

Algoritmen & Heuristieken

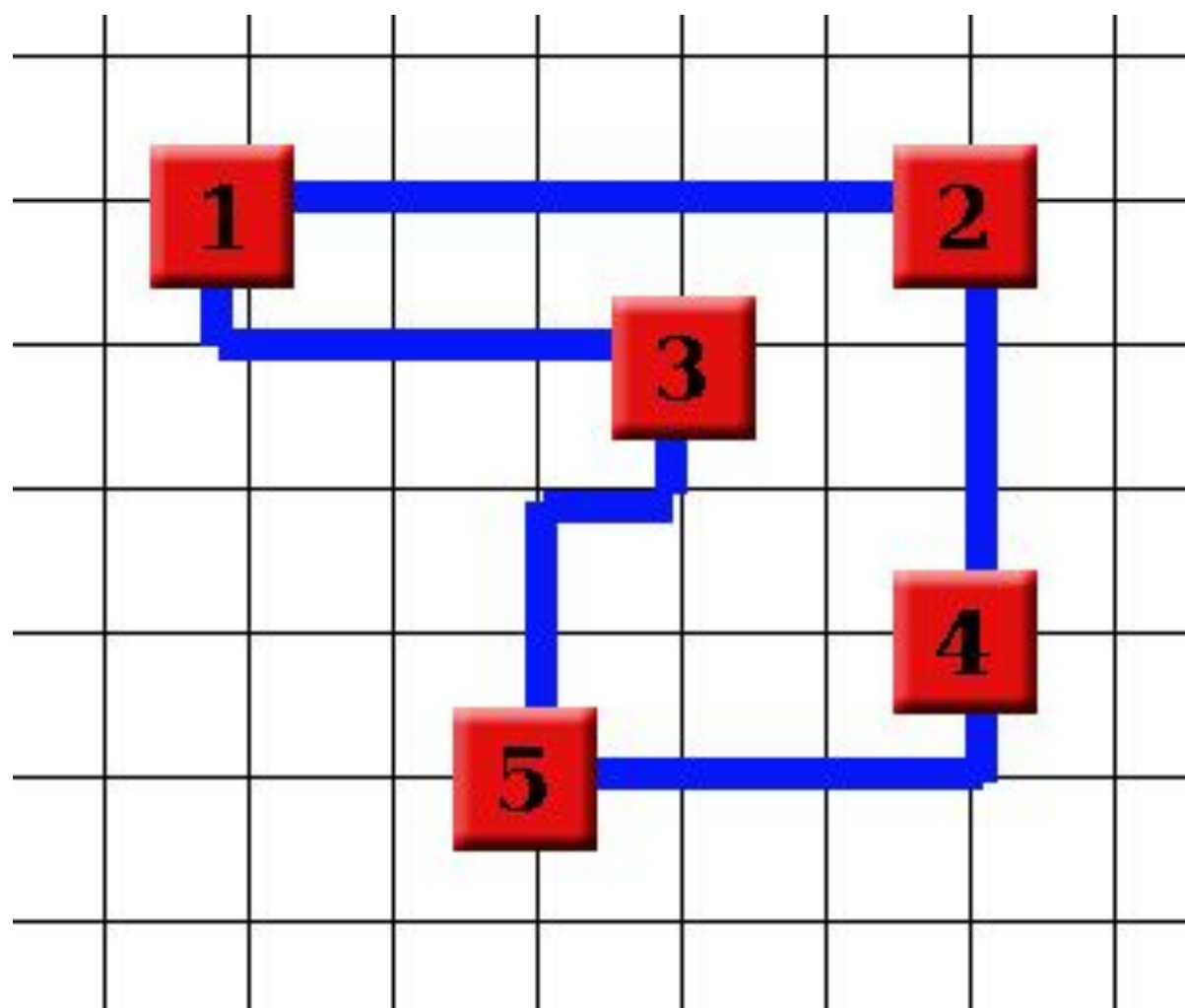
Experimenteren met optimaliseren

Jelle van Assema
proglab.nl

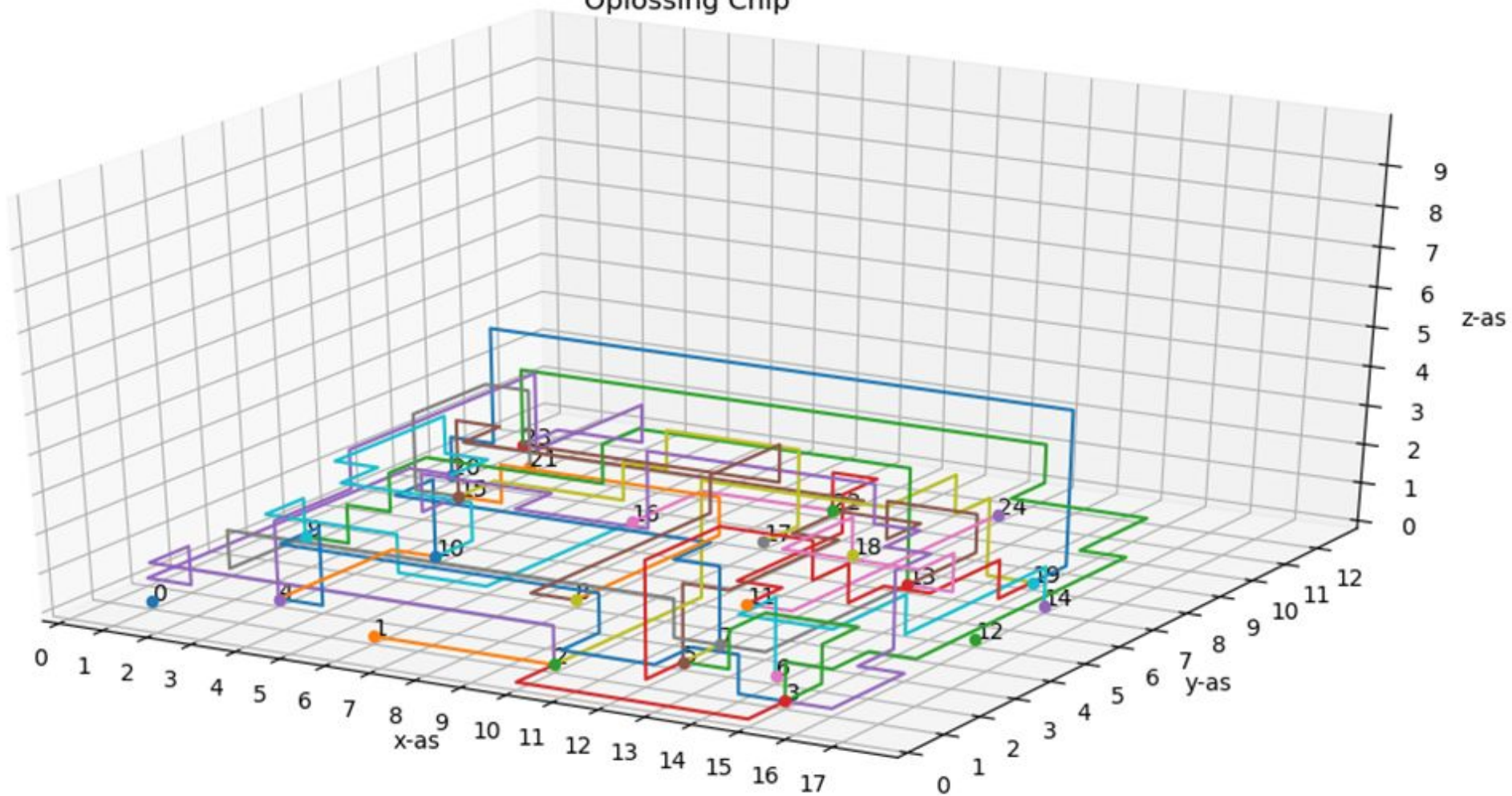
Experimenteren

Presenteren

The assignment is to implement all nets
in all netlists at minimum cost.



Oplossing Chip



$$C = 578$$

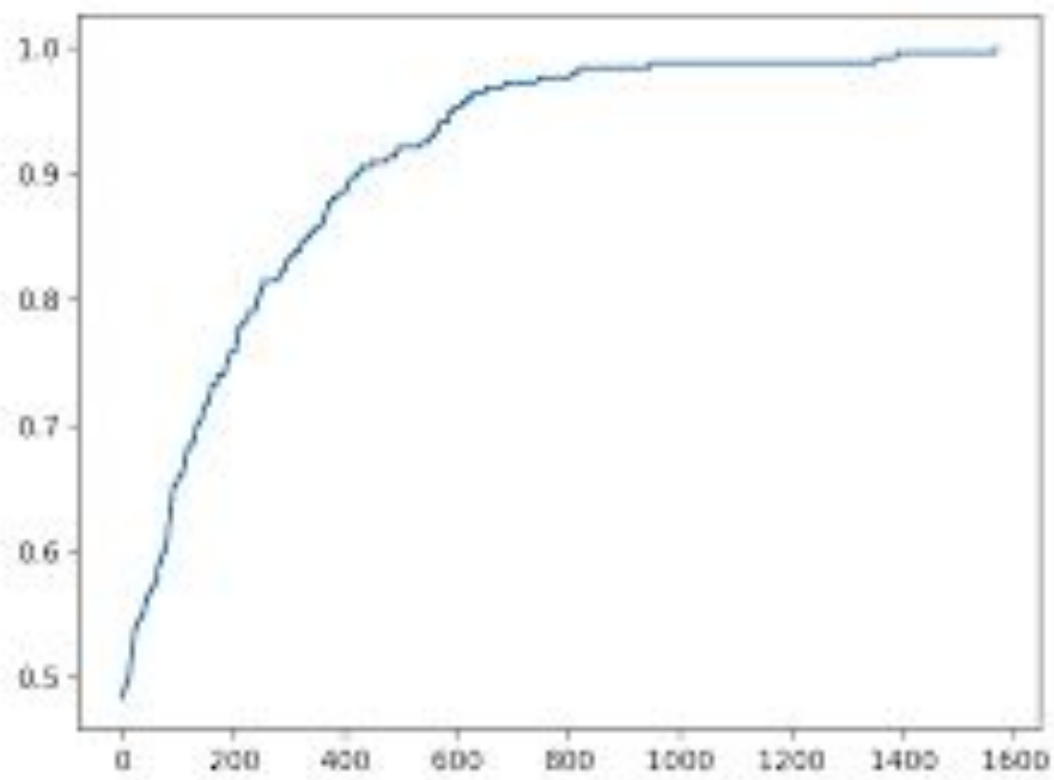
Optimaal antwoord

Waarschijnlijk niet

$$C = 42$$

What was the ~~question~~ algorithm?

