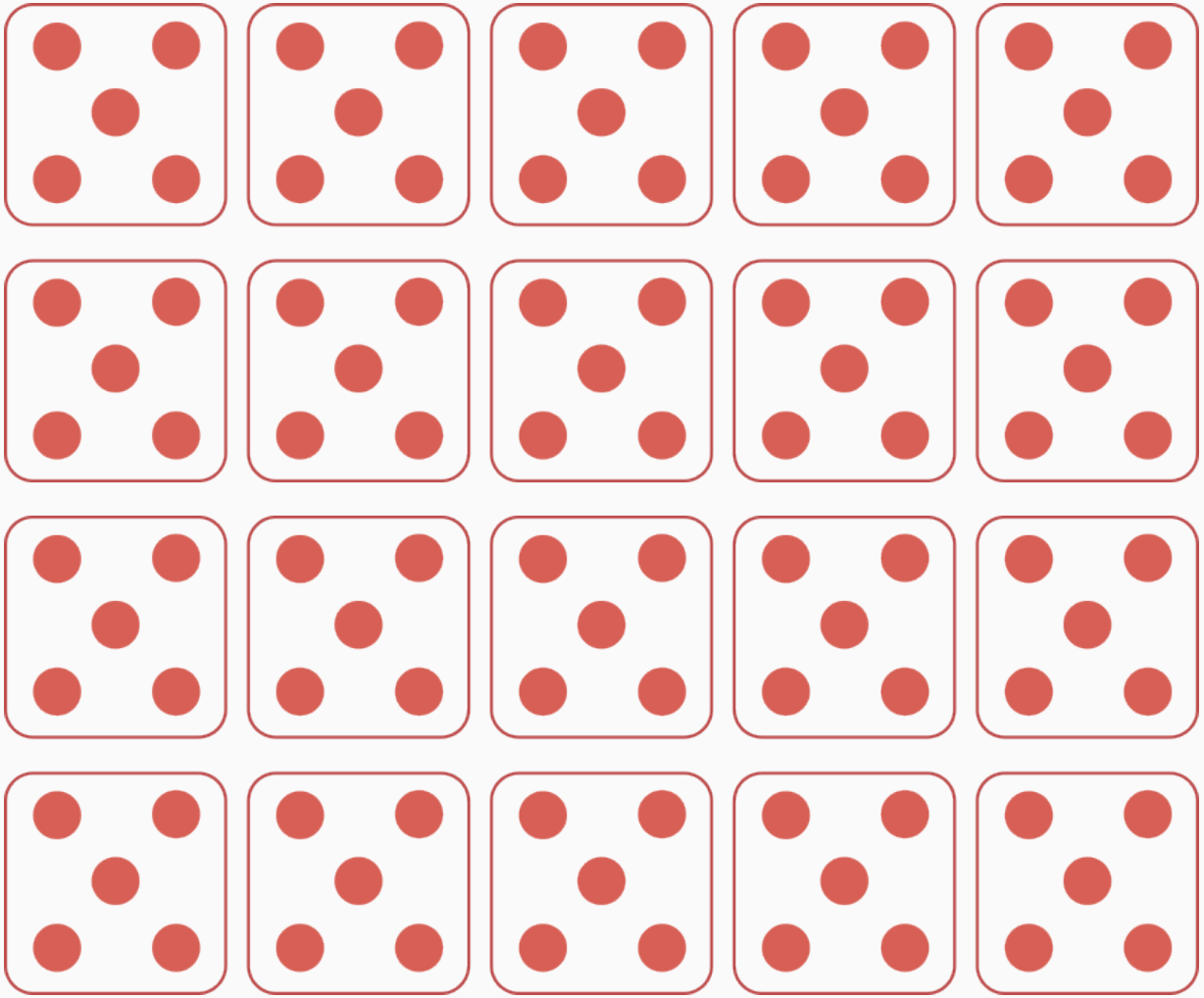
Algorithms for patent data

When efficiency (and brain) matters

Laurent R. Bergé
B_xSE, University of Bordeaux
2025-06-19 REGIS summer school

Why care about algorithms?

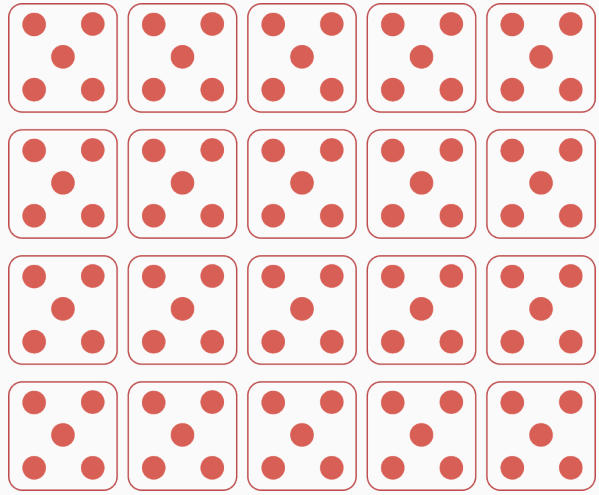
Can you count how many dots there is?



Two algorithms: Same results, different speeds

Algo 1: One step

1. count each dots 1 by 1
   - $1 + 1 + 1 + ... = 100$
   - ≈ 40s



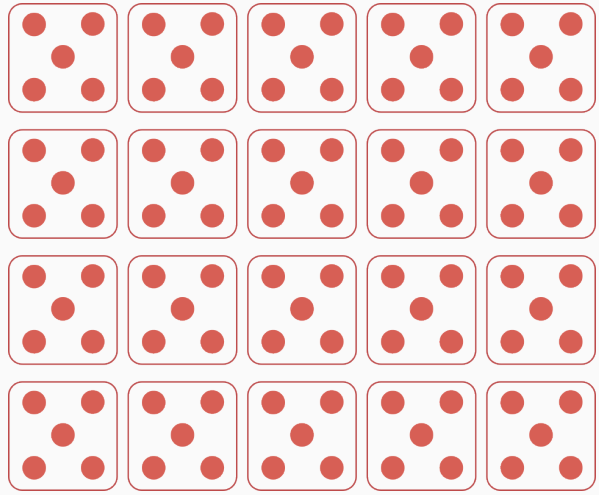
Two algorithms: Same results, different speeds

Algo 1: One step

1. count each dots 1 by 1
   - $1 + 1 + 1 + ... = 100$
   - ≈ 40s

Algo 2: Three steps

1. count the number of dots per square
2. count the number of squares
3. multiply
   - $5 \times 20 = 100$
   - ≈ 6s



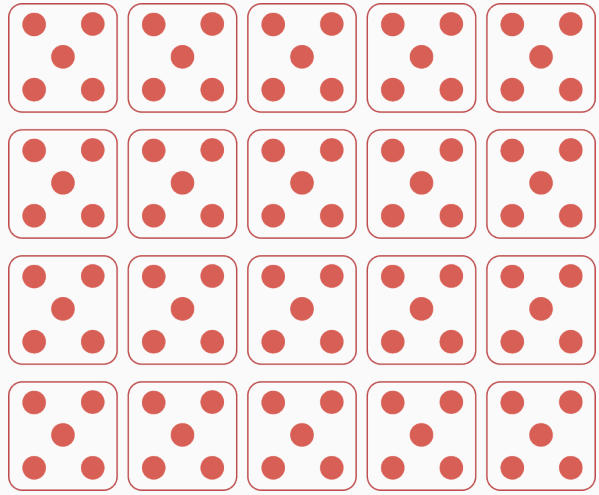
Two algorithms: Same results, different speeds

Algo 1: One step

1. count each dots 1 by 1
   - $1 + 1 + 1 + ... = 100$
   - ≈ 40s

Algo 2: Three steps

1. count the number of dots per square
2. count the number of squares
3. multiply
   - $5 \times 20 = 100$
   - ≈ 6s



Everybody chooses Algo 2

Two algorithms: Same results, different speeds

Algo 1: One step

1. count each dots 1 by 1
   - $1 + 1 + 1 + ... = 100$
   - ≈ 40s

Algo 2: Three steps

1. count the number of dots per square
2. count the number of squares
3. multiply
   - $5 \times 20 = 100$
   - ≈ 6s

