COMPX216-24A
Artificial Intelligence

Local search and optimisation

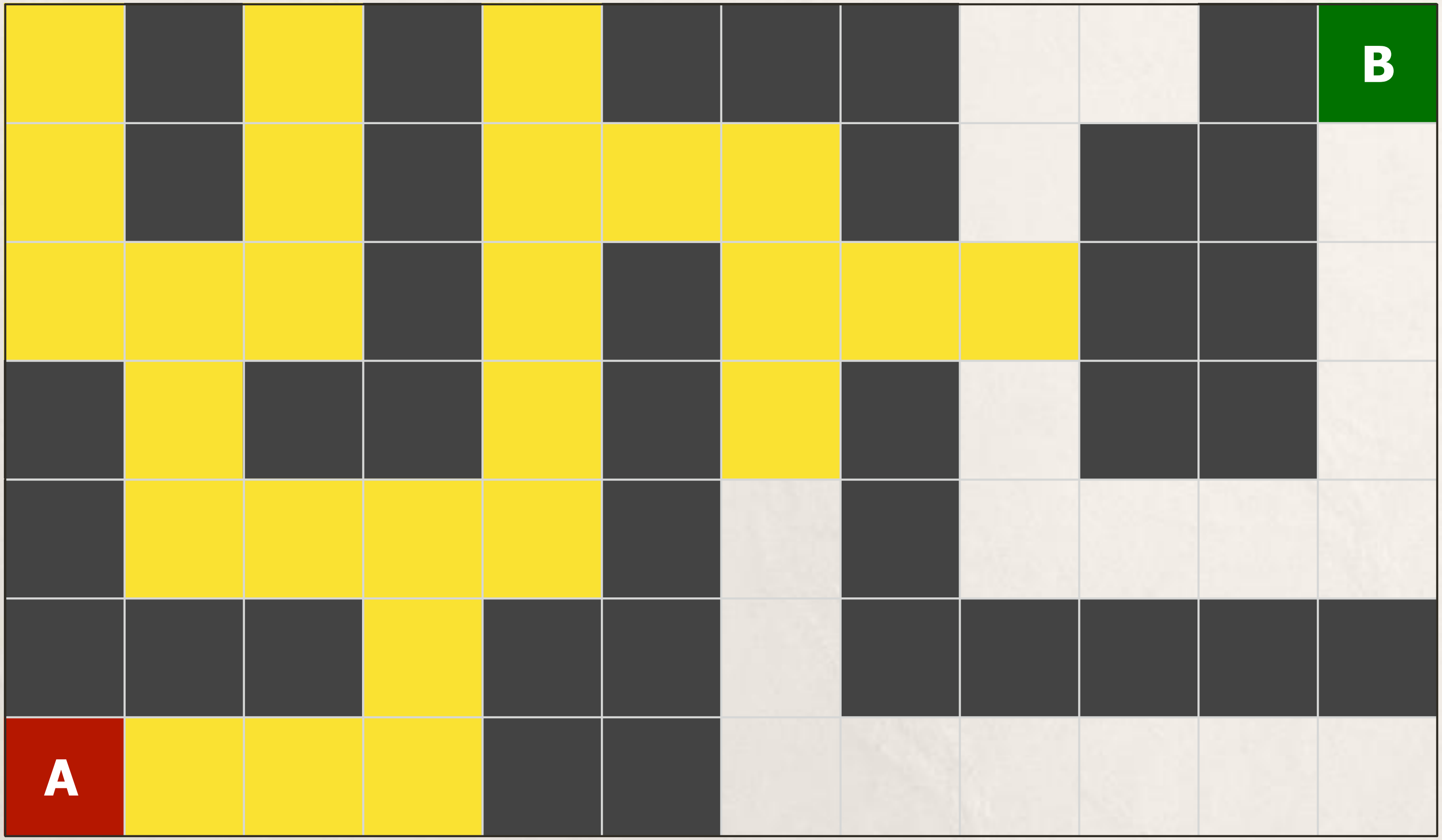
Today: Local search and optimisation

- Optimisation problems
- Local search
- Hill-climbing search
- Local and global optima
- Ridges and plateaus
- The 8-queens problem
- Variants of hill-climbing
- Simulated annealing
- Local beam search

Optimisation problems

- In many applications, we just want to find a good state, not a sequence of actions that get us to a goal from an initial state
- Example: given items of different weight, how can we distribute them as fairly as possible into two backpacks?
  - This is an instance of the *number partitioning problem*
- Finding a good state is called an **optimisation problem**
- In the context of optimization problems, the evaluation function used for states is called the **objective function**
- Depending on the problem, we may want to find a state where the objective function is **maximised** or **minimised**