Om precies **00:37:31:678**
op **11/06/2023** produceerde
random_baseline.py
een grid met
 $C = 42$



~~Global Optimum~~

∞

Random

~~The assignment is to implement all nets
in all netlists at minimum cost.~~

What is a good algorithm to implement all nets in all
netlists at minimum cost?

Wat is een goed algoritme?

Correct

Optimaal

Efficiënt

Consistent

...

Wat is een goed algoritme?

Today

14°

6°



 **78%**

Thu 20

17°

11°



 24%

Fri 21

16°

9°



 70%

Sat 22

13°

7°



 85%

Sun 23

14°

8°



 64%

Wat is een goed algoritme?

HINT



Wat is een goed
optimalisatie-algoritme?

Efficiëntie

Optimaal antwoord

Score

Waarschijnlijk niet

Efficiëntie

Algoritme A

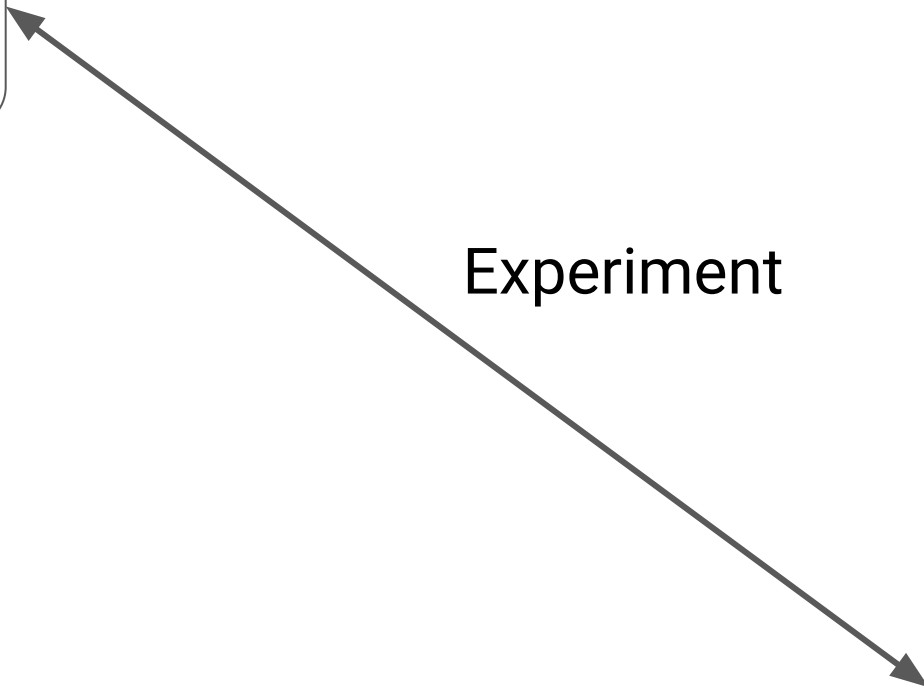
Wat is een beter
optimalisatie-algoritme?

Algoritme B

Algoritme A

Experiment

Algoritme B



Resultaten

Conclusie

Experiment

Evaluatie

Algoritme A

Wat is een beter
optimalisatie-algoritme?

Algoritme B

Experimenteren met algoritmes

Experimenteren met algoritmes

```
while(change > 1) {  
    coins += 1;  
    change -= 1;  
}
```

```
coins += change;  
change = 0;
```


Experimenteren met algoritmes



~~Experimenteren met algoritmes~~

Experimenteren met
implementaties van algoritmes

Implementaties van algoritmes

```
def sum_of_numbers_between(a, b):  
    return sum(range(a, b))
```

$$C = 578$$

Praktisch onmogelijk om correctheid te bewijzen

Heisenbugs

Antwoord
onbekend

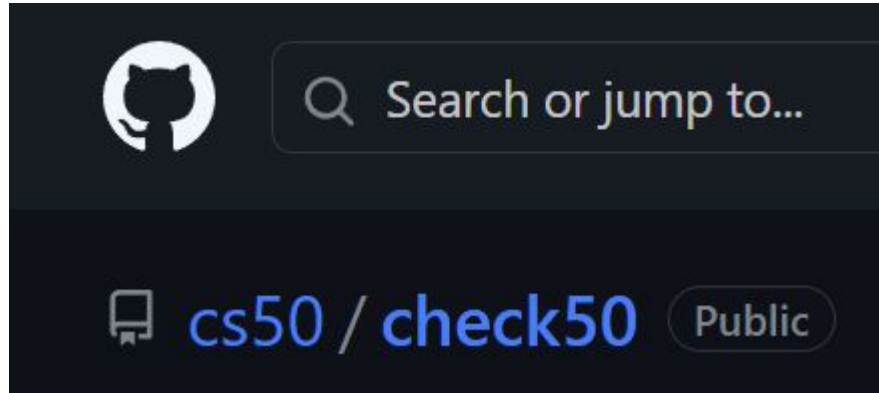
Mensen

~~Experimenteren met algoritmes~~

~~Experimenteren met
implementaties van algoritmes~~

Experimenteren met **buggy**
implementaties van algoritmes

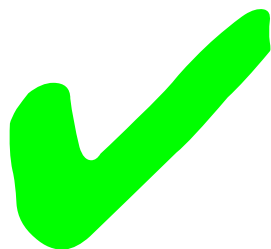
(unit)tests

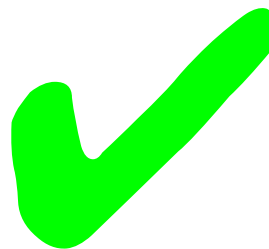
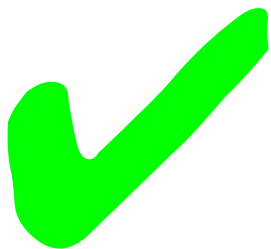


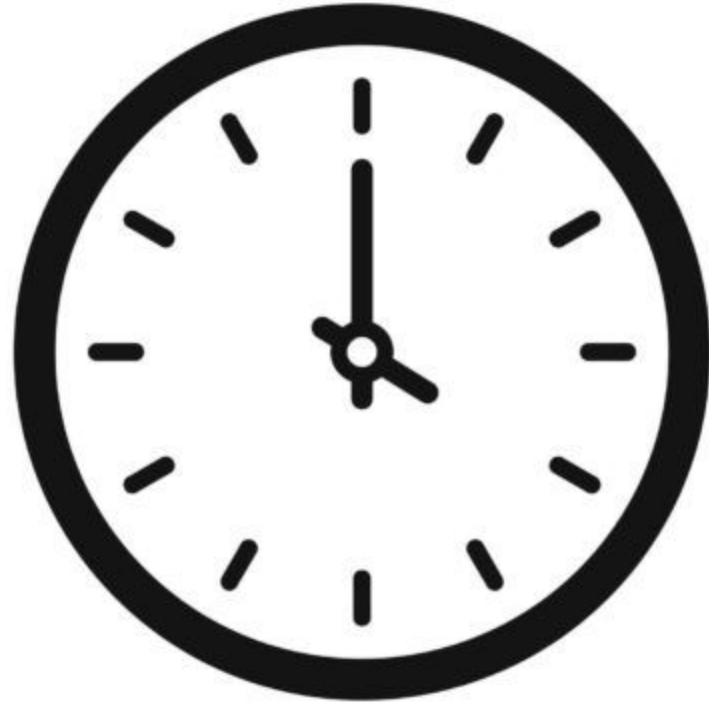


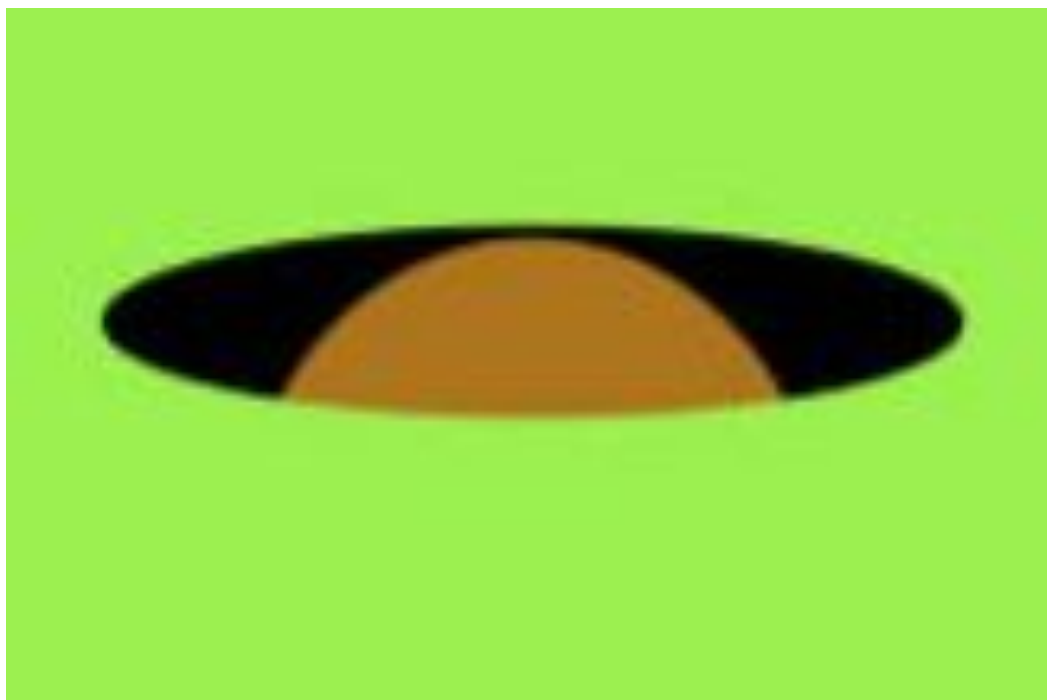


```
def spawn(self, cmd, env=None):  
    """Spawns a new child process."""  
    if self._valgrind:  
        self.log.append("running valgrind {}".format(cmd))  
        cmd = "valgrind --show-leak-kinds=all --xml=yes --xml-file={} -- {}".format(  
            os.path.join(self.dir, self._valgrind_log), cmd)
```