When you use your **brain:**
=> you choose `algo 2`

Two algorithms: Same results, different speeds

Algo 1: One step

1. count each dots 1 by 1
   - $1 + 1 + 1 + ... = 100$
   - ≈ 40s

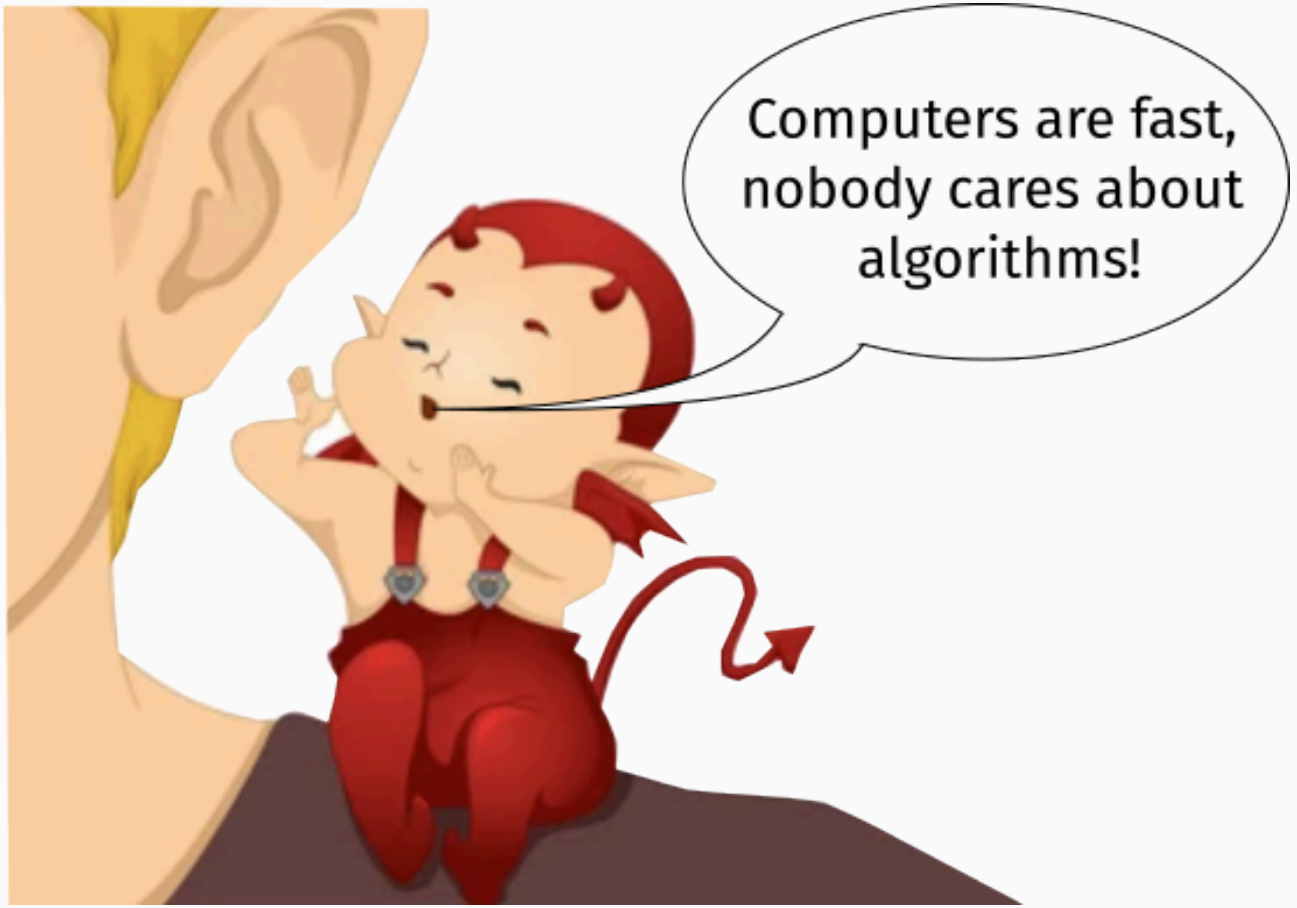
Algo 2: Three steps

1. count the number of dots per square
2. count the number of squares
3. multiply
   - $5 \times 20 = 100$
   - ≈ 6s

When you use your **brain:**
=> you choose `algo 2`

When you use your **computer:**
=> why do you use `algo 1`?

In the meantime…


Computers are fast,
nobody cares about
algorithms!

Why algorithms matter in empirical research

Typical research workflow

Before review

- find an idea (read stuff, think)
- develop a research design
- find the data
- clean the data
- produce desc stats, econometrics and graphs
- write

Why algorithms matter in empirical research

Typical research workflow

Before review

- find an idea (read stuff, think)
- develop a research design
- find the data
- clean the data
- produce desc stats, econometrics and graphs
- write

After review: i.e. please the referrees

- add more variables, more data
- get new results
- hope it works

Why algorithms matter in empirical research

Typical research workflow

Before review

- find an idea (read stuff, think)
- develop a research design
- find the data
- **clean the data**
- **produce desc stats, econometrics and graphs**
- write

After review: i.e. please the referrees

- **add more variables, more data**
- **get new results**
- hope it works

bold pink = highly time consuming!
affected by algorithms!

Research: Theory vs Practice

Theory

research design → clean data → get results → done

Research: Theory vs Practice

Theory

research design → clean data → get results → done

Reality

How to be more productive?

Cut the time cleaning the data and getting the results

How to be more productive?

Cut the time cleaning the data and getting the results

That's the point of good algorithms!

Why algorithms matter for innovation studies?

Data sets are large + the size explodes when we use text:

MAG (publications) 400 GB (200 GB for the abstracts)
Patstat full data set about 300 GB (includes lots of texts)
OECD REGPAT 6 GB (processed patent data)
USPTO dozens of GB

Algorithmic efficiency
A tale of two examples

Example 1: Text similarity measures

Arts, S., Cassiman, B., Gomez, J. C.
Text matching to measure patent similarity.
2018, **Strategic Management Journal** 39(1): 62-84.

Question

Can we use the text contained in **patent titles and abstracts** to assess similarity between them?

Example 1: Text similarity measures

Arts, S., Cassiman, B., Gomez, J. C.
Text matching to measure patent similarity.
2018, **Strategic Management Journal** 39(1): 62-84.

Question

Can we use the text contained in **patent titles and abstracts** to assess similarity between them?

Simple measure: Jaccard measure

i, j: patents W_i: set of words in abstract/title of patent i

$$\text{sim}_{i,j} = \frac{\text{size}\left(W_i \cap W_j\right)}{\text{size}\left(W_i \cup W_j\right)}$$

Practice!

patent number	title
1	regis is a super cool summer school
2	the regis summer school rocks
3	I will go on vacations this summer

Similarity of patents 1 and 2

`title 1:` regis is a super cool summer school
`title 2:` the regis summer school rocks

3 words in common
9 different words in total

$$\text{sim}_{12} = \frac{3}{9} = 0.33$$

Practice!

patent number	title
1	regis is a super cool summer school
2	the regis summer school rocks
3	I will go on vacations this summer

Similarity of patents 1 and 2

`title 1:` regis is a super cool summer school
`title 2:` the regis summer school rocks

3 words in common
9 different words in total

$$\text{sim}_{12} = \frac{3}{9} = 0.33$$

Let's code this up!

Similarity between two sentences

```
sim_single = function(x, y){
  words_x = strsplit(x, " ")[[1]]
  words_y = strsplit(y, " ")[[1]]

  sim = length(intersect(words_x, words_y)) / length(union(words_x, words_y))

  return(sim)
}
```

NOTE 1: This code is just an example in **R**. There are many ways to do this.
NOTE 2: There is no text cleaning.

Pariwise similarity for a vector of sentences

```
jaccard_algo_1 = function(x){
  #  in: vector of text
  # out: data.frame with i,j, jaccard measure

  n = length(x)
  base_pairs = expand.grid(words_i = x, words_j = x, stringsAsFactors = FALSE)

  pairs_id = expand.grid(i = 1:n, j = 1:n)

  base_pairs = within(base_pairs, {
    i = pairs_id$i
    j = pairs_id$j
  })

  base_pairs = subset(base_pairs, i != j)

  n_pairs = nrow(base_pairs)
  sim = numeric(n_pairs)
  for(index in 1:n_pairs){
    sim[index] = sim_single(base_pairs$words_i[index], base_pairs$words_j[index])
  }

  base_pairs$jaccard = sim

  return(base_pairs)
}
```

Q: What do you think of the previous code?

Q: What do you think of the previous code?

A: It is perfectly valid....

Q: What do you think of the previous code?

A: It is perfectly valid.... but sloooooow!

Q: What do you think of the previous code?

A: It is perfectly valid.... but sloooooow!

Before trying to understand why, let's see another algorithm!

Pairwise text similarity: New algorithm

1. notice that $\text{size}(W_i \cup W_j) = \text{size}(W_i) + \text{size}(W_j) - \text{size}(W_i \cap W_j)$
2.
3.
4.
5.

Pairwise text similarity: New algorithm

1. notice that $\text{size}(W_i \cup W_j) = \text{size}(W_i) + \text{size}(W_j) - \text{size}(W_i \cap W_j)$
2. create the data set `base_word_i` where each row is a patent x word, with variables `i` and `word`, and deduplicate it
3.
4.
5.

Pairwise text similarity: New algorithm

1. notice that $\text{size}(W_i \cup W_j) = \text{size}(W_i) + \text{size}(W_j) - \text{size}(W_i \cap W_j)$
2. create the data set `base_word_i` where each row is a patent x word, with variables `i` and `word`, and deduplicate it
3. create `base_word_j` a copy of `base_word_i` where you replaced `i` with `j`

`base_word_i`

i	**word**
1	regis
1	is
⋮	⋮
3	summer

`base_word_j`

j	**word**
1	regis
1	is
⋮	⋮
3	summer

4.

