient if P = NP

Blockchain

- Blockchain is a distributed database of transactions
- "Hard to find, easy to check" principle used
- incentives used to ensure the fairness

Market capitalization as of 22.04.17

- Bitcoin \$20b
- Ethereum \$4.5b

Blockchain

- Some blockchain allows smart contracts (e.g. Ethereum)
- Would they reduce some risks (e.g. counterparty risk) and create the new ones? How to price them?
- Still an emerging field
- Cryptocurrencies are not exactly money

Money properties

- medium of exchange
- measure of value
- store of value

Auctions

- 2nd price auction principle - the highest bid wins, but the 2nd highest bid is paid
- Everytime you see an ad from google, an auction is run (i.e. millions auctions a day take place)

Non-monetary mechanisms

- not all tasks are solved with money, examples:
- matchings
- allocations

Matchings

- How to match different parties with different preferences
- e.g. stable marriage problem (match couples s.t. noone could improve their choice)

Examples

- In USA, medical students are assigned to hospitals in a similar way
- In France, teachers to public school

Allocations

- Allocate through exchange

Examples

- dormitory rooms exchange in China

Non-monetary mechanisms

- Simple matching/allocation algorithms are "easy to compute"
- Some not, e.g. hospital/residents problem with couples
- Hospital might admit multiple students, the constraint to assign a married couple to one hospital makes the problem "hard"

Thank you for your attention

- Thank you for your attention
- You can find the slides here: `https://github.com/maxlit/lectures/`
- Questions?

Material sources

- xkcd.com
- Dasgupta et al. "Algorithms"
- M.Hetland "Mastering Basic Algorithms in the Python language"