Vertrouwen is **niet** goed,
controle is nodig

(unit)tests

score-functie

(unit)tests

score-functie

(unit)tests

representatie

score-functie

(unit)tests

algoritmes

representatie

(unit)tests

Nu. Anders is het te laat.

check50

De kans dat we het allebei fout doen
is kleiner toch?

check50

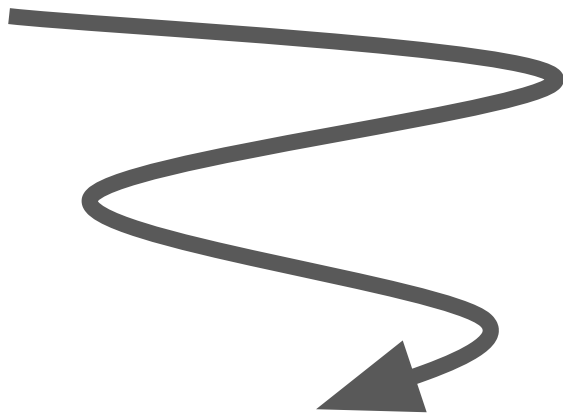
De kans dat we het allebei fout doen

Jelleas fixed smartgrid example ...

on Jan 18 ↻ 207

IS kleiner toch?

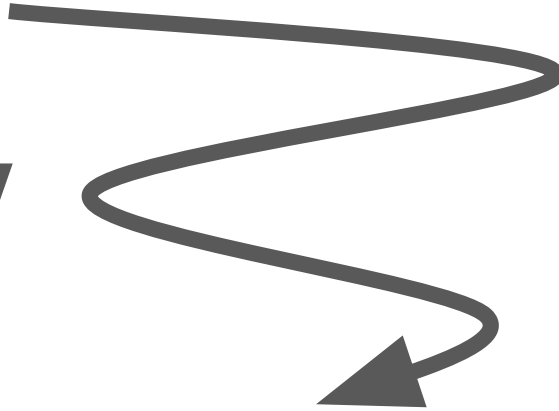
Experimenteren met **buggy** **implementaties** van algoritmes



Is algoritme A beter dan algoritme B?

Repeatability

Reproducibility



Replicability

Reproducibility

- Deel de code
- Deel de input
- Documenteer hoe de code is ingezet op de input
- Geef de resultaten
- Geef de tools om de resultaten te visualiseren

Reproducibility tips

Schrijf scripts voor de experimenten.

Reproducibility tips

Liever te veel data, dan te weinig.

Reproducibility tips

"Ask yourself: If I put my data on my website and someone else downloads it, does every single file contain sufficient information to understand its content?"[1]

Reproducibility tips

Gebruik een “seed” bij random algoritmes.

Reproducibility tips

Schrijf scripts voor het visualiseren van resultaten.

Reproducibility tips

requirements.txt

numpy==1.24.3

Testing & Reproducibility

Begin nu: minder werk & beter resultaat

Code lever je in

Schrijf scripts

Efficiëntie

Optimaal antwoord

Score

Waarschijnlijk niet

Efficiëntie

Optimale algoritmes vergelijken

```
for problem in problems:  
    for _ in range(X):  
        algorithm_a(problem)
```

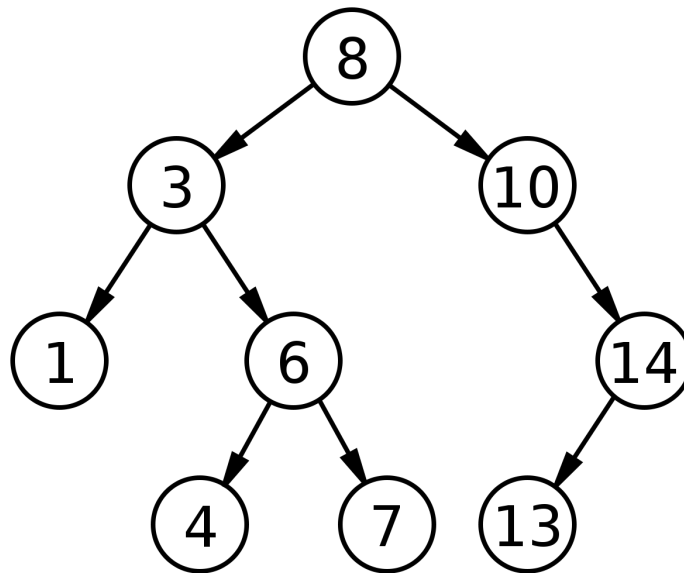
```
for problem in problems:  
    for _ in range(X):  
        algorithm_b(problem)
```

Optimale algoritmes vergelijken

Tijd

Ruimte

Tijd



Wall time



```
import time
start = time.time()
...
end = time.time() - start
```

Wall time



- Meet implementatie, niet algoritme
- Afhankelijk van hardware
- Makkelijk te begrijpen
- Goed in praktijk

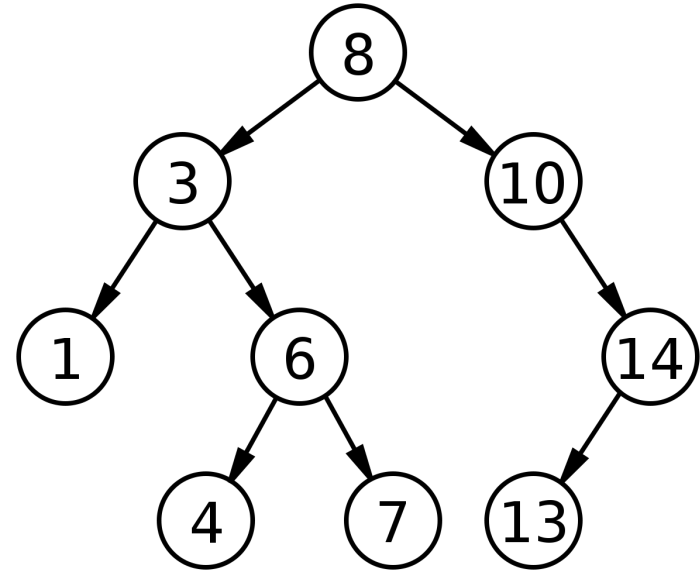
Wall time



- **Dezelfde computer**
- **Rapporteer specs computer**
- **Deel de code**
- **Gebruik geen profiler**

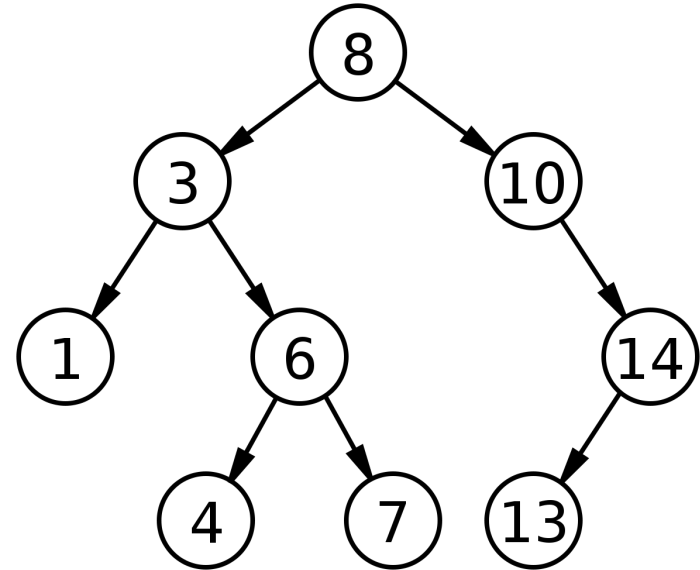
Aantal evaluaties

```
evaluation_count = 0  
def evaluate(state):  
    evaluation_count += 1  
    ...
```

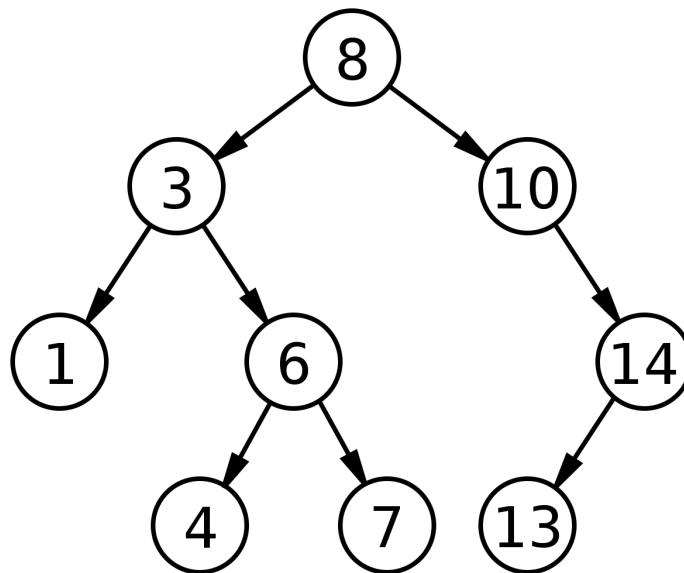


Aantal evaluaties

- Meet algoritme, niet implementatie
- Onafhankelijk van hardware
- Geen directe relatie met tijd
 - Tijd per state verschilt enorm
- Goed in theorie