5.

Pairwise text similarity: New algorithm

1. notice that $\text{size}(W_i \cup W_j) = \text{size}(W_i) + \text{size}(W_j) - \text{size}(W_i \cap W_j)$
2. create the data set `base_word_i` where each row is a patent x word, with variables `i` and `word`, and deduplicate it
3. create `base_word_j` a copy of `base_word_i` where you replaced `i` with `j`

`base_word_i`

i	word
1	regis
1	is
⋮	⋮
3	summer

`base_word_j`

j	word
1	regis
1	is
⋮	⋮
3	summer

4. merge the two tables by `word` and drop the pairs with the same IDs

`base_words_ij`

word	i	j
regis	1	2
regis	2	1
⋮	⋮	⋮
summer	3	2

5.

Pairwise text similarity: New algorithm

1. notice that $\text{size}(W_i \cup W_j) = \text{size}(W_i) + \text{size}(W_j) - \text{size}(W_i \cap W_j)$
2. create the data set `base_word_i` where each row is a patent x word, with variables `i` and `word`, and deduplicate it
3. create `base_word_j` a copy of `base_word_i` where you replaced `i` with `j`

`base_word_i`

i	**word**
1	regis
1	is
⋮	⋮
3	summer

`base_word_j`

j	**word**
1	regis
1	is
⋮	⋮
3	summer

4. merge the two tables by `word` and drop the pairs with the same IDs

`base_words_ij`

word	**i**	**j**
regis	1	2
regis	2	1
⋮	⋮	⋮
summer	3	2

5. count, for each $i \times j$ pair, the number of words:
   tadam you have the numerator for all pairs!

Pairwise text similarity: New algorithm

6. in `base_word_i`, count, for each i, the number of words, you obtain $\text{size}(W_i)$
7.
8.
9.
10.

Pairwise text similarity: New algorithm

6. in `base_word_i`, count, for each i, the number of words, you obtain $\text{size}(W_i)$
7. merge that to the previous data set, you end up with

i	j	numerator	size_i	size_j
1	2	3	5	7
1	3	1	7	7
2	3	1	7	5
⋮	⋮	⋮	⋮	⋮

8.
9.
10.

Pairwise text similarity: New algorithm

6. in `base_word_i`, count, for each i, the number of words, you obtain $\text{size}(W_i)$
7. merge that to the previous data set, you end up with

i	j	numerator	size_i	size_j
1	2	3	5	7
1	3	1	7	7
2	3	1	7	5
⋮	⋮	⋮	⋮	⋮

8. create the denominator as: `denom = size_i + size_j - numerator`
9.
10.

Pairwise text similarity: New algorithm

6. in `base_word_i`, count, for each i, the number of words, you obtain $\text{size}(W_i)$
7. merge that to the previous data set, you end up with

i	j	numerator	size_i	size_j
1	2	3	5	7
1	3	1	7	7
2	3	1	7	5
⋮	⋮	⋮	⋮	⋮

8. create the denominator as: `denom = size_i + size_j - numerator`
9. compute the jaccard measure as: `numerator/denominator`
10.

Pairwise text similarity: New algorithm

6. in `base_word_i`, count, for each i, the number of words, you obtain $\text{size}(W_i)$
7. merge that to the previous data set, you end up with

i	j	numerator	size_i	size_j
1	2	3	5	7
1	3	1	7	7
2	3	1	7	5
⋮	⋮	⋮	⋮	⋮

8. create the denominator as: `denom = size_i + size_j - numerator`
9. compute the jaccard measure as: `numerator/denominator`
10. don't forget to add the pairs having 0 similarity (if needed)

New algorithm: code

```r
jaccard_algo_2 = function(x){
  #  in: x, text vector
  # out: data frame with i, j, jaccard

  library(data.table)

  # we split the sentences into words
  n = length(x)
  x_split = strsplit(x, " ")
  n_all = lengths(x_split)

  # we create the id x word data set
  base_word_i = data.frame(i = rep(1:n, n_all),
                           word = unlist(x_split))
  base_word_i = unique(base_word_i)

  base_word_j = setNames(base_word_i, c("j", "word"))

  # word, i, j data set
  base_word_ij = merge(base_word_i, base_word_j)
  base_word_ij = subset(base_word_ij, i != j)

  # continued on the right...
```

```r
  # we get the numerator
  setDT(base_word_ij)
  base_sim = base_word_ij[, .(numerator = .N),
                          by = .(i, j)]

  # we add the two sizes
  setDT(base_word_i)
  base_size_i = base_word_i[, .(size_i = .N),
                            by = i]

  base_sim = merge(base_sim, base_size_i, by = "i")
  base_sim = merge(
    base_sim,
    base_size_i[, .(j = i, size_j = size_i)],
    by = "j"
  )

  # compute the denom and the measure
  base_sim[, denom := size_i + size_j - numerator]
  base_sim[, jaccard := numerator / denom]

  # final step: adding numerator = 0
  res = as.data.table(expand.grid(i = 1:n, j = 1:n))
  res = res[i != j]
  res = merge(res, base_sim[, .(i, j, jaccard)],
              by = c("i", "j"), all.x = TRUE)
  res[is.na(jaccard), jaccard := 0]

  return(res)
}
```

Application

Let's test the two algorithms

Data set

The abstracts of 200 publications: 40k pairs.

Application

Let's test the two algorithms

Data set

The abstracts of 200 publications: 40k pairs.

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

base_1 = jaccard_algo_1(abstracts)
#> Elapsed:      5s 546ms 444us

base_2 = jaccard_algo_2(abstracts)
#> Elapsed:           6ms 544us

all(sort(base_1$jaccard) == sort(base_2$jaccard))
# [1] TRUE
```

Application

Let's test the two algorithms

Data set

The abstracts of 200 publications: 40k pairs.

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

base_1 = jaccard_algo_1(abstracts)
#> Elapsed:      5s 546ms 444us

base_2 = jaccard_algo_2(abstracts)
#> Elapsed:           6ms 544us

all(sort(base_1$jaccard) == sort(base_2$jaccard))
# [1] TRUE
```

Same results, different speeds!

Question

Q: Why is algo 2 faster depsite having more steps?

Let's try to understand why!

How a computer works 101

How does a computer work?

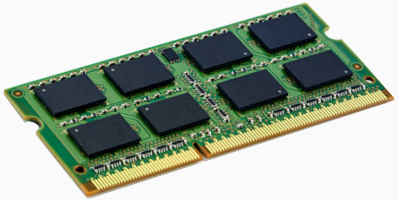
Magic!

How does a computer work?

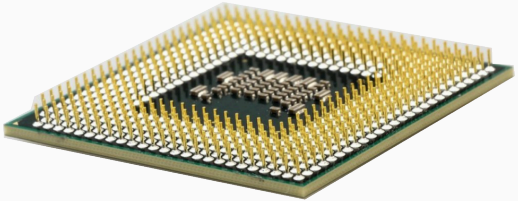
~~Magic!~~ Nope, physics!

It's just electrons moving!

Never forget the hardware!