Recall: Search problems

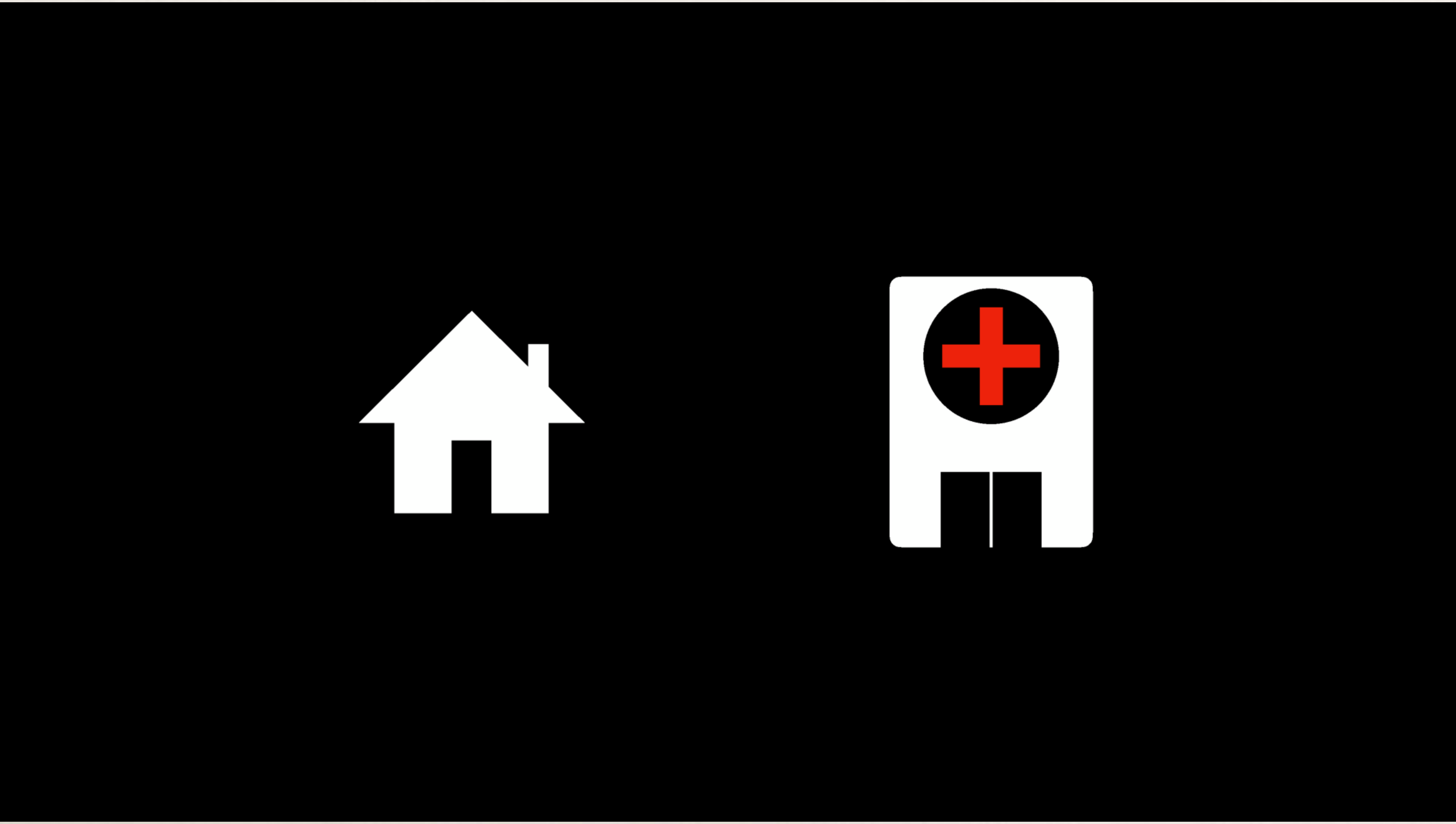


Local search problems

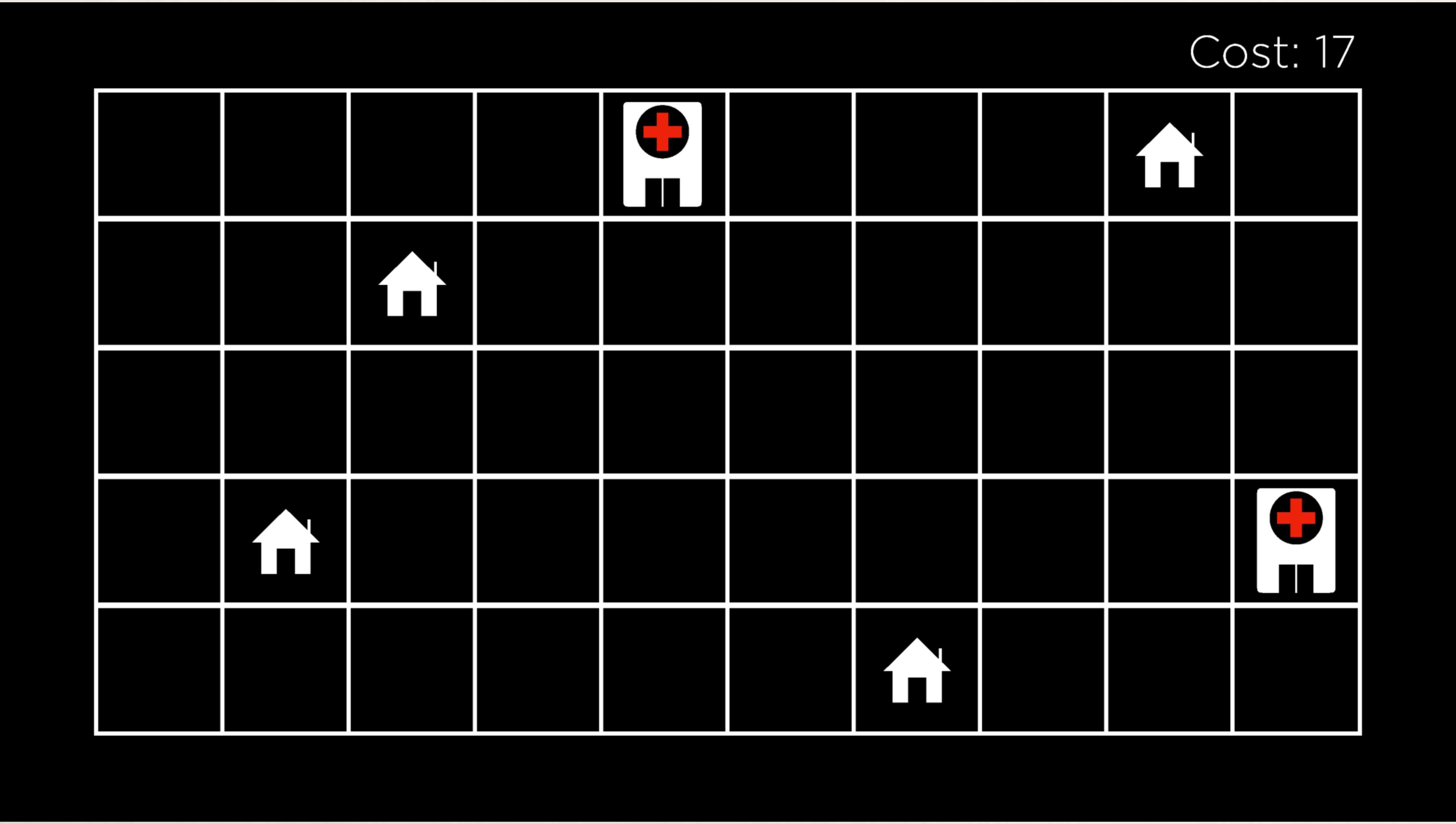
Local search

- Optimisation problems are often tackled using **local search** algorithms that may not find the best possible state
- Local search algorithms do not keep track of paths used to reach states and do not keep track of states reached previously
- In the simplest case, they move around the state space by keeping information only about the state that looks best so far
- Local search algorithms require very limited memory but are unsystematic in their search