Tijd: doe het allebei



Optimale algoritmes vergelijken

```
for problem in problems:
    for _ in range(X):
        algorithm_a(problem)
    ..
    file.write(
        f"{time} {evals}\n"
    )
```

```
for problem in problems:
    for _ in range(X):
        algorithm_b(problem)
    ..
    file.write(
        f"{time} {evals}\n"
    )
```

Efficiëntie

Optimaal antwoord

Score

Waarschijnlijk niet

Efficiëntie

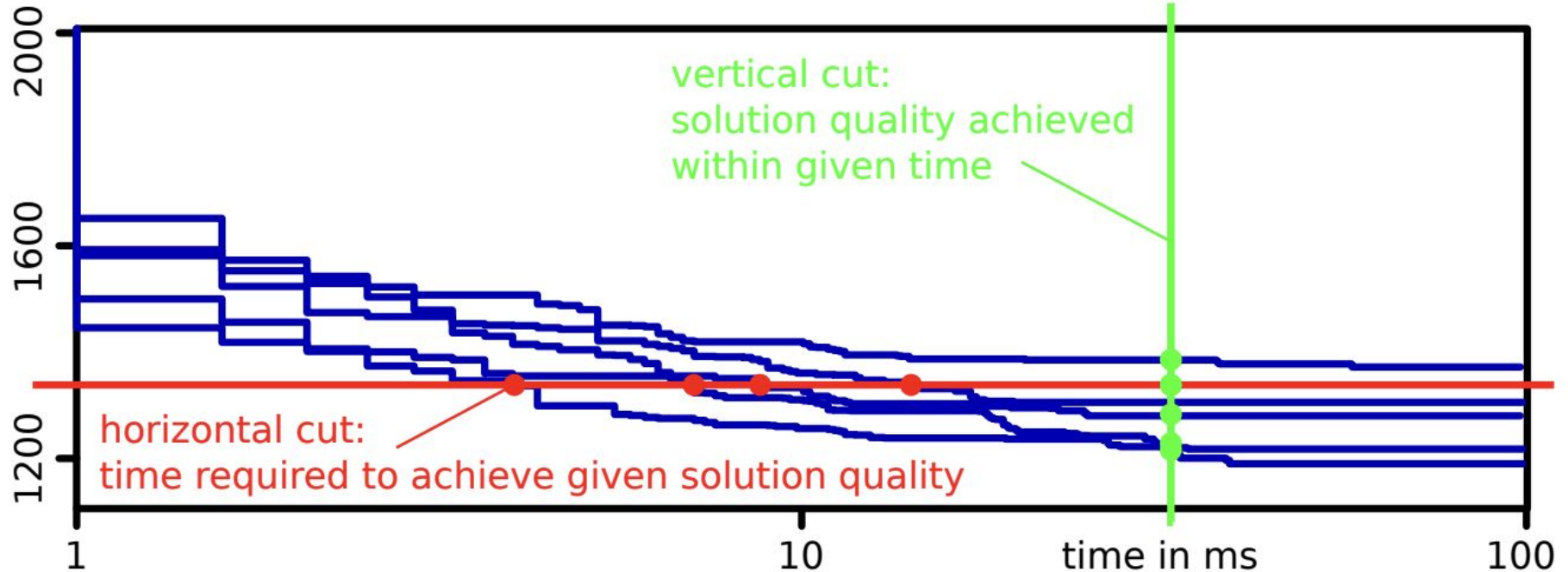
Niet optimale algoritmes vergelijken

Tijd

Versus

Score

Niet optimale algoritmes vergelijken



Restrictie op tijd (vertical cut)

Goed voor praktijk: binnen X tijd wordt Y resultaat behaald

Vergelijken op score is lastig te interpreteren

```
if end_time - start_time >= X:  
    return best_solution_so_far
```

Restrictie op kwaliteit (horizontal cut)

Goed voor theorie: X resultaat wordt in Y tijd behaald

Vergelijken met tijd is makkelijker te interpreteren

```
if evaluate(best_solution_so_far) >= X:  
    return best_solution_so_far
```


Niet optimale algoritmes vergelijken

Doe zowel horizontaal als verticaal

"Cuts" zijn afhankelijk van case en context

Meerdere "cuts" ook mogelijk!

Liever te veel data dan te weinig

Niet optimale algoritmes vergelijken

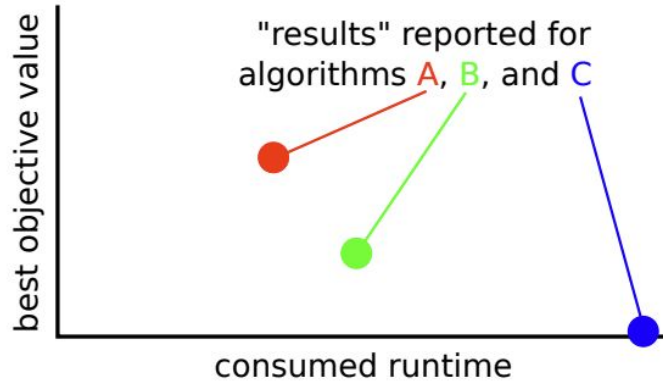


Figure 4.4: “End results” experiments with algorithms versus how the algorithms could actually have performed.

Niet optimale algoritmes vergelijken

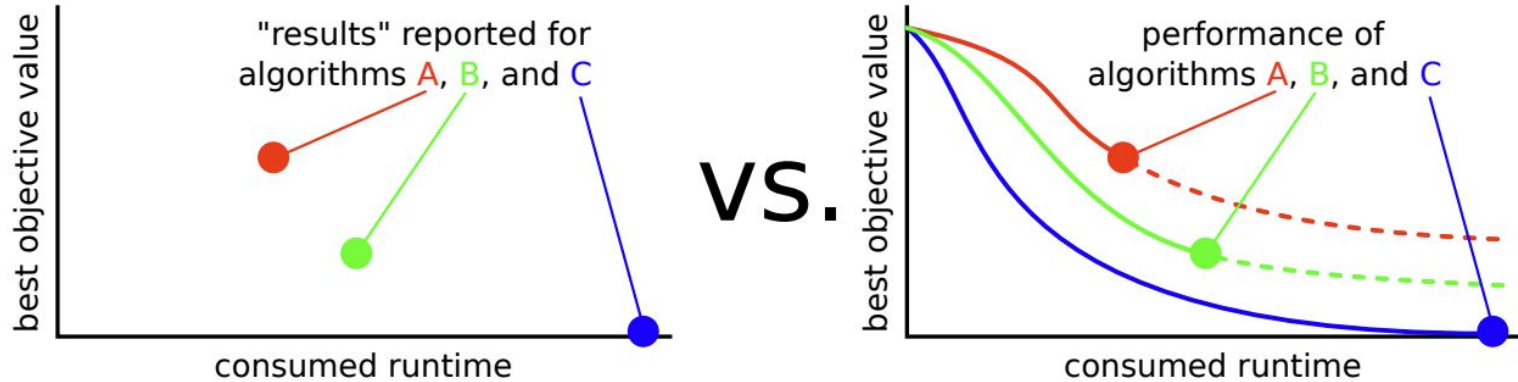


Figure 4.4: “End results” experiments with algorithms versus how the algorithms could actually have performed.

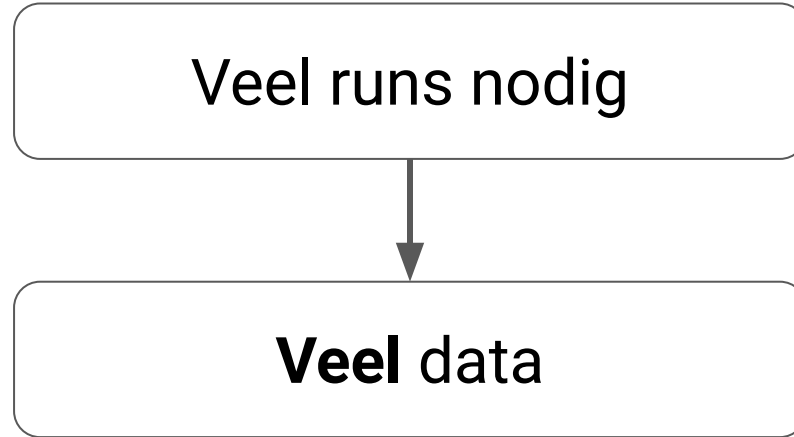
Resultaten

Conclusie

Experiment

Evaluatie

Randomness



Samenvatting nodig

Resultaten

...300 250 1000 275 190 320 195 270 280 350...

Resultaten

...300 250 1000 275 190 320 195 270 280 350...

best = max(results) = 1000

Resultaten

...300 250 1000 275 190 320 195 270 280 350...

`best = max(results) = 1000`

`mean = sum(results) / len(results) = 343`

Resultaten

...300 250 1000 275 190 320 195 270 280 350...

`best = max(results) = 1000`

`mean = sum(results) / len(results) = 343`

`median = sort(results)[len(results) // 2] = 275`

Resultaten - spreiding

...300 250 1000 275 190 320 195 270 280 350...

Resultaten - spreiding

...300 250 1000 275 190 320 195 270 280 350...

```
variance = 0
for result in results:
    variance += (result - mean(results))**2
variance /= len(results)
```

Resultaten - spreiding

...300 250 1000 275 190 320 195 270 280 350...

```
variance = 0
for result in results:
    variance += (result - mean(results))**2
variance /= len(results)

standard_deviation = variance**0.5
```

Resultaten - spreiding

...300 250 1000 275 190 320 195 270 280 350...

```
from statistics import variance, stdev
```

Resultaten - spreiding

...300 250 1000 275 190 320 195 270 280 350...

```
variance = 0
```

```
for result in results:
```

```
    variance += (result - mean(results))**2
```

```
variance /= len(results)
```

```
standard_deviation = variance**0.5
```

Resultaten - spreiding

...300 250 1000 275 190 320 195 270 280 350...

```
sorted_results = sort(results)
n_results = len(results)
quartiles = [
    sorted_results[int(n_results * 0.25)]
    sorted_results[int(n_results * 0.5)]
    sorted_results[int(n_results * 0.75)]
]
```

Resultaten

Conclusie

Experiment

Evaluatie

Evaluatie

...300 250 1000 275 190 320 195 270 280 350...