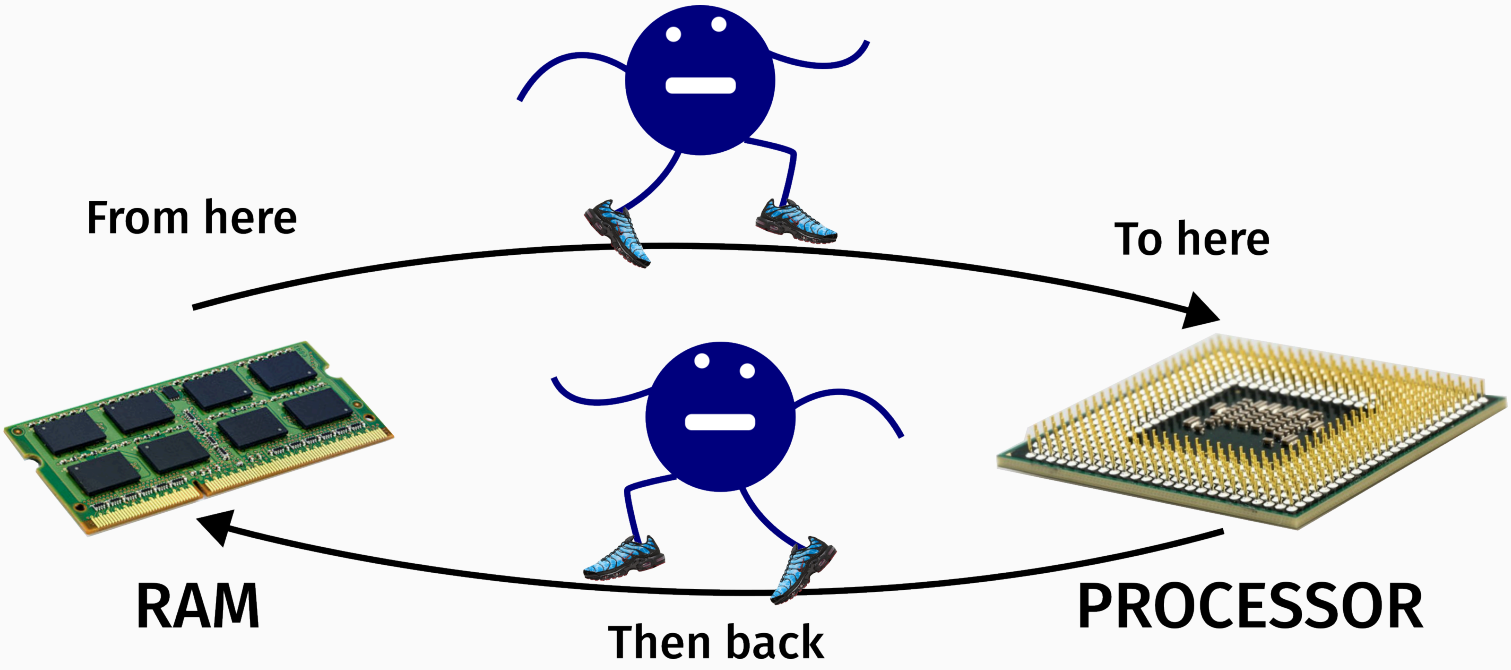


RAM



PROCESSOR

An electron's journey



Let’s put oursevles in the shoes of an electron’s family

The problem

You are the parent of a family of 10 electrons.

They all go to the university and you want to buy, for each of them:

- a table
- a desk
- a lamp

The problem

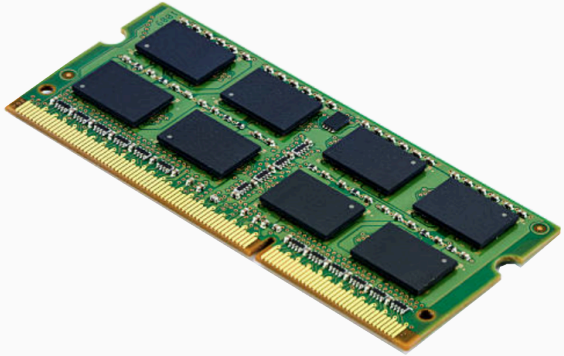
You are the parent of a family of 10 electrons.

They all go to the university and you want to buy, for each of them:

- a table
- a desk
- a lamp

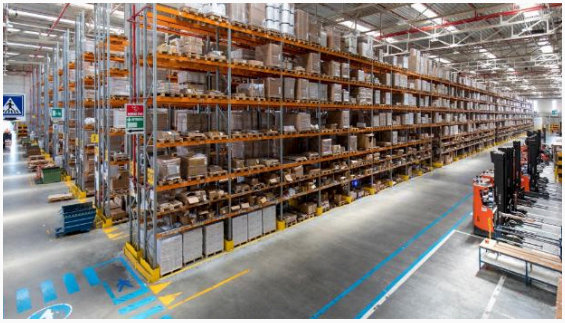
You buy all this at ~~IKEA~~ the **RAM**

RAM



=

Warehouse



Q: How do you plan your shopping?

Two shopping algorithms

Algo E1: Child-wise

For children 1
1. pick one table
2. pick one chair
3. pick one lamp

For children 2
1. pick one table
2. pick one chair
3. pick one lamp

... etc ...

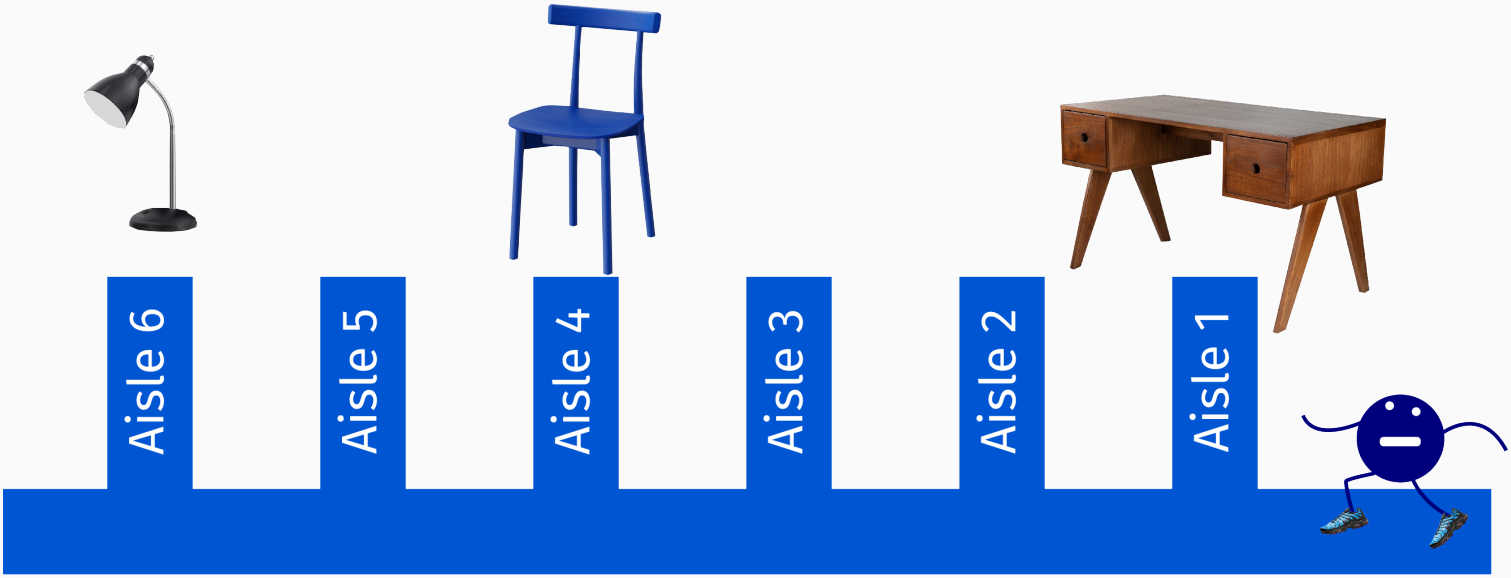
For children 10
1. pick one table
2. pick one chair
3. pick one lamp

Algo E2: Item-wise

1. pick 10 tables
2. pick 10 chairs
3. pick 10 lamps

Let’s shop!

With you finger: Apply the two algorithms



RAM's warehouse

Conclusion

Algo E2 minimizes the distance travelled by the electron. Hence it is faster.

Conclusion

Algo E2 minimizes the distance travelled by the electron. Hence it is faster.

Back to the text-similarity example

Solution 1

1. loop on each publication pair:
    - compute the text similarity for the pair

Solution 2

1. decompose the measure into simple components
2. compute each component on the full table
3. compute the text similarity from the components

Conclusion

Algo E2 minimizes the distance travelled by the electron. Hence it is faster.

Back to the text-similarity example

Solution 1

1. loop on each publication pair:
   - compute the text similarity for the pair

=> identical to Algo E1

Solution 2

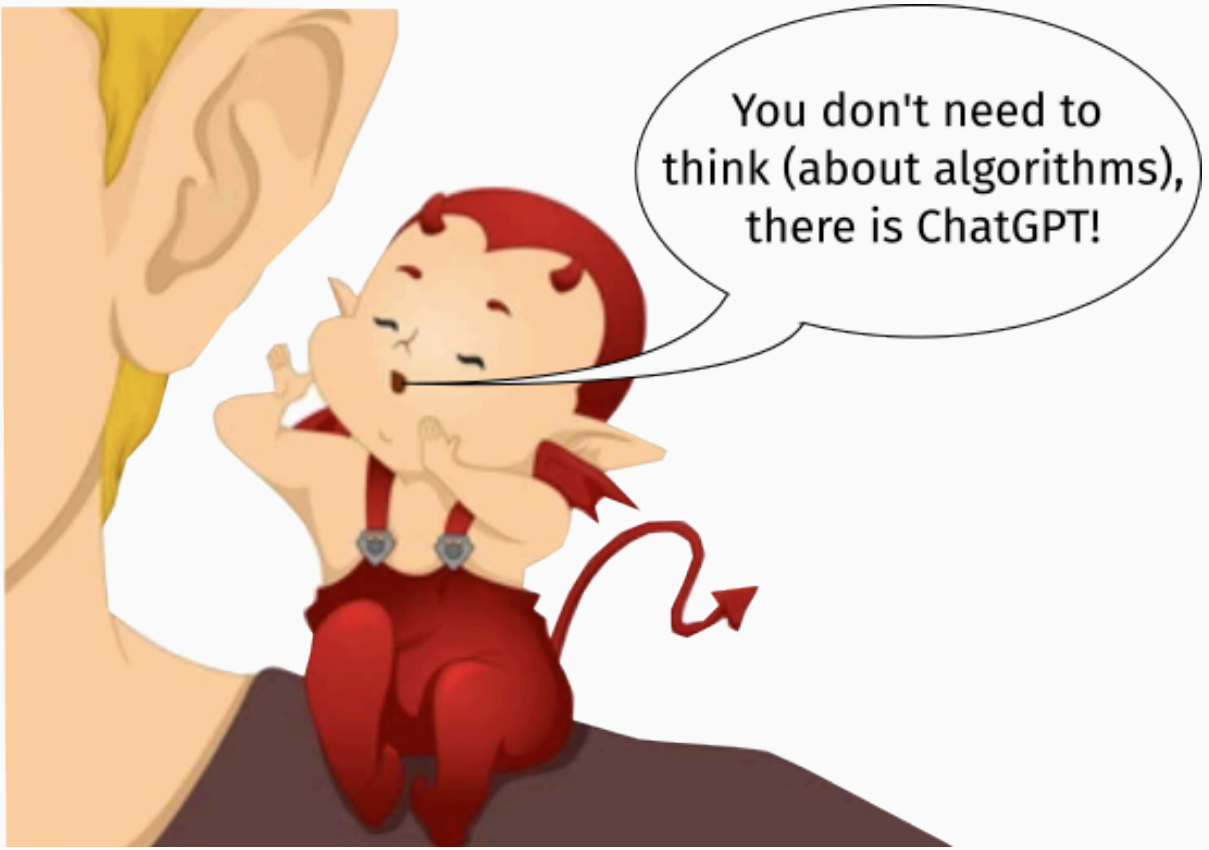
1. decompose the measure into simple components
2. compute each component on the full table
3. compute the text similarity from the components