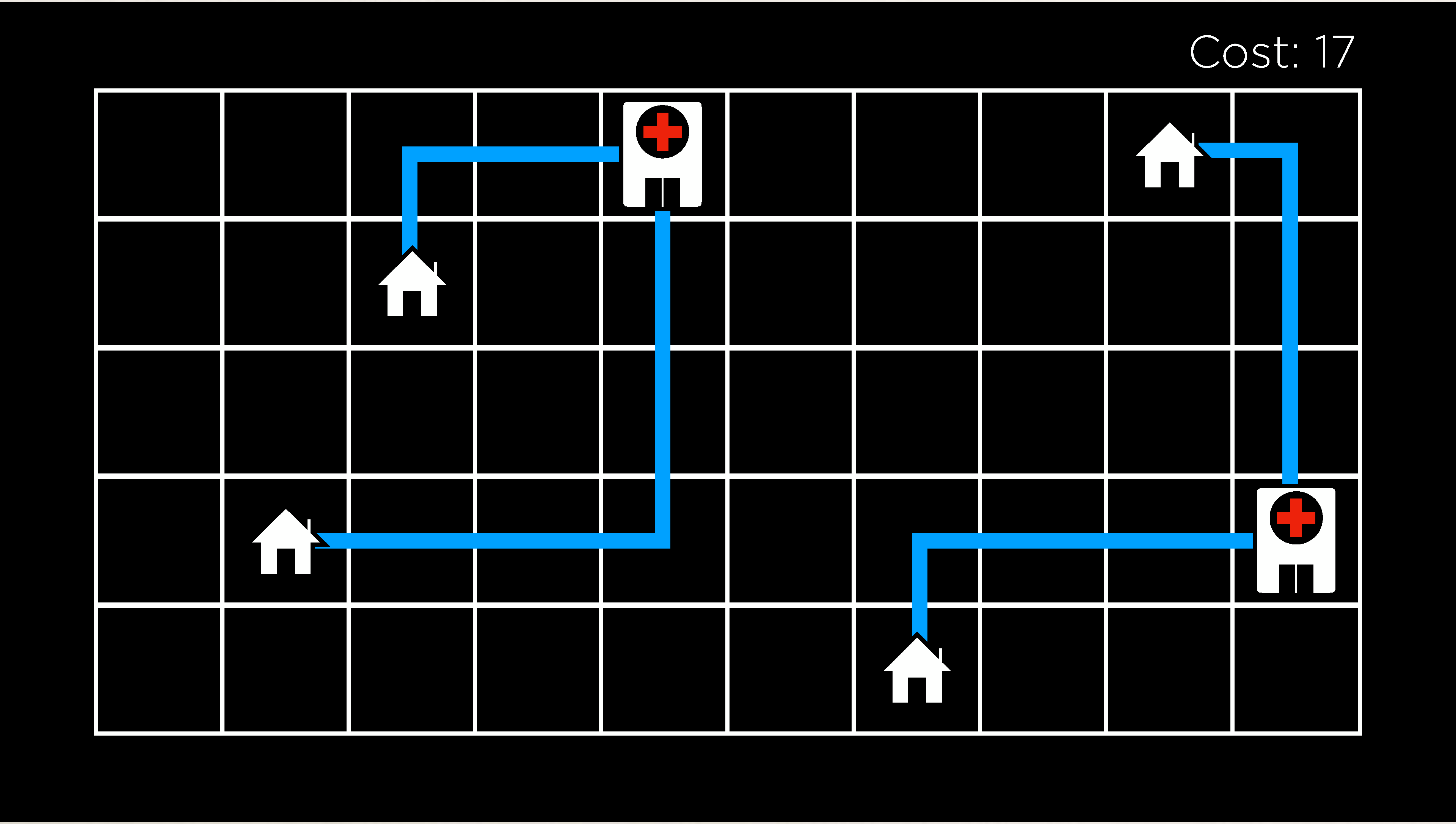
Local search and optimisation



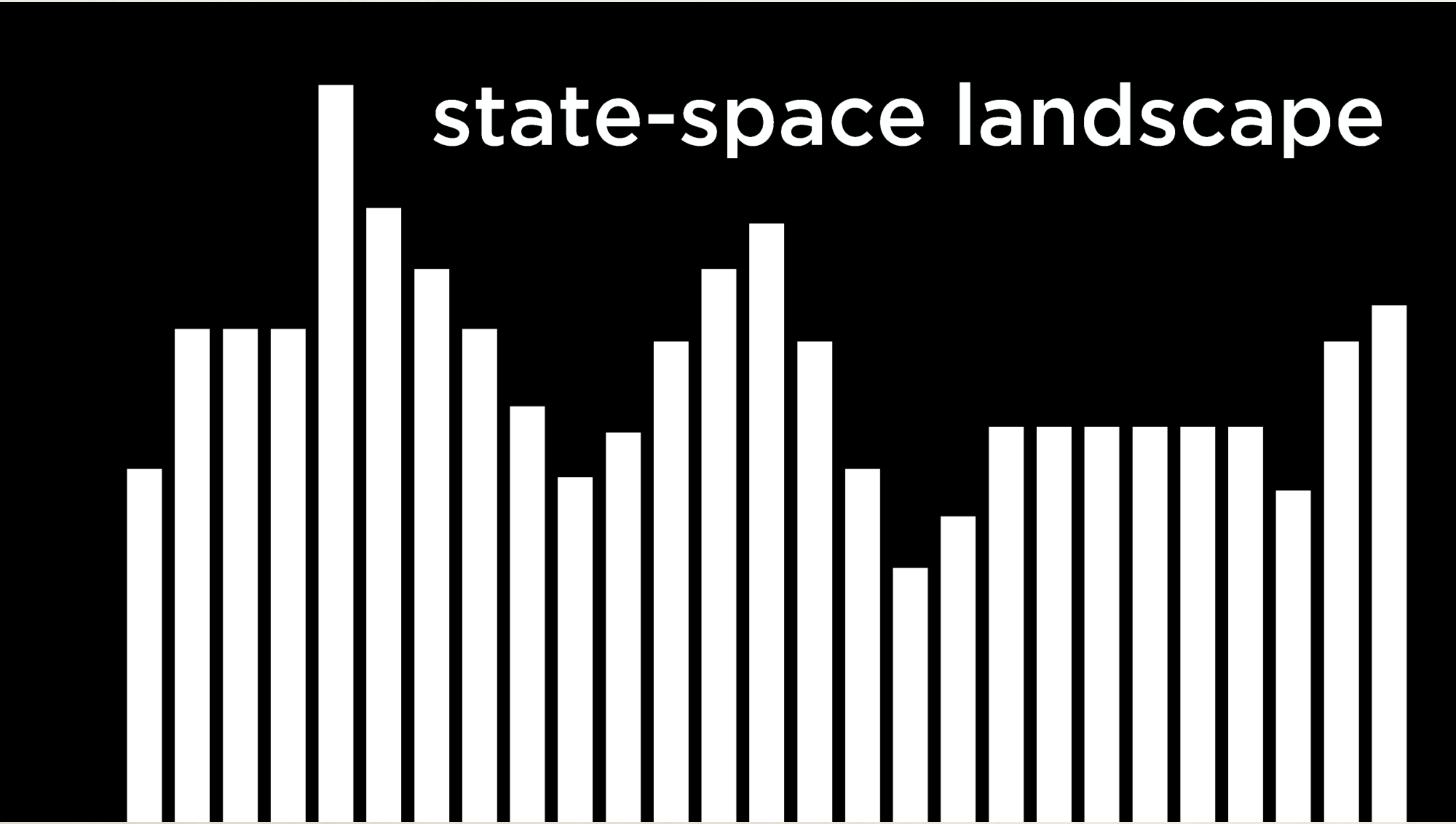
Local search and optimisation



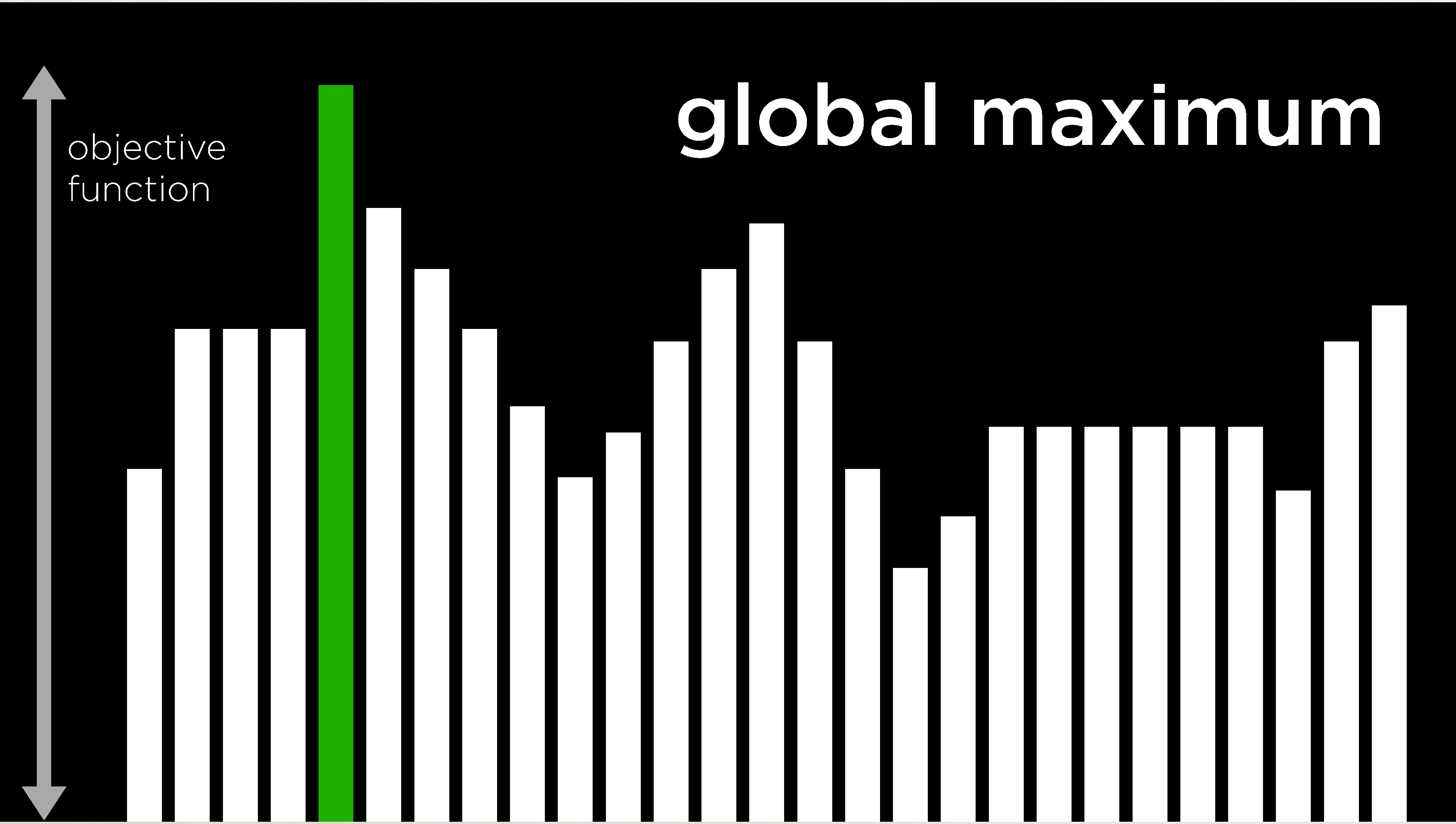