...350 900 1100 760 600 260 340 750 570 350...

Evaluatie

...300 250 1000 275 190 320 195 270 280 350...

...350 900 1100 760 600 260 340 750 570 350...

best(res_a) = 1000

best(res_b) = 1100

median(res_a) = 275

median(res_b) = 570

quintile_90(res_a) = 350

quintile_90(res_b) = 900

Evaluatie

...300 250 1000 275 190 320 195 270 280 350...

...350 900 1100 760 600 260 340 750 570 350...

```
>>> from scipy.stats import mannwhitneyu  
>>> print(mannwhitneyu(res_a, res_b))  
0.013875
```

Resultaten

Conclusie

Experiment

Evaluatie

Conclusie

Conclusie

- Kijk uit met claims
- Buggy implementations of algorithms
- Under promise, over deliver
- Geef een oplossing voor de case

Just one more thing...

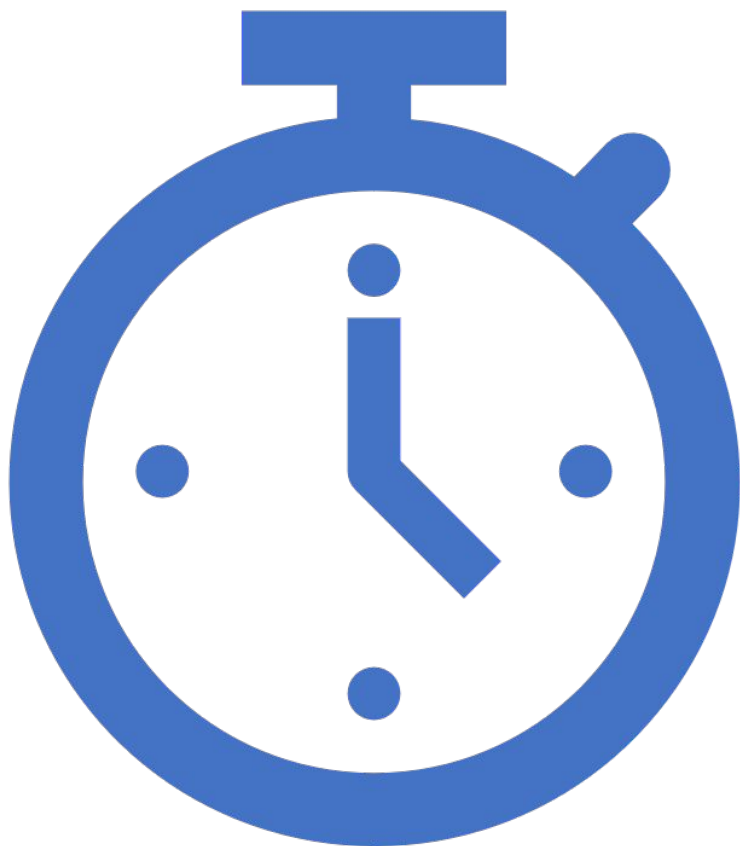


Just one more thing...

One Factor At a Time

Grid Search

Presenteren



Case 2 min

State Space 1 min

Methode 2-3 min

Resultaten 3 min

Conclusie 1-2 min

Wat is het probleem?

Case

Wat is het probleem?

Terminologie

Case

Wat is het probleem?

Terminologie

Case

Wat is een oplossing?

Wat is het probleem?

Terminologie

Case

2 min

Wat is een oplossing?



The background of the image is a deep space scene. It features a dark blue to black background filled with numerous small, bright white stars. There are also larger, more diffuse nebulae or gas clouds in shades of blue and purple, some with bright, glowing centers. The overall effect is a sense of vastness and cosmic wonder.

State Space

BFS
Beam Search
Genetic Algorithm

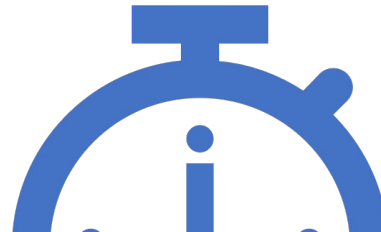
Methode (algorithmes)

BFS + Case = ?

BFS
Beam Search
Genetic Algorithm

Methode (algorithmes)

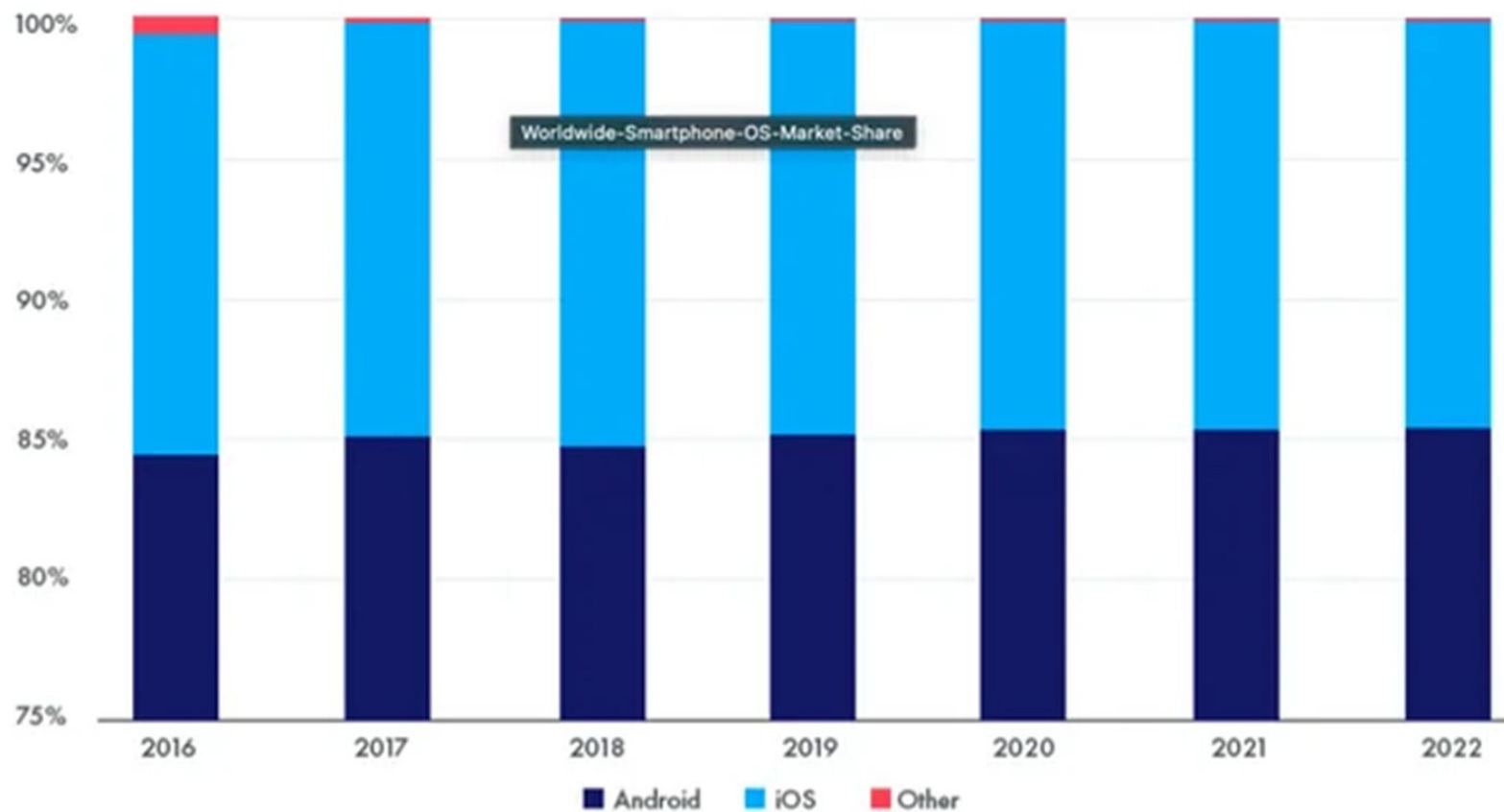
BFS + Case = ?