=> identical to Algo E2

Takeaway

Rule 1: Shop efficiently in the RAM's warehouse!

In the meantime…


You don't need to think (about algorithms), there is ChatGPT!

ChatGPT: The prompt

Prompt

I am using R, and I have a vector of three sentences, as follows:

```
x = c(
  "regis is a super cool summer school",
  "the regis summer school rocks",
  "I will go on vacations this summer"
)
```

I would like to compute a the similarity between each of the elements of the vector. The similarity measure would be equal to the number of words in common divided by the total number of words across the two sentences.

Can you write a function that takes that vector as input and, as output, returns a data.frame with three variables: “i”, “j”, the indexes of the pairs being compared, and “sim” the similarity measure? Thanks.

Full discussion here: https://chatgpt.com/share/68529265-c2b8-800b-9340-0a2b9c6f31a9

ChatGPT: The result

```r
jaccard_algo_chatgpt <- function(sentences) {
  # Tokenize sentences into lowercase word sets
  word_sets <- lapply(sentences, function(s) {
    words <- tolower(s)
    words <- unlist(strsplit(words, "\\s+"))
    return(unique(words))
  })

  # Prepare result container
  results <- data.frame(i = integer(), j = integer(), sim = numeric())

  n <- length(sentences)

  for (i in 1:(n - 1)) {
    for (j in (i + 1):n) {
      words_i <- word_sets[[i]]
      words_j <- word_sets[[j]]

      common <- intersect(words_i, words_j)
      total <- union(words_i, words_j)

      sim <- length(common) / length(total)

      results <- rbind(results, data.frame(i = i, j = j, sim = sim))
    }
  }

  return(results)
}
```

Benchmark: ChatGPT algo

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

res1 = jaccard_algo_2(abstracts)
#> Elapsed:          91ms 512us

res2 = jaccard_algo_chatgpt(abstracts)
#> Elapsed:      9s 626ms 168us
```

Benchmark: ChatGPT algo

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

res1 = jaccard_algo_2(abstracts)
#> Elapsed:          91ms 512us

res2 = jaccard_algo_chatgpt(abstracts)
#> Elapsed:      9s 626ms 168us
```

Not great: let's make ChatGPT try harder!

ChatGPT: New prompt

Continuing the first prompt

Can you make the algorithm faster?

ChatGPT: New result

```r
jaccard_algo_chatgpt_faster <- function(sentences) {
  # Preprocess: convert to lowercase and split into unique word sets
  word_sets <- lapply(sentences, function(s) {
    unique(strsplit(tolower(s), "\\s+")[[1]])
  })

  n <- length(sentences)
  n_pairs <- choose(n, 2)

  # Preallocate result list
  results <- vector("list", n_pairs)
  idx <- 1

  for (i in 1:(n - 1)) {
    words_i <- word_sets[[i]]
    for (j in (i + 1):n) {
      words_j <- word_sets[[j]]

      common <- intersect(words_i, words_j)
      total <- union(words_i, words_j)
      sim <- length(common) / length(total)

      results[[idx]] <- c(i, j, sim)
      idx <- idx + 1
    }
  }

  # Convert to data.frame
  results_df <- as.data.frame(do.call(rbind, results))
  names(results_df) <- c("i", "j", "sim")

  # Convert column types
  results_df$i <- as.integer(results_df$i)
  results_df$j <- as.integer(results_df$j)
  results_df$sim <- as.numeric(results_df$sim)

  return(results_df)
}
```

Benchmark: New ChatGPT algo

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

res1 = jaccard_algo_2(abstracts)
#> Elapsed:           78ms 544us

res2 = jaccard_algo_chatgpt_faster(abstracts)
#> Elapsed:         472ms 554us
```

6 times faster, but does it scale?

Benchmark: New ChatGPT algo

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

res1 = jaccard_algo_2(abstracts)
#> Elapsed:           78ms 544us

res2 = jaccard_algo_chatgpt_faster(abstracts)
#> Elapsed:         472ms 554us
```

6 times faster, but does it scale?

```
# does it scale? Let's check with 400 abstracts
double_abstracts = c(abstracts, abstracts)

res1 = jaccard_algo_2(double_abstracts)
#> Elapsed:           96ms 456us

res2 = jaccard_algo_chatgpt_faster(double_abstracts)
#> Elapsed:      1s 952ms 263us
```

Now 20 times faster: it does scale.

ChatGPT: Give all you got!

Accepting ChatGPT's suggestion

`ChatGPT:` Let me know if you want an even faster implementation using a package like `data.table`.

`Me:` Yes, can you make a faster one with data table?

ChatGPT's fastest code

```r
jaccard_algo_chatgpt_fastest <- function(sentences) {

  library(data.table)

  n <- length(sentences)

  # Tokenize each sentence into a list of unique lowercase words
  tokens <- lapply(sentences, function(s) {
    unique(strsplit(tolower(s), "\\s+")[[1]])
  })

  # Create all unique pairs of sentence indices
  pairs <- CJ(i = 1:n, j = 1:n)[i < j]

  # Compute similarity for each pair
  pairs[, sim := mapply(function(i, j) {
    words_i <- tokens[[i]]
    words_j <- tokens[[j]]
    length(fintersect(words_i, words_j)) / length(funion(words_i, words_j))
  }, i, j)]

  return(pairs[])
}
```

Benchmark: Fastest ChatGPT algo

```
# 200 abstracts
abstracts = readLines("./_DATA/pub-abstracts.txt")

res1 = jaccard_algo_2(abstracts)
#> Elapsed:           77ms 378us

res2 = jaccard_algo_chatgpt_fastest(abstracts)
#> Error in .set_ops_arg_check(x, y, all, .seqn = TRUE) :
#>  x and y must both be data.tables
```