Local search and optimisation



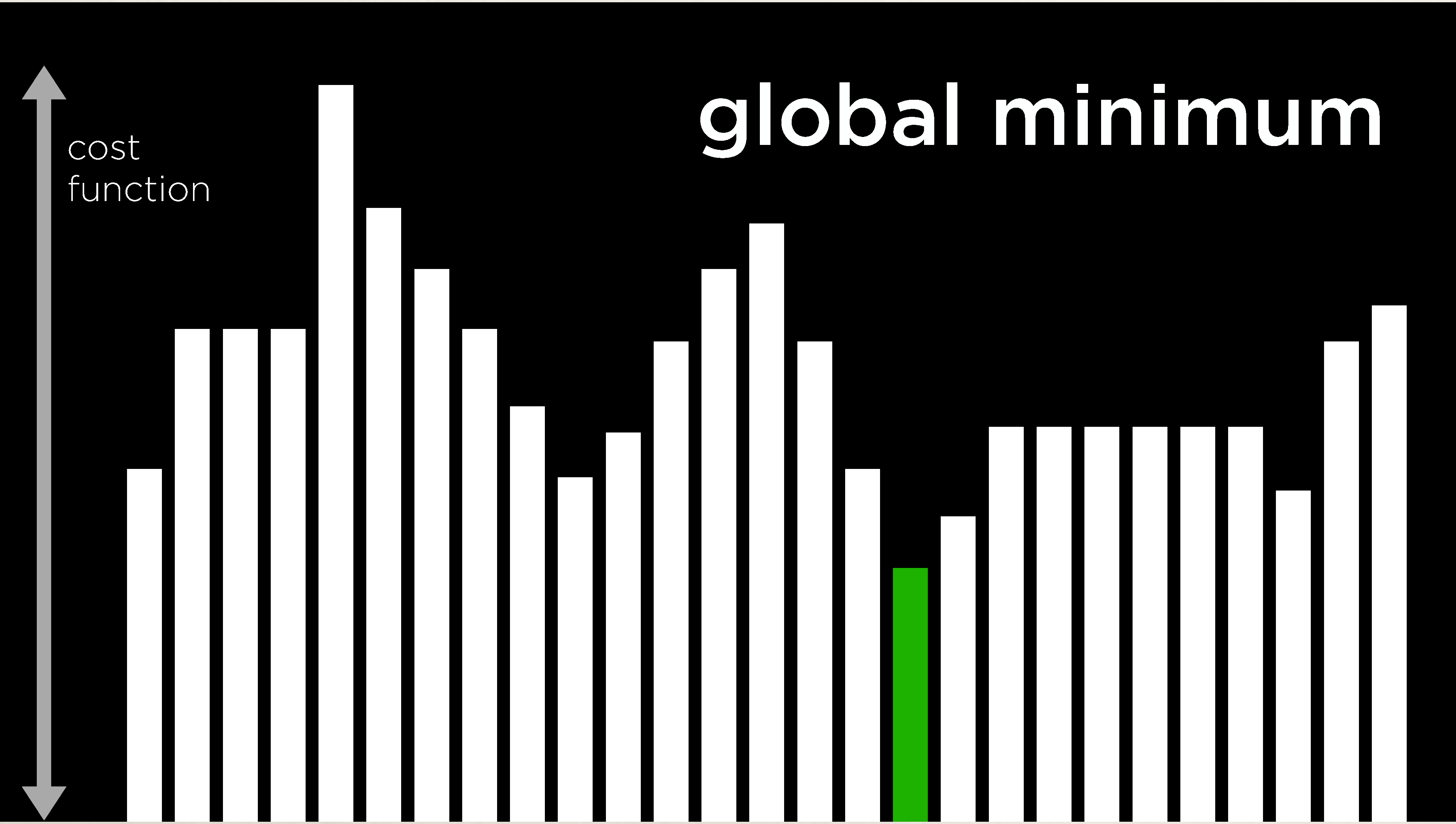
Local search and optimisation



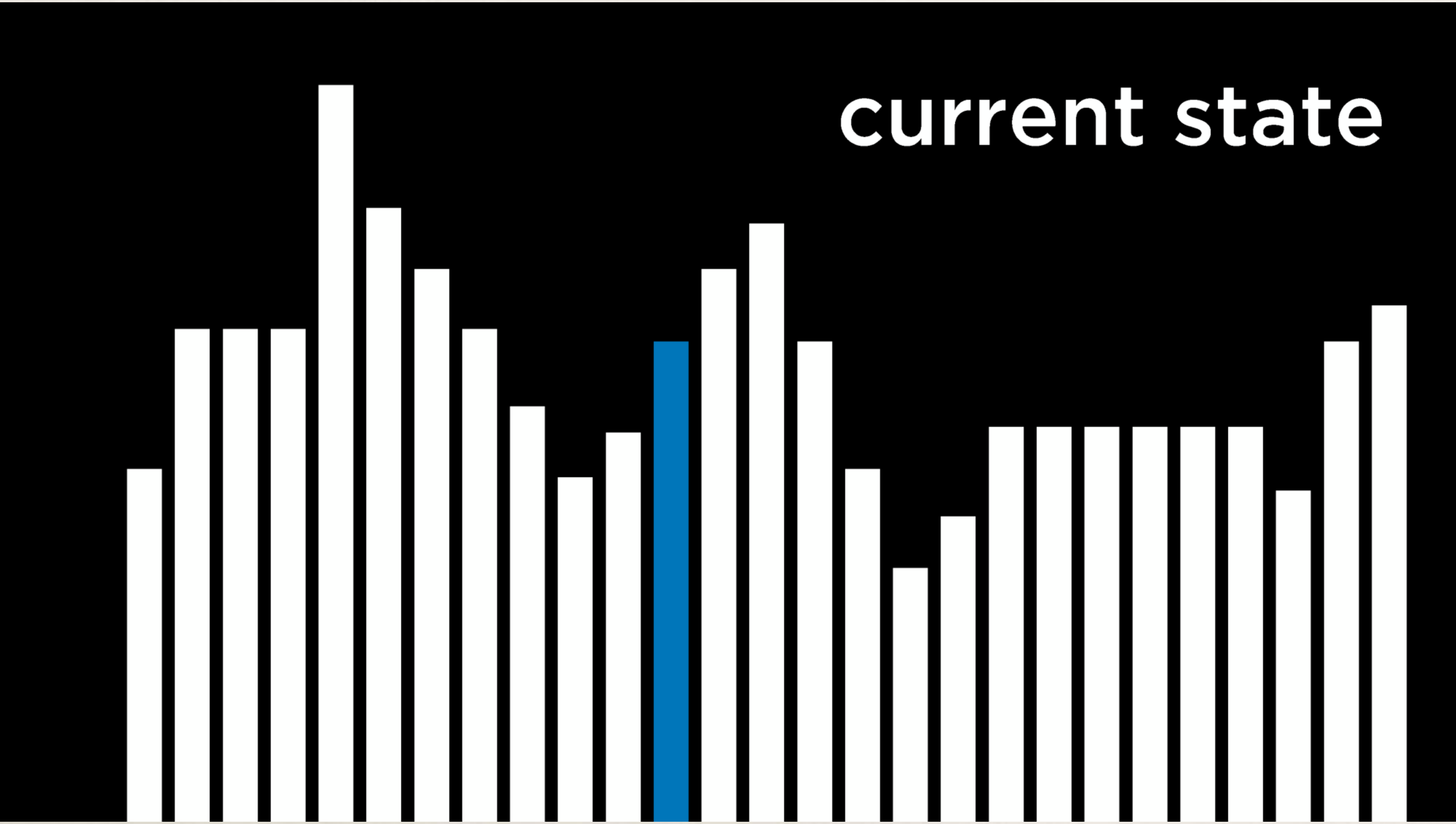