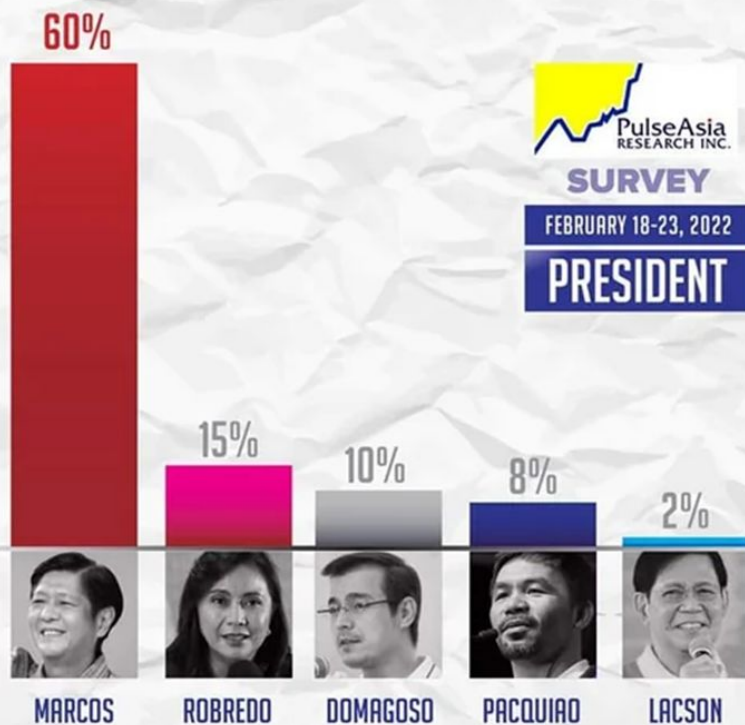


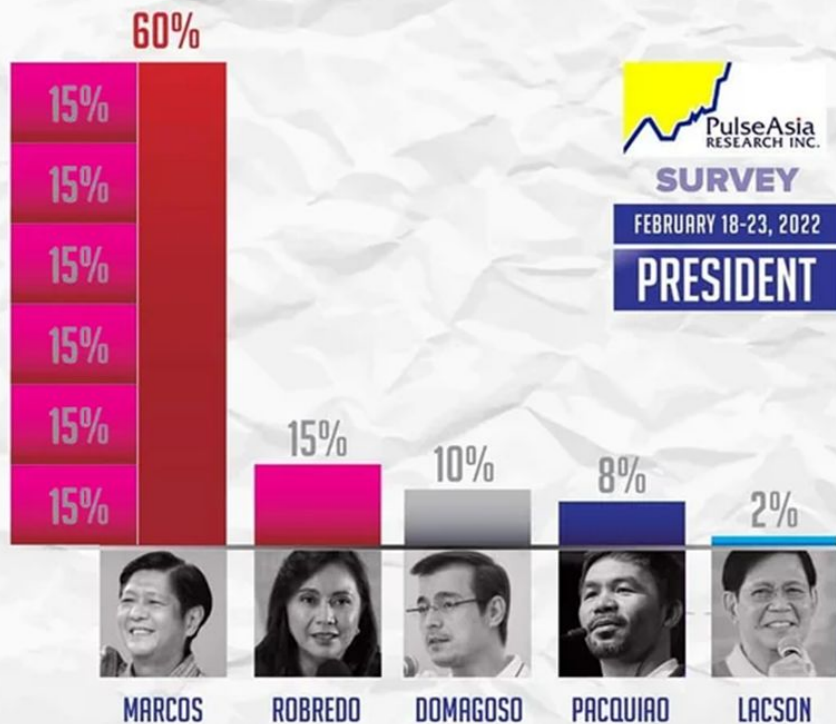
2-3 min

Resultaten

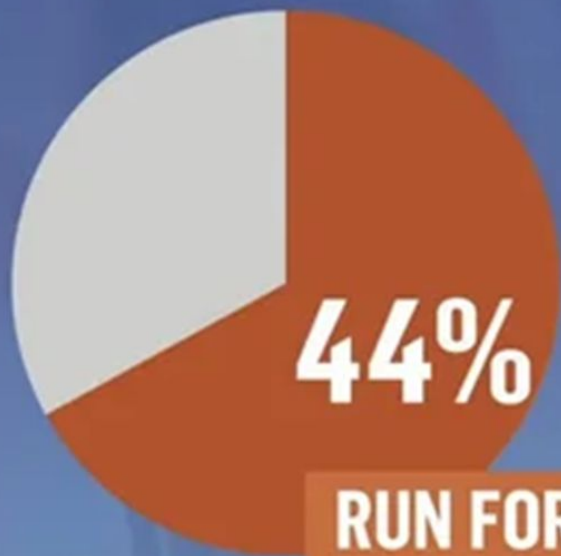
Worldwide Smartphone OS Market Share







REPUBLICANS & TRUMP



RUN FOR PRESIDENT

SOURCE: PEW RESEARCH CENTER

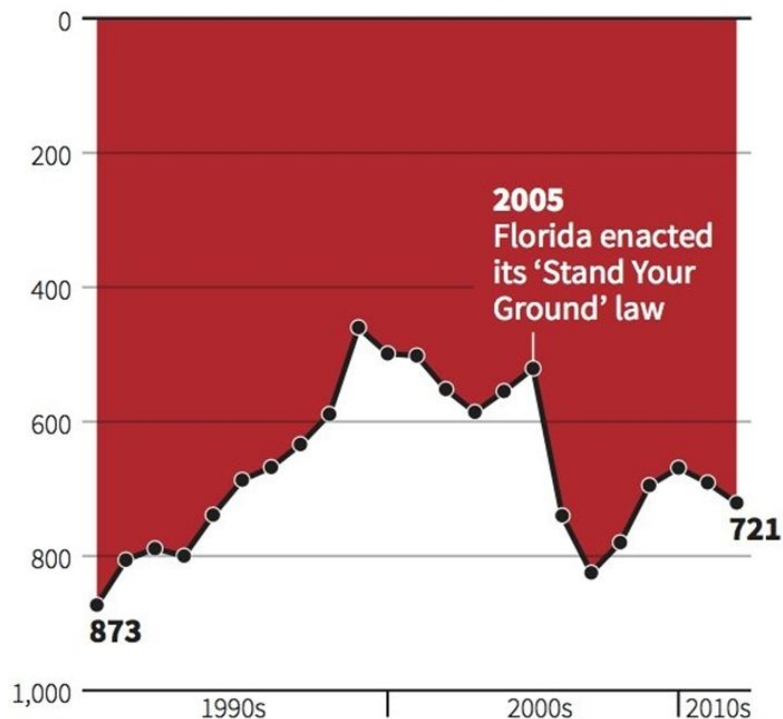


POLL: 44% OF REPUBLICANS WANT TRUMP TO RUN AGAIN

SUNDAY TODAY
with Willie Geist

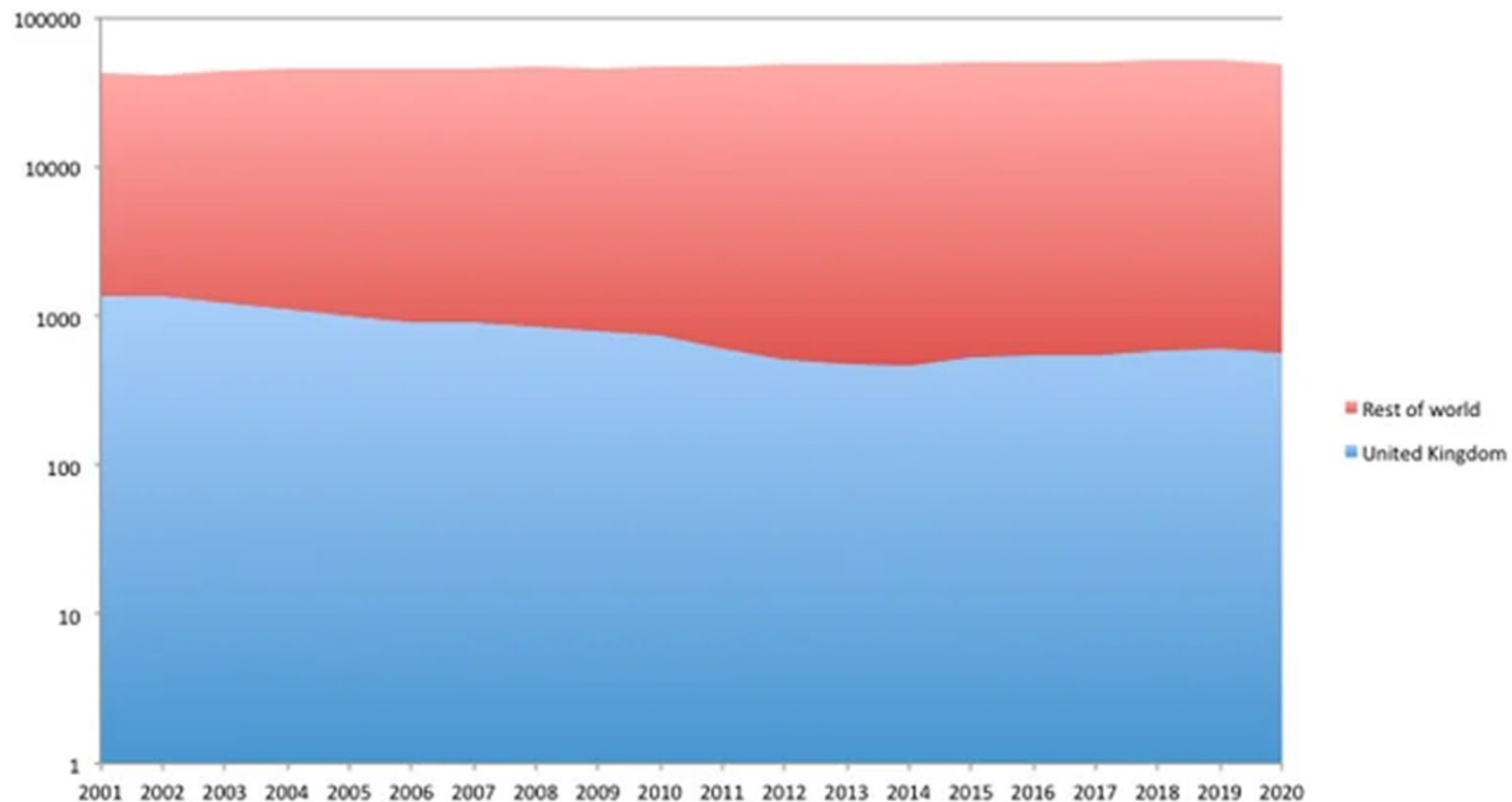
Gun deaths in Florida

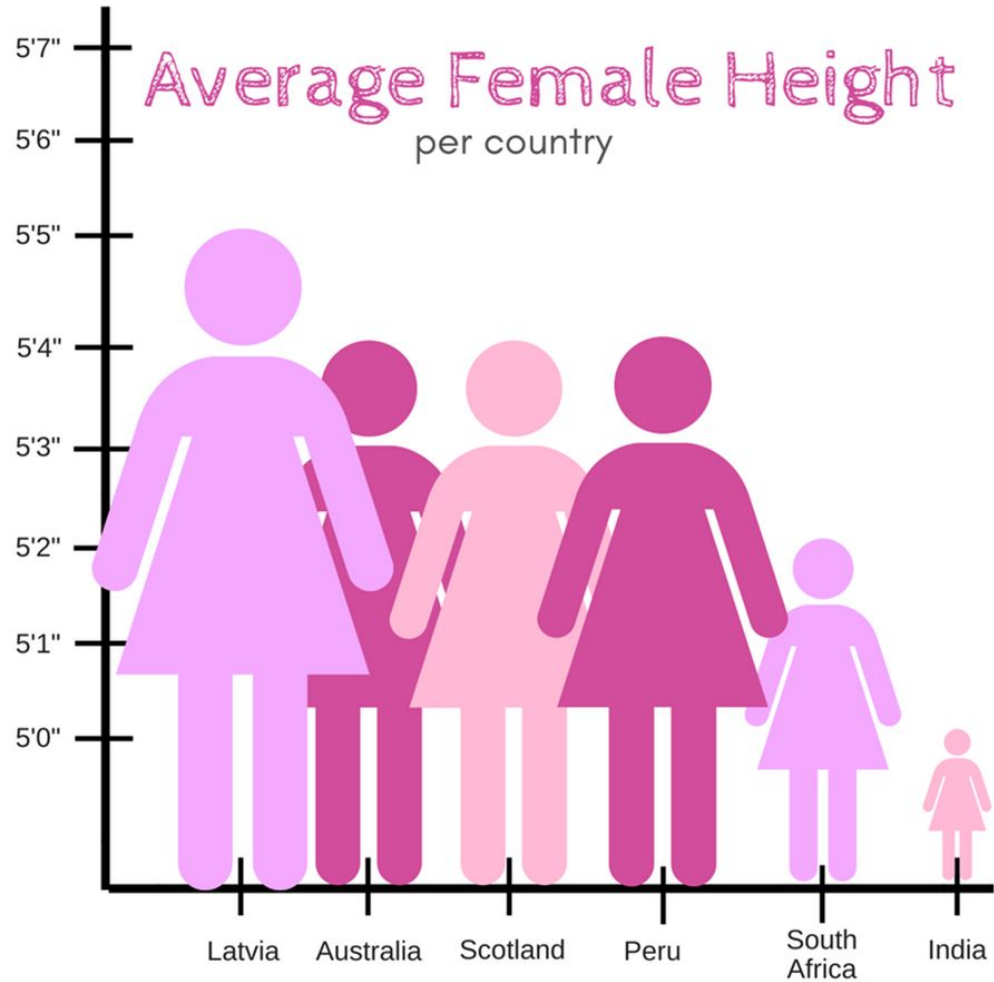
Number of murders committed using firearms






















Source: Florida Department of Law Enforcement

World oil production (TWh/year)





Top Goal Scorers of FIFA World Cup History

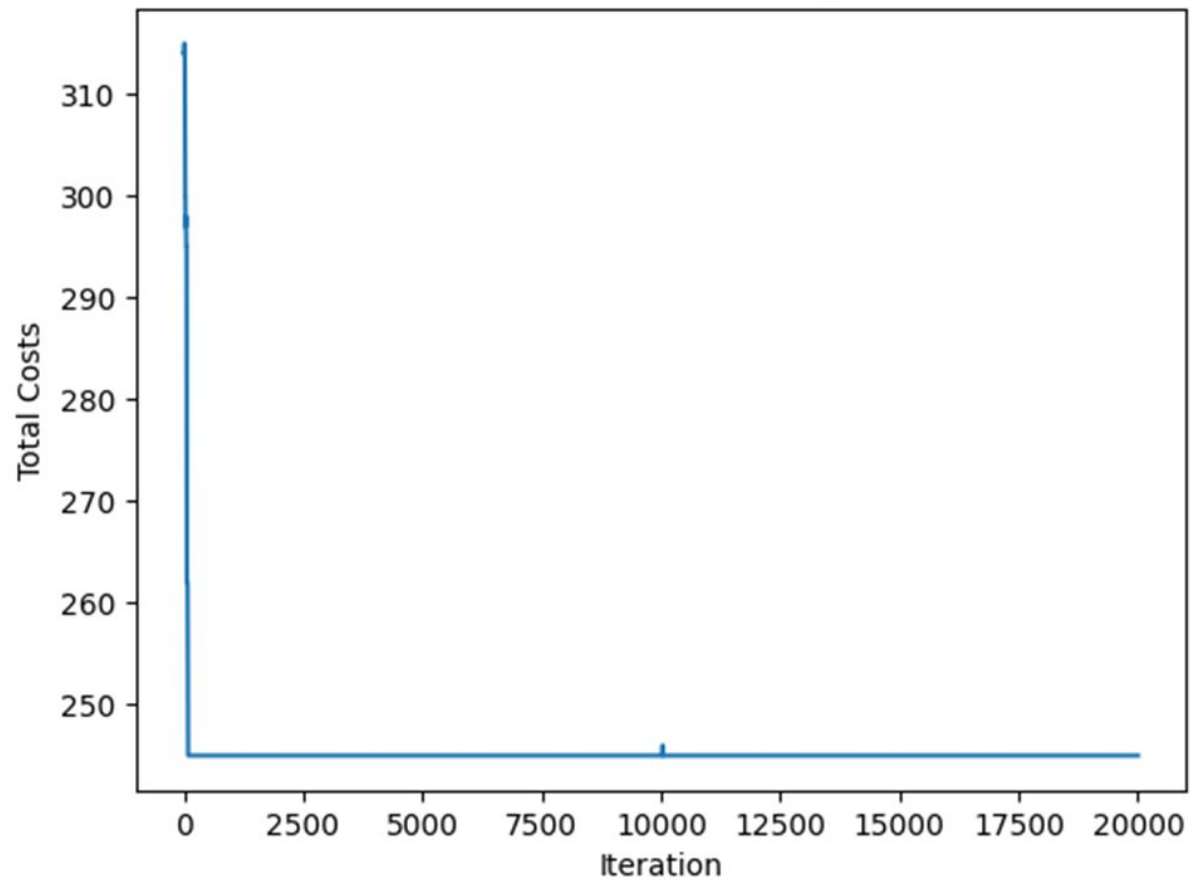