Brain vs ChatGPT: Conclusion

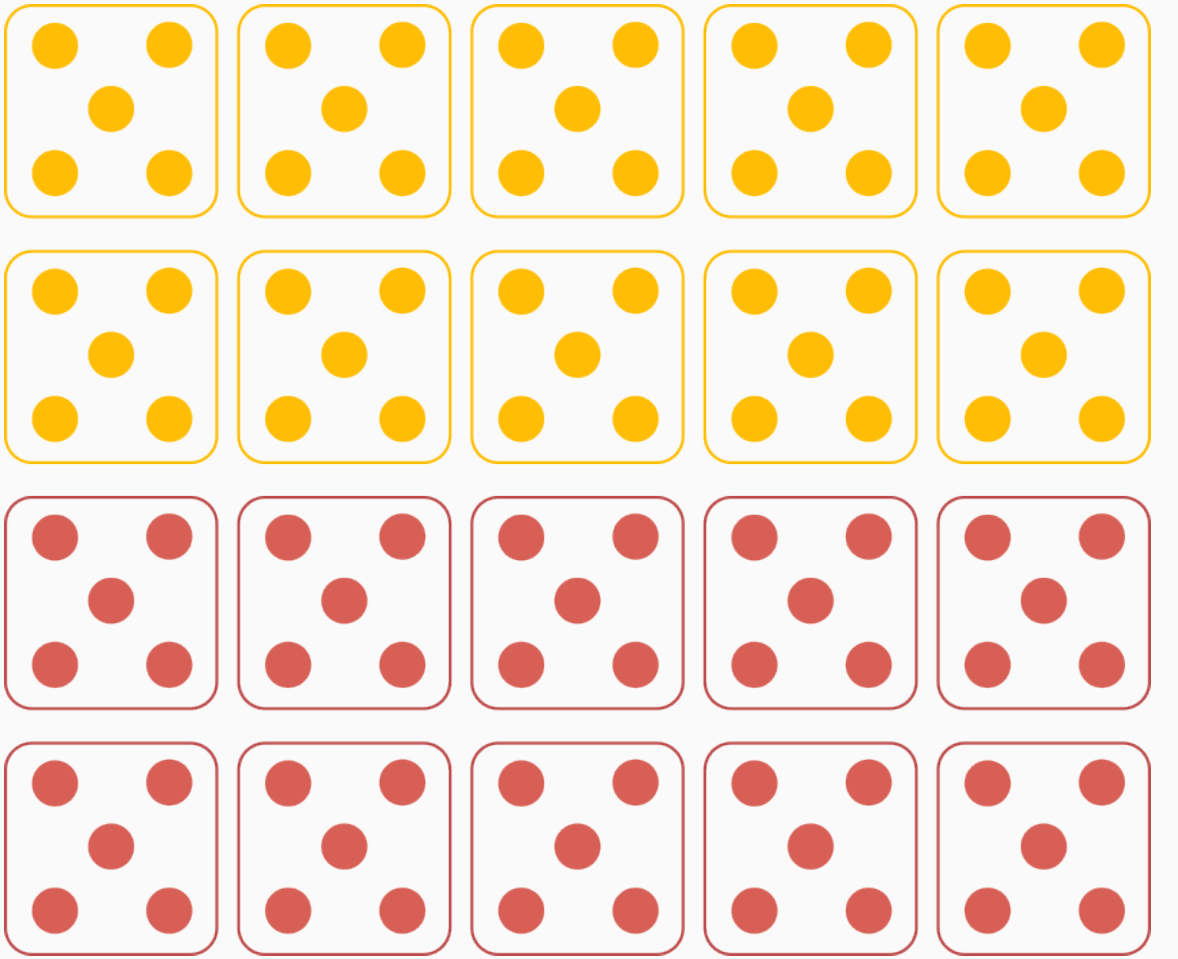
Contrary to what AI enthousiasts want you to believe:

Learning and thinking is still useful

Example 2: Econometrics

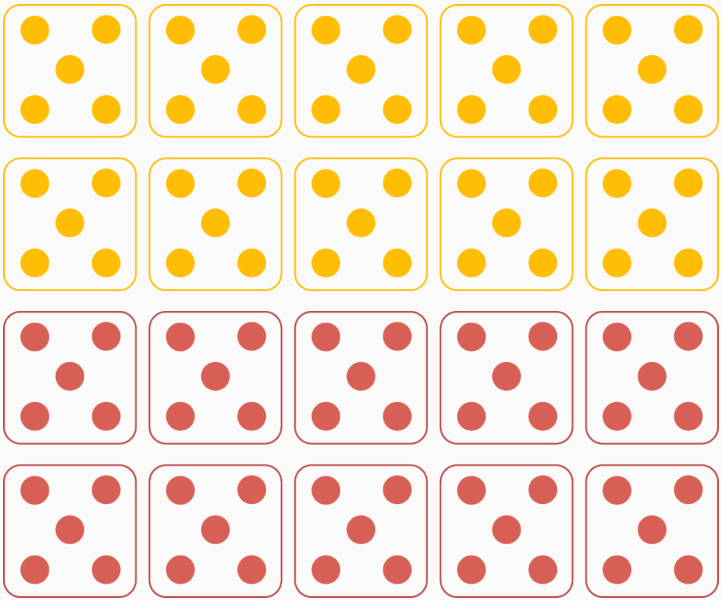
Can you count how many red dots there is?

Task: Count the red dots



Never do more than what you need

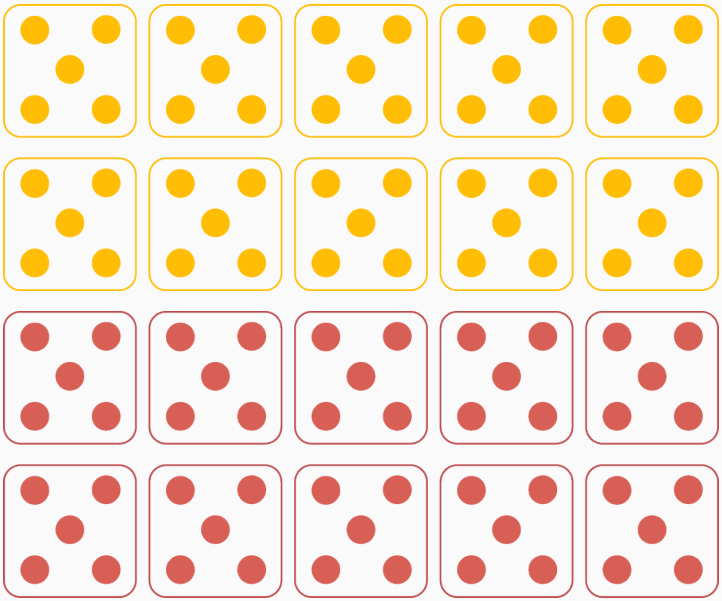
Task: Count the red dots



When you use your **brain:**
=> you only count the red dots

Never do more than what you need

Task: Count the red dots



When you use your **brain:**
=> you only count the red dots

When you use your **computer:**
=> why do you also count the yellow dots?

Back in 2020 (when twitter was cool)

- PhD student asking on twitter:★
  - "I'm using Imai and Song weighted fixed-effects estimator."
  - "My estimations take forever. "
  - "Help!"

★: I tried but couldn't find the tweet! If by chance you find it, please send it to me!!!

In the meantime…


Why try to find an answer to this problem? Just ask ChatGPT!

ChatGPT solution to the econometrics problem

The prompt

Prompt

I am using the function wfe from the wfe package in R. Here is an example of how I use it:

```
# start example code
library(wfe)
library(fixest)
library(data.table)

# example data set
data(base_did, package = "fixest")
set.seed(0)
base_example = as.data.table(base_did)
id_first = sample(108, 20) ; id_last = sample(108, 20)
base_example = base_example[!(id %in% id_first & period <= 3) & !(id %in% id_last & period >= 7)]
base_example[, treat_post := treat*post]

wfe_v1 = wfe(y ~ treat_post + x1, data = base_example, treat = "treat_post",
             unit.index = "id", time.index = "period", method = "unit",
             qoi = "ate", hetero.se = TRUE, auto.se = TRUE)
# end of example code
```

I find this function slow, especially for large data sets. Can you optimize it, and make it run faster?

ChatGPT: Solution 1

✅ 1. Use `fixest` as a faster alternative

If your estimation can be approximated via high-dimensional fixed effects and clustered SEs, `fixest::feols()` is extremely fast and memory-efficient. Example:

r

Copier Modifier

```r
library(fixest)

# Approximate wfe with two-way fixed effects
fixest_model = feols(y ~ treat_post + x1 | id + period, data = base_example, cluster = ~id)
```

This won't be identical to `wfe()` (which weights by treatment timing and accounts for dynamic timing of treatment effects), but it gives a similar ATE interpretation under parallel trends.

ChatGPT: Solution 1

✅ 1. Use `fixest` as a faster alternative

If your estimation can be approximated via high-dimensional fixed effects and clustered SEs, `fixest::feols()` is extremely fast and memory-efficient. Example:

r

Copier Modifier

```r
library(fixest)

# Approximate wfe with two-way fixed effects
fixest_model = feols(y ~ treat_post + x1 | id + period, data = base_example, cluster = ~id)
```

This won't be identical to `wfe()` (which weights by treatment timing and accounts for dynamic timing of treatment effects), but it gives a similar ATE interpretation under parallel trends.

LOL, I'm fixest's author!

ChatGPT: Solution 1

✅ 1. Use `fixest` as a faster alternative

If your estimation can be approximated via high-dimensional fixed effects and clustered SEs, `fixest::feols()` is extremely fast and memory-efficient. Example:

r

Copier Modifier

```r
library(fixest)

# Approximate wfe with two-way fixed effects
fixest_model = feols(y ~ treat_post + x1 | id + period, data = base_example, cluster = ~id)
```

This won't be identical to `wfe()` (which weights by treatment timing and accounts for dynamic timing of treatment effects), but it gives a similar ATE interpretation under parallel trends.

ChatGPT: All solutions

✅ Summary

Optimization	Action
Faster Estimation	Use `fixest::feols()` with high-dimensional FE
Preprocessing	Drop never-treated or always-treated units
Avoid slow SE	Set `hetero.se = FALSE`, `auto.se = FALSE`
Parallelize	Use `future.apply` or `furrr` if looping
Alternative	Use `did` or `fixest::sunab()` for DiD/event-study

ChatGPT: All solutions

✅ Summary

Optimization	Action
Faster Estimation	Use `fixest::feols()` with high-dimensional FE
Preprocessing	Drop never-treated or always-treated units
Avoid slow SE	Set `hetero.se = FALSE`, `auto.se = FALSE`
Parallelize	Use `future.apply` or `furrr` if looping
Alternative	Use `did` or `fixest::sunab()` for DiD/event-study

Summary in lay terms

Don't use the wfe method. All suggestions are off.