Local search and optimisation



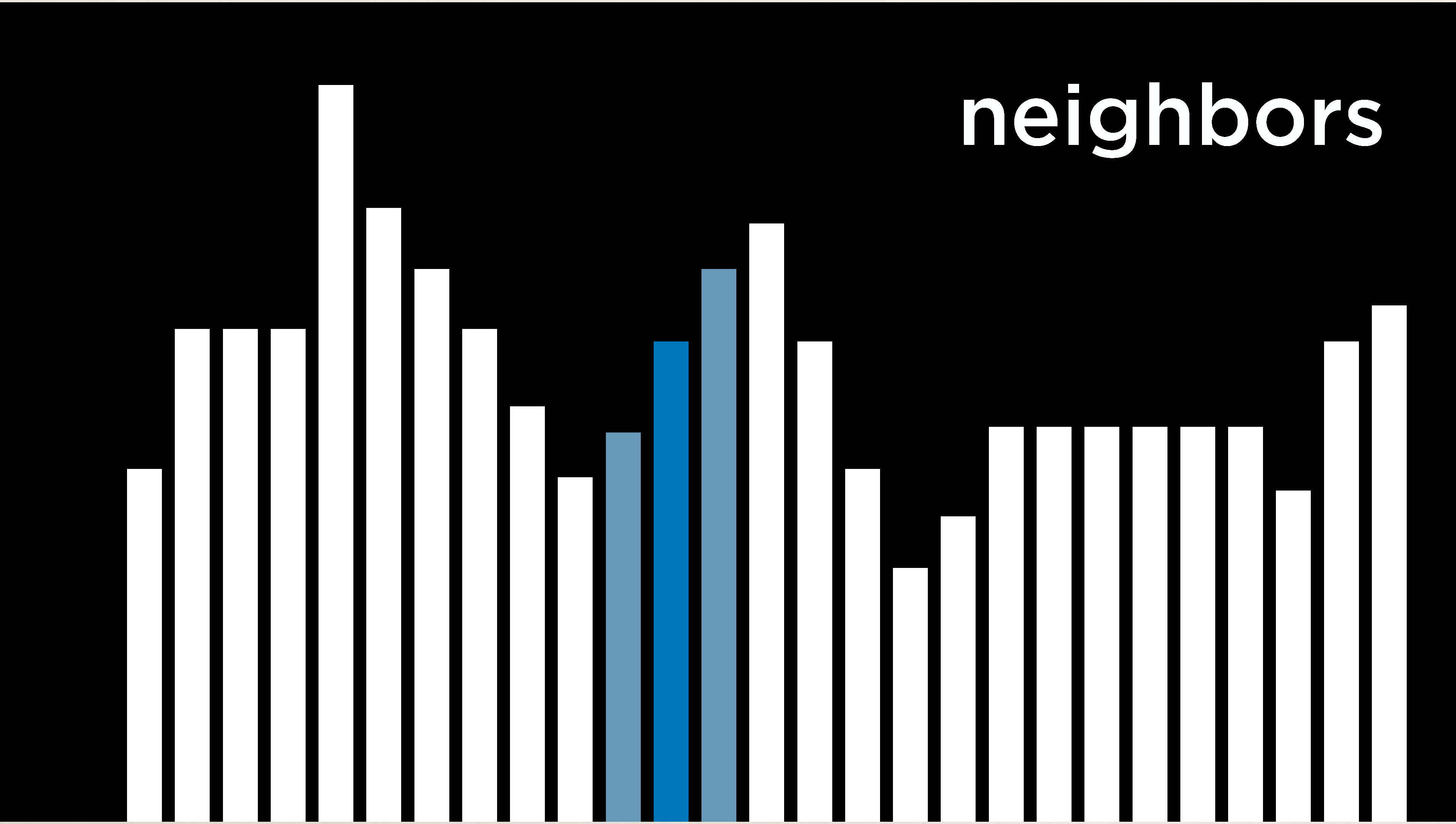
Local search and optimisation



Local search and optimisation



Local search and optimisation