	Goals scored	Matches played	Tournaments
 Miroslav Klose  Germany	16	24	4
 Ronaldo  Brazil	15	19	3
 Gerd Müller  Germany	14	13	2
 Just Fontaine  France	13	1	
 Pelé  Brazil	12	4	
 Sándor Kocsis  Hungary	11	1	
 Jürgen Klinsmann  Germany	11	17	3
 Helmut Rahn  Germany	10	2	
 Gary Lineker  England	10	12	2
 Gabriel Batistuta  Argentina	10	12	3

German striker Miroslav Klose holds the record for most FIFA World Cup goals in history, having scored 16 times in 24 appearances across four editions (2002, 2006, 2010 and 2014). He won the tournament in 2014.

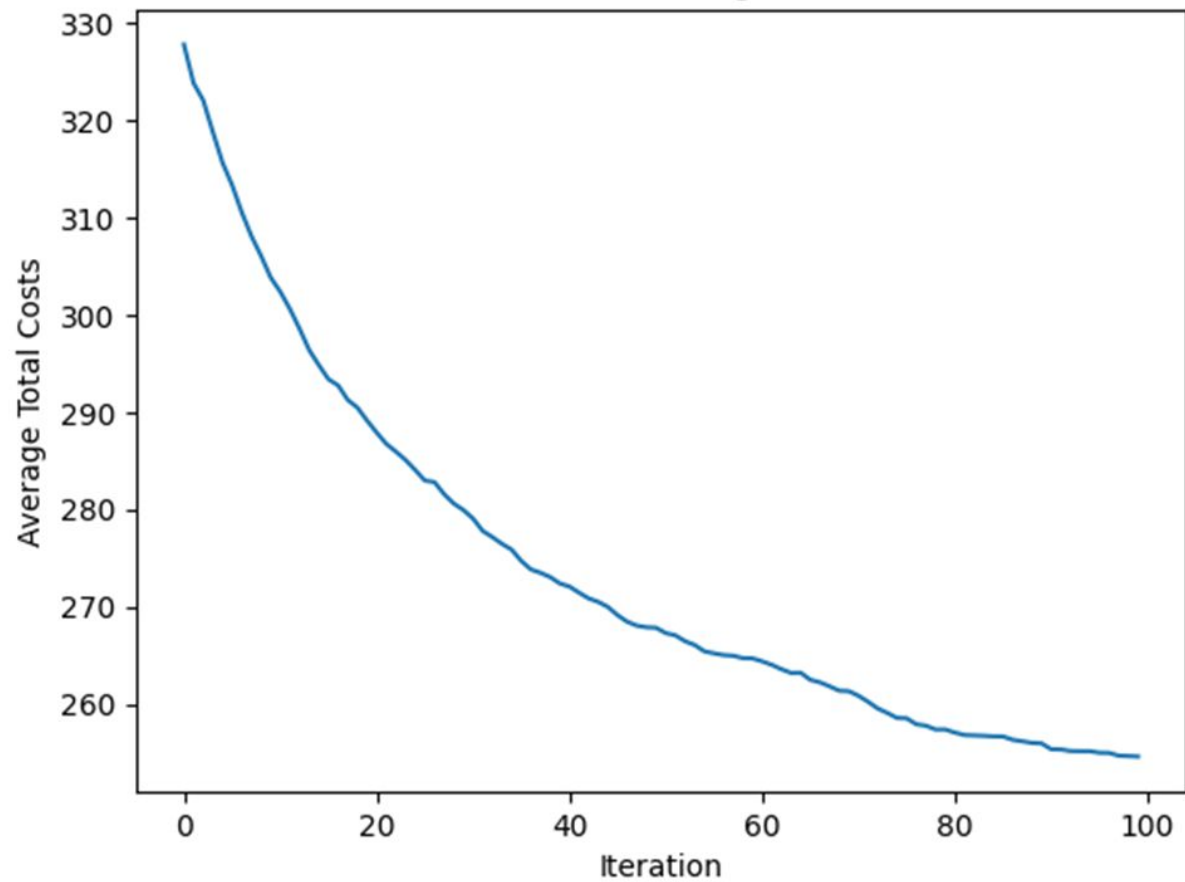
FIFA match reports do not include exact information about players' appearances before 1970