Full ChatGPT discussion here: https://chatgpt.com/share/68527bc3-cf70-800b-848b-f4e02924979c

Back in 2020 (when twitter was cool)

- PhD student asking on twitter:★
  - "I'm using Imai and Song weighted fixed-effects estimator."
  - "My estimations take forever. "
  - "Help!"
-

★: I tried but couldn't find the tweet! If by chance you find it, please send it to me!!!

Back in 2020 (when twitter was cool)

- PhD student asking on twitter:★
  - "I'm using Imai and Song weighted fixed-effects estimator."
  - "My estimations take forever. "
  - "Help!"
- I had just landed an AP job (this job!), I had some spare time:
  -
  -

★: I tried but couldn't find the tweet! If by chance you find it, please send it to me!!!

Back in 2020 (when twitter was cool)

- PhD student asking on twitter:★
  - "I'm using Imai and Song weighted fixed-effects estimator."
  - "My estimations take forever. "
  - "Help!"
- I had just landed an AP job (this job!), I had some spare time:
  - I got interested
  -

★: I tried but couldn't find the tweet! If by chance you find it, please send it to me!!!

☆: Maybe I'm bragging here ;-) but it was definitely less than 1h.

Back in 2020 (when twitter was cool)

- PhD student asking on twitter:★
  - "I'm using Imai and Song weighted fixed-effects estimator."
  - "My estimations take forever. "
  - "Help!"
- I had just landed an AP job (this job!), I had some spare time:
  - I got interested
  - 20 min later:☆ same estimation running at least 10 times faster

★: I tried but couldn't find the tweet! If by chance you find it, please send it to me!!!

☆: Maybe I'm bragging here ;-) but it was definitely less than 1h.

Back in 2020 (when twitter was cool)

- PhD student asking on twitter:★
  - "I'm using Imai and Song weighted fixed-effects estimator."
  - "My estimations take forever. "
  - "Help!"
- I had just landed an AP job (this job!), I had some spare time:
  - I got interested
  - 20 min later:☆ same estimation running at least 10 times faster

What was the problem?

★: I tried but couldn't find the tweet! If by chance you find it, please send it to me!!!

☆: Maybe I'm bragging here ;-) but it was definitely less than 1h.

The method

Imai, K. and Song, I
When should we use unit fixed-effects regression models for causal inference with longitudinal data?
2019, **Amercan Journal of Political Science** 63(2): 467-490

Theorem 1 (Within-Unit Matching Estimator as a Weighted Unit Fixed Effects Estimator). *Any within-unit matching estimator $\hat{\tau}$ defined by a matched set $\mathcal{M}_{it}$ equals the weighted linear fixed effects estimator, which can be computed as*

$$\hat{\beta}_{\text{WFE}} = \underset{\beta}{\text{argmin}} \sum_{i=1}^{N} \sum_{t=1}^{T} W_{it} \left\{ (Y_{it} - \overline{Y}_i^*) - \beta (X_{it} - \overline{X}_i^*) \right\}^2, \qquad (24)$$

where $\overline{X}_i^* = \sum_{t=1}^{T} W_{it} X_{it} / \sum_{t=1}^{T} W_{it}$, $\overline{Y}_i^* = \sum_{t=1}^{T} W_{it} Y_{it} / \sum_{t=1}^{T} W_{it}$, *and the weights are given by*

$$W_{it} = D_{it} \sum_{i'=1}^{N} \sum_{t'=1}^{T} w_{it}^{i't'}, \quad \text{where}$$

$$w_{it}^{i't'} = \begin{cases} 1 & \text{if } (i,t) = (i',t') \\ 1/|\mathcal{M}_{i't'}| & \text{if } (i,t) \in \mathcal{M}_{i't'} \\ 0 & \text{otherwise.} \end{cases} \qquad (25)$$

It's a weighted OLS esimation, the key is the *weights*.

The code implementing the weights

- https://github.com/insongkim/wfe/blob/master/src/wfe.c
- function `GenWeightsUnit`, coded in **c**, ≈ 200 lines, does many things

```c
void GenWeightsUnit(int *unit_index, int *time_index, int *tr, int *C_it,
       int *len_data, /* total number of rows in the data */
       int *len_u_index,
       int *len_t_index,
       int *ate, int *att,
       int *verbose,  /* 1 if extra print is needed */
       double *weight) {
  int i, j;
  /* Rprintf("u_index: %d, t_index: %d, len_data: %d\n", *len_u_index,
*len_t_index, *len_data); */

  int** exist = intMatrix(*len_t_index, *len_u_index);
  is_index_exist(unit_index, time_index, len_u_index, len_t_index,
len_data, exist);

  /* #pragma omp parallel for   */
  for (i = 0 ; i < *len_u_index ; i++) {


    if(*verbose && *len_t_index>10) {
      int percent = *len_u_index/10;
      if (i % percent == 0){
  /*  Rprintf("- %d th year calculated:\n", (j+1)); */
  Rprintf(".");
  R_FlushConsole();
      }
    }


    for (j = 0 ; j < *len_t_index ; j++) {
      double w_it[*len_t_index];

      // initialize all elements in w_it to 0
      memset(w_it, 0, sizeof(double)*(*len_t_index));


      int c_it = 0;
      int t_it = 0;

      // initialize c_it and t_it
      int k;
      for (k = 0 ; k < *len_data ; k++) {
  if ( unit_index[k] == (i+1) && time_index[k] == (j+1) ) {
    c_it = C_it[k];
    t_it = tr[k];
    /* Rprintf(" t_it: %d\n", t_it); */
    /* Rprintf("0] unit: %d\n", i+1); */
  }
      }
```

```c
      /* Rprintf("time index exist for unit %d and time %d: %d\n",
  i+1, j+1, is_time_index_exist(unit_index, time_index, i+1,
  j+1, *len_data )); */

      /* if (is_time_index_exist(unit_index, time_index, i+1, j+1,
*len_data)) */
      if (exist[j][i]) {
  if (t_it == 1) { /* ifTRUE(sub[,treat]...) == 1 */
    // count control
    double count_control = 0;
    int k = 0;
    for (k = 0 ; k < *len_data ; k++) {
      if (unit_index[k] == (i+1) && tr[k] == 0)
        count_control++;
    }

    if (count_control > 0) {
      double v_it = 1 / count_control;
      /* Rprintf("1] time: %d\n", j+1); */
      /* Rprintf("5.1] number of control: %f\n", count_control); */
      /* Rprintf("6] v_it: %f\n", v_it); */
      w_it[j] = 1;

      for (k = 0 ; k < *len_data ; k++) {
        if (unit_index[k] == (i+1) && tr[k] == 0) {
    int t_index = time_index[k] - 1;
    // Rprintf("2] opposite treatment index: %d\n", t_index+1);
    w_it[t_index] = v_it;

        }
      }

    }
    /* PdoubleArray(w_it, *len_t_index); */
    if (*ate == 1) {
      int k = 0;
      for (k = 0 ; k < *len_t_index ; k++)
        weight[k*(*len_u_index)+i] = weight[k*(*len_u_index)+i] +
(w_it[k] * c_it);
    }
    else if (*att == 1) {
      int k = 0;
      for (k = 0 ; k < *len_t_index ; k++)
        weight[k*(*len_u_index)+i] = weight[k*(*len_u_index)+i] +
(w_it[k] * c_it * t_it);
    }
  }
```

```c
else if (t_it == 0) { /* ifTRUE(sub[,treat]...) == 0 */
    // count treate
    double count_treat = 0;
    int k = 0;
    for (k = 0 ; k < *len_data ; k++) {
      if (unit_index[k] == (i+1) && tr[k] == 1)
        count_treat++;
    }

    if (count_treat > 0) {
      double v_it = 1 / count_treat;
      /* Rprintf("3] j: %d\n", j+1); */
      /* Rprintf("5.2] number of treated: %f\n", count_treat); */
      /* Rprintf("6] v_it: %f\n", v_it); */
      w_it[j] = 1;

      for (k = 0 ; k < *len_data ; k++) {
        if (unit_index[k] == (i+1) && tr[k] == 1) {
    int t_index = time_index[k] - 1;
    /* Rprintf("4] opposite treatment index: %d\n", t_index+1); */
    w_it[t_index] = v_it;
        }
      }
    }
    /* PdoubleArray(w_it, *len_t_index); */
    if (*ate == 1) {
      int k = 0;
      for (k = 0 ; k < *len_t_index ; k++)
        weight[k*(*len_u_index)+i] = weight[k*(*len_u_index)+i] +
(w_it[k] * c_it);
    }
    else if (*att == 1) {
      int k = 0;
      for (k = 0 ; k < *len_t_index ; k++)
        weight[k*(*len_u_index)+i] = weight[k*(*len_u_index)+i] +
(w_it[k] * c_it * t_it);
    }

  }


      }
    }
  }

  FreeintMatrix(exist, *len_t_index);
}
```

New version

LB's implementation:

- tailored to the task
- coded in **R**, ≈ 20 lines, does a single thing

```r
unit_weights = function(index){
  # index: data.frame with two columns: unit and treatment indicator (treat x post)

  # for easy aggregations
  library(data.table)

  res = as.data.table(index)
  names(res) = c("unit", "treatment")

  # we compute the weigts as in Theorem 1 (p. 427)
  res[, n_treated := sum(treatment), by = unit]
  res[, n_not_treated := sum(1 - treatment), by = unit]
  res[, m_size := treatment * n_not_treated + (1 - treatment) * n_treated]
  res[, w := 1 + treatment * (m_size / n_treated) +
              (1 - treatment) * (m_size / n_not_treated)]
  res[is.na(w), w := 0]

  # we return the weights
  res$w
}
```

Side to side comparison

```
library(wfe)
library(fixest)
library(data.table)

# example data set
data(base_did, package = "fixest")
set.seed(0)
base_example = as.data.table(base_did)
id_first = sample(108, 20) ; id_last = sample(108, 20)
base_example = base_example[!(id %in% id_first & period <= 3) & !(id %in% id_last &
period >= 7)]
base_example[, treat_post := treat*post]

# wfe V1 vs V2
microbenchmark(
  wfe_v1 = wfe(y ~ treat_post + x1, data = base_example, treat = "treat_post",
               unit.index = "id", time.index = "period", method = "unit",
               qoi = "ate", hetero.se = TRUE, auto.se = TRUE),

  wfe_v2 = feols(y ~ treat_post + x1 | id, base_example,
                 weights = unit_weights(base_example[, .(id, treat_post)])),
  times = 4)
#> Unit: milliseconds
#>    expr      min       lq      mean    median        uq      max neval
#>  wfe_v1 119.9691 122.8909 125.53608 126.92805 128.18125 128.3191     4
#>  wfe_v2  10.4454  11.3273  11.79133  12.21255  12.25535  12.2948     4
```

Understanding the speed difference

Q: V2 is about 10 times faster than V1. Why?

Understanding the speed difference

Q: V2 is about 10 times faster than V1. Why?

- to handle the generality (i.e. handle more than the case of *Theorem 1*), V1's function loops through each unit (and doing so it loops several times across the full data set[1])
- V2 is tailored for *Theorem 1* and takes advantage of all possible simplifications associated to this data structure. Getting the simplifications required a bit of conceptual thinking

[1]: the electrons are tired fetching the data!

Takeaway

Rule 2: Never do more than what you need!

How to make fast algorithms? Summing up

How to make fast algorithms?

Rule 0.a: Open your eyes

You cannot fix a problem that you do not see!
So open the box! Do not accept existing code without criticism.
You can improve existing algorithms, and you should.

How to make fast algorithms?

Rule 0.a: Open your eyes

You cannot fix a problem that you do not see!
So open the box! Do not accept existing code without criticism.
You can improve existing algorithms, and you should.

Rule 0.b: Do not trust AI

Do note take AI answers as definitive answers!
This is tempting, but don't!
AI models use an **authoritative tone that makes you feel dumb and meek.**
It's part of their game because they want a monopoly on "thinking", and you win market power by discouraging competitors (you).
Never forget that you're smart, so use your brain!★

★: even when the problem is diffcult.

How to make fast algorithms?
Continued

Rule 1: Shop efficiently in the RAM's warehouse

- vectorize as much as possible (i.e. shop in the same aisle)
- try to avoid row based algorithms

How to make fast algorithms?
Continued

Rule 1: Shop efficiently in the RAM's warehouse

- vectorize as much as possible (i.e. shop in the same aisle)
- try to avoid row based algorithms

Rule 2: Never do more than what you need

- always try to tailor the algorithm to the problem at hand, not the general case
- always think conceptually first, only then start to code
- low level languages (like C, C++, Rust, Julia) do not compensate inefficient algorithms

Thank you for your attention!

Appendix

Bonus: Productivity tips

Bonus 1: The future of typesetting

The future of typesetting

1. take Latex
2.
3.
4.
5.

The future of typesetting

1. take Latex
2. remove all the problems with it
3.
4.
5.

The future of typesetting

1. take Latex
2. remove all the problems with it
3. keep the best of it
4.
5.

The future of typesetting

1. take Latex
2. remove all the problems with it
3. keep the best of it
4. improve it
5.

The future of typesetting

1. take Latex
2. remove all the problems with it
3. keep the best of it
4. improve it
5. you end up with typst!

typst★ may be the best thing that happened to typesetting in ~~decades~~ ever

★: https://typst.app/ + VSCode extension tinymist

OK what's the fuss?

typst key features (among other):

- instant compile time
- compilation issues are flagged on the spot
- great documentation
- it's a proper programming langage (variables/functions/loops/conditions)
- anything you have in mind can be done

In 5-6 years time, it will completely supersede Latex.
We just need to wait for the current editors to retire 😉

OK what's the fuss?

typst key features (among other):

- instant compile time
- compilation issues are flagged on the spot
- great documentation
- it's a proper programming langage (variables/functions/loops/conditions)
- anything you have in mind can be done

In 5-6 years time, it will completely supersede Latex.
We just need to wait for the current editors to retire 😉

Catch the train to the future now!

Bonus 2: Reproductibility made easy

The context

Typical research project

- takes years
- thousands of lines of code
- dozens and dozens of files
- it's a hot mess

Problem

You found a problem in a variable: you clean it differently now:
=> *many results should be changed downstream*

What do you do?

Solutions to the problem

Problem

You found a problem in a variable: you clean it differently now:
=> *many results should be changed downstream*

What do you do?

Solution 1

You run **all** the existing code

Problems:

- P1: you need to track the specific order in which the code needs to be run
- P2: can be very slow if code takes a while to run

Solution 2

You run only the code that explicily depends on your changes.

Problems:

- P1: you need to track the specific order in which the code needs to be run
- P3: you need to precisely map the code dependencies

rmake

rmake: New R-package to handle all this:

- automatic dependency detection (files/functions)
- detects when code is changed (ignore comments) and automatically updates what needs to be updated – and nothing more
- you★ don't need to care about anything! It's all automatic!

★: almost

rmake

`rmake`: New R-package to handle all this:

- automatic dependency detection (files/functions)
- detects when code is changed (ignore comments) and automatically updates what needs to be updated – and nothing more
- you★ don't need to care about anything! It's all automatic!

How does it work?

- the code is divided into independent code chunks
- you declare a chunk with a starting `=` in a header comment:☆
  `#### = This is a chunk ####`
- file dependencies are statically deduced from I/O functions

★: almost

☆: header comment = comment ending with four hashes

rmake example

```
####
#### = main data set ####
####

set.seed(1)

n = 100
base = data.frame(x = rnorm(n))
base$y = 2 + 0.5 * base$x + rnorm(n)

saveRDS(base, "rmake-example/_DATA/data-example.rds")

####
#### = OLS estimation ####
####

base = readRDS("rmake-example/_DATA/data-example.rds")

library(fixest)

est = feols(y ~ x, base)
etable(est, export = "rmake-example/images/EST_yx.png")
```

- 2 chunks
- any change in the data creation leads to change in the estimation
- run `rmake()` and everything is taken care of

rmake example

1. Create file with the previous code, run `rmake`:

```
rmake::rmake()
#> The following chunks need to be run:
#> file            chunk          new code fun input output indirect
#> rmake-example.R main data set   X
#> rmake-example.R OLS estimation  X                           X
> Proceed (y, yes; anything else: no)?  y
#> rmake-example.R@01: main data set ... < 0.1s
#> rmake-example.R@02: OLS estimation ... 3.2s
rmake::rmake()
#> All code chunks are up to date.
```

rmake example

1. Create file with the previous code, run `rmake`:

```
rmake::rmake()
#> The following chunks need to be run:
#> file            chunk          new code fun input output indirect
#> rmake-example.R main data set   X
#> rmake-example.R OLS estimation  X                          X
> Proceed (y, yes; anything else: no)?  y
#> rmake-example.R@01: main data set ... < 0.1s
#> rmake-example.R@02: OLS estimation ... 3.2s
rmake::rmake()
#> All code chunks are up to date.
```

2. Modify code in "main data set", then run `rmake`:

```
rmake::rmake()
#> The following chunks need to be run:
#> file            chunk          new code fun input output indirect
#> rmake-example.R main data set       X
#> rmake-example.R OLS estimation                             X
#> Previous run-time for 2 existing chunks: 3.2s
> Proceed (y, yes; anything else: no)?  y
#> rmake-example.R@01: main data set ... < 0.1s
#> rmake-example.R@02: OLS estimation ... 1.6s
```

rmake

Available on Github:

https://github.com/lrberge/rmake

```
remotes::install_github("https://github.com/lrberge/rmake")
```

Bonus 3: New R console

sircon

I created a new R console for Windows (later it will be ported to linux/MacOS)

sircon: Simple R console

https://github.com/lrberge/sircon

Although it's still in development, it currently works, and will be improved!

Thanks for your attention!

Have fun!