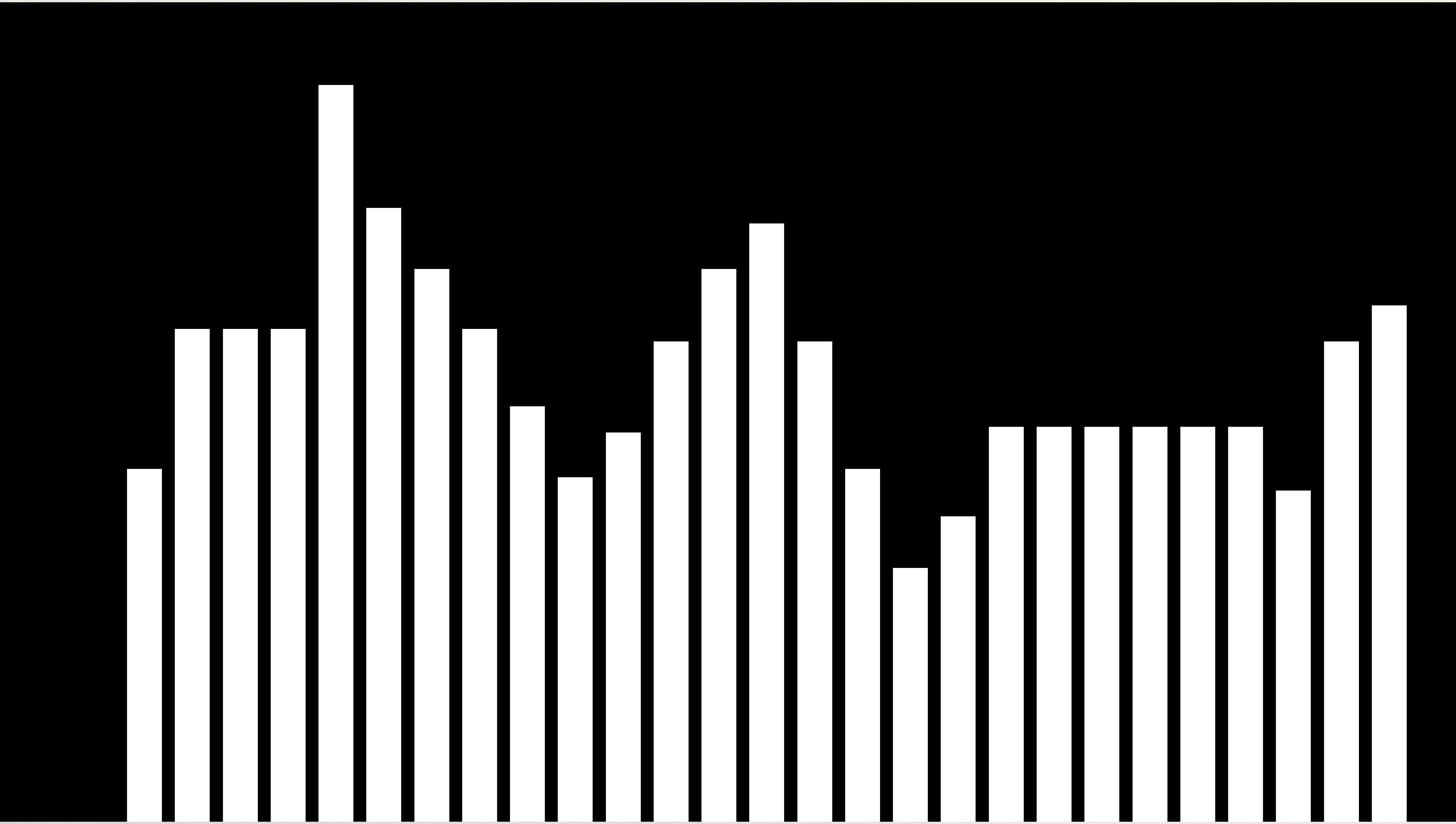


Hill-climbing search

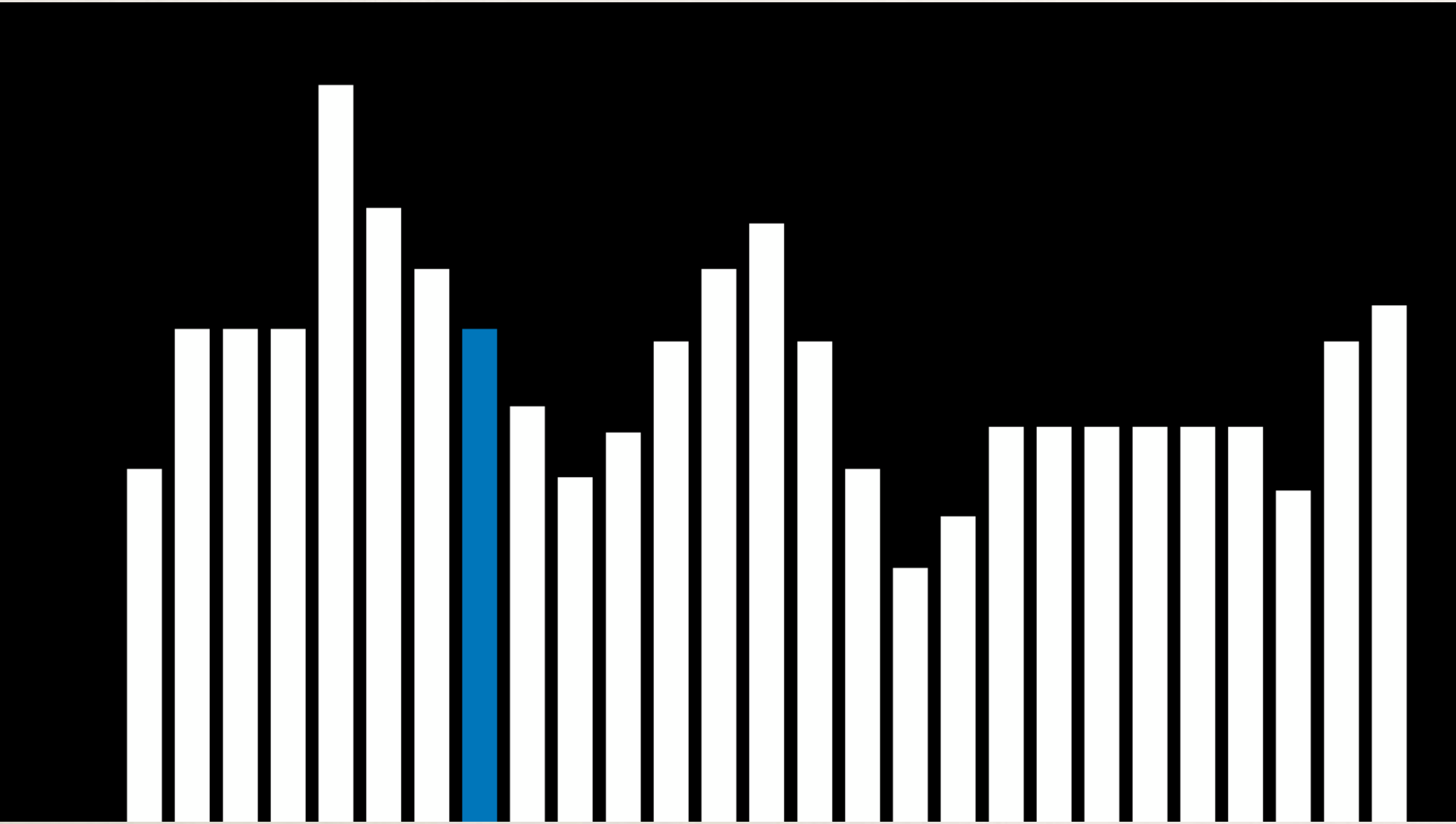
- **Hill-climbing search** is a type of optimsation problem.
- Moves to the best neighbour in state space in each iteration until no neighbour is better
- More sophisticated variants keep information about a fixed-size set of states

https://inteligenciaartificial360.com/glosario/algoritmo-de-hill-climbing/

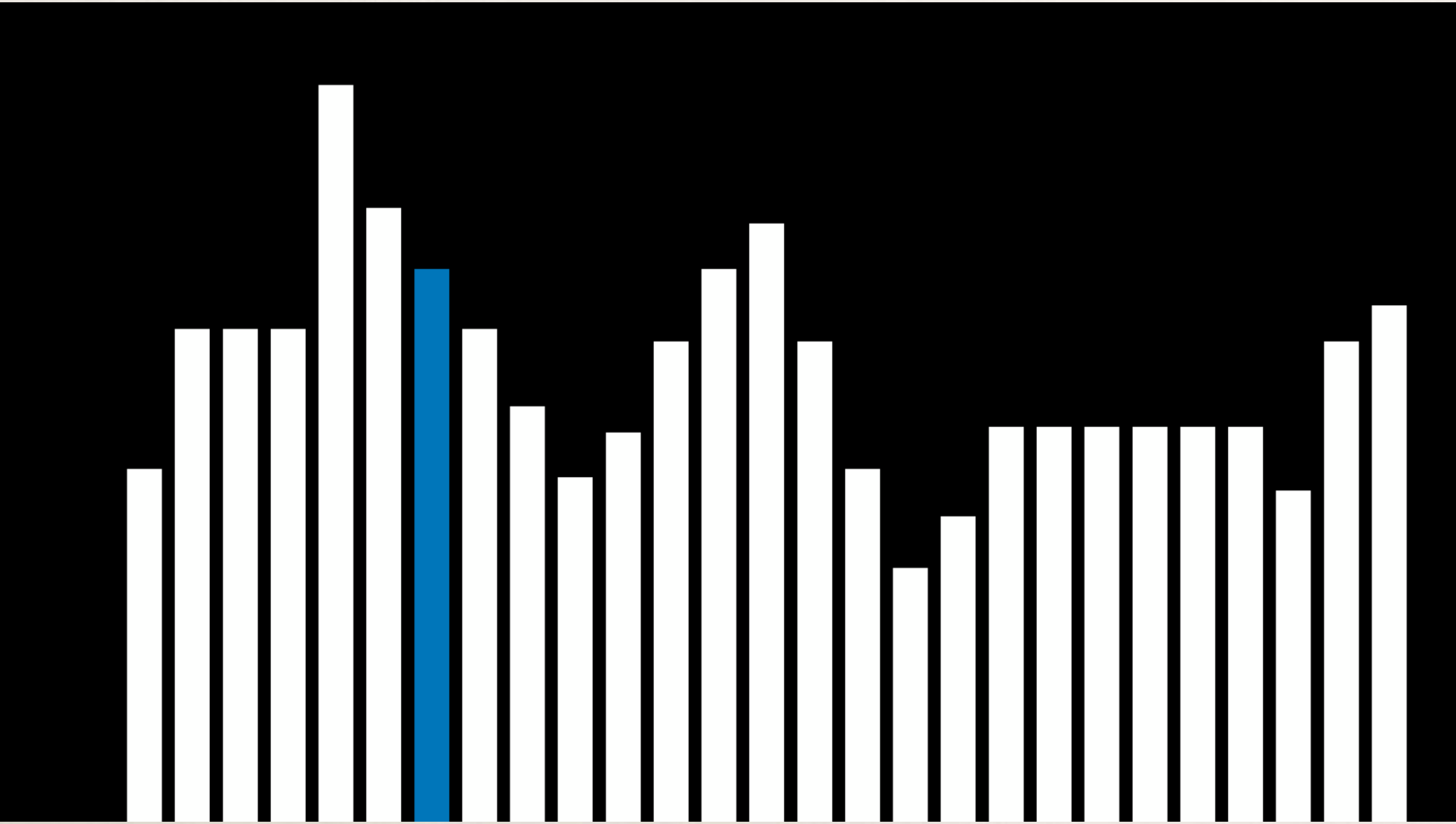
Hill-climbing search