Source : FIFA

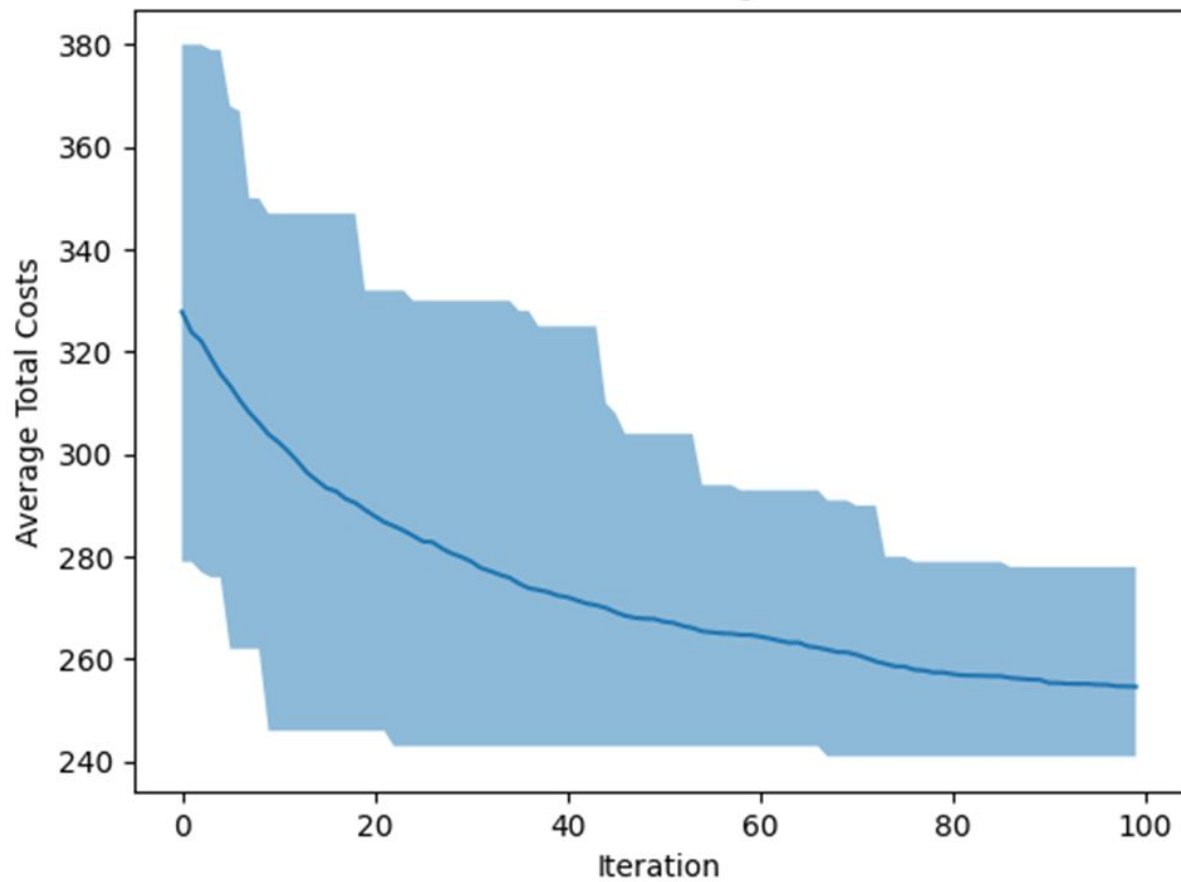
Simulated Annealing



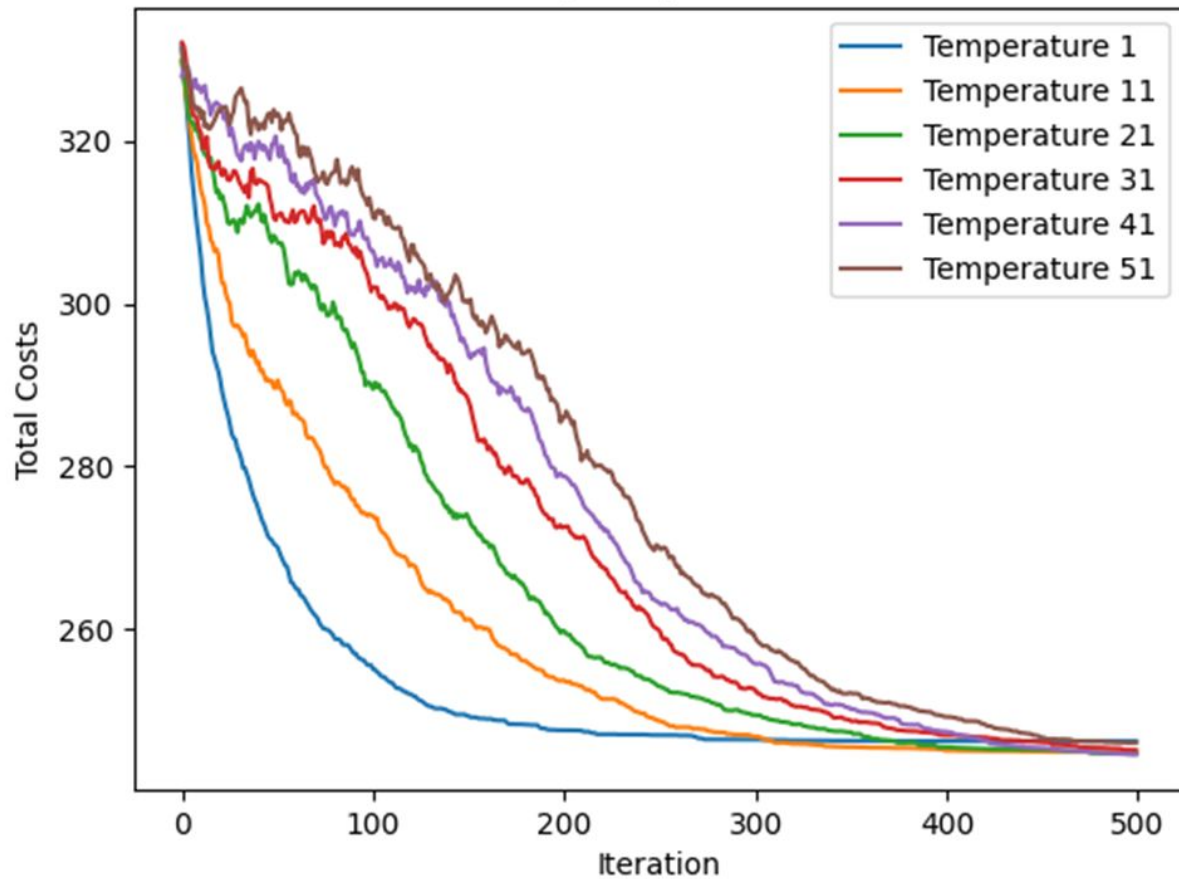
Simulated Annealings (n=100)



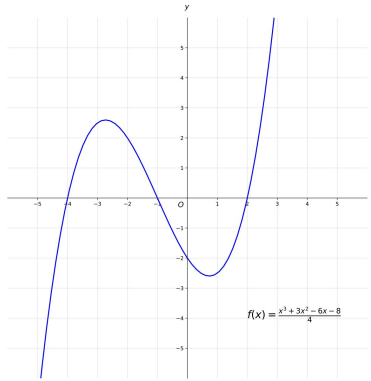
SimulatedAnnealing (n=100)



Sim Annealing Temperatures (n=100)



Problem	Algorithm	Time Complexity	Space Complexity	Best Case	Worst Case	Advantages/Disadvantages
Sorting	QuickSort	$O(n \log n)$	$O(\log n)$	$O(n \log n)$	$O(n^2)$	QuickSort is efficient in practice and has good average case performance. It's also an in-place sorting algorithm.
	MergeSort	$O(n \log n)$	$O(n)$	$O(n \log n)$	$O(n \log n)$	MergeSort is a stable and efficient sorting algorithm. It's not an in-place sorting algorithm.
	BubbleSort	$O(n^2)$	$O(1)$	$O(n)$	$O(n^2)$	BubbleSort is easy to understand and implement, but it's not efficient for large datasets.
Searching	Binary Search	$O(\log n)$	$O(1)$	$O(1)$	$O(\log n)$	Binary search is efficient and has a good average case performance. It requires that the data is sorted.
	Linear Search	$O(n)$	$O(1)$	$O(1)$	$O(n)$	Linear search is simple and easy to understand but not efficient for large datasets.
	Ternary Search	$O(\log n)$	$O(1)$	$O(1)$	$O(\log n)$	Ternary search is similar to binary search but with a slightly worse average case performance. It also requires that the data is sorted.



Problem	Algorithm 1	Algorithm 2	Algorithm 3
Sorting	QuickSort	MergeSort	BubbleSort
Searching	Binary Search	Linear Search	Ternary Search
Graph traversal	DFS	BFS	Dijkstra's
String matching	KMP	Boyer Moore	Rabin-Karp
Shortest Path	Bellman Ford	Dijkstra's	A*

Resultaten



Conclusie

De oplossing voor de case

Inzichten in algoritmen & case

Future work

Pro Tips

```
public static void  
main(String[]  
args) {  
    System.out.println  
    ("Geen code");  
}
```

Afbeelding > Tekst

Goed dat je het vraagt!

Hebben we een slide voor...



The end

<https://thomasweise.github.io/oa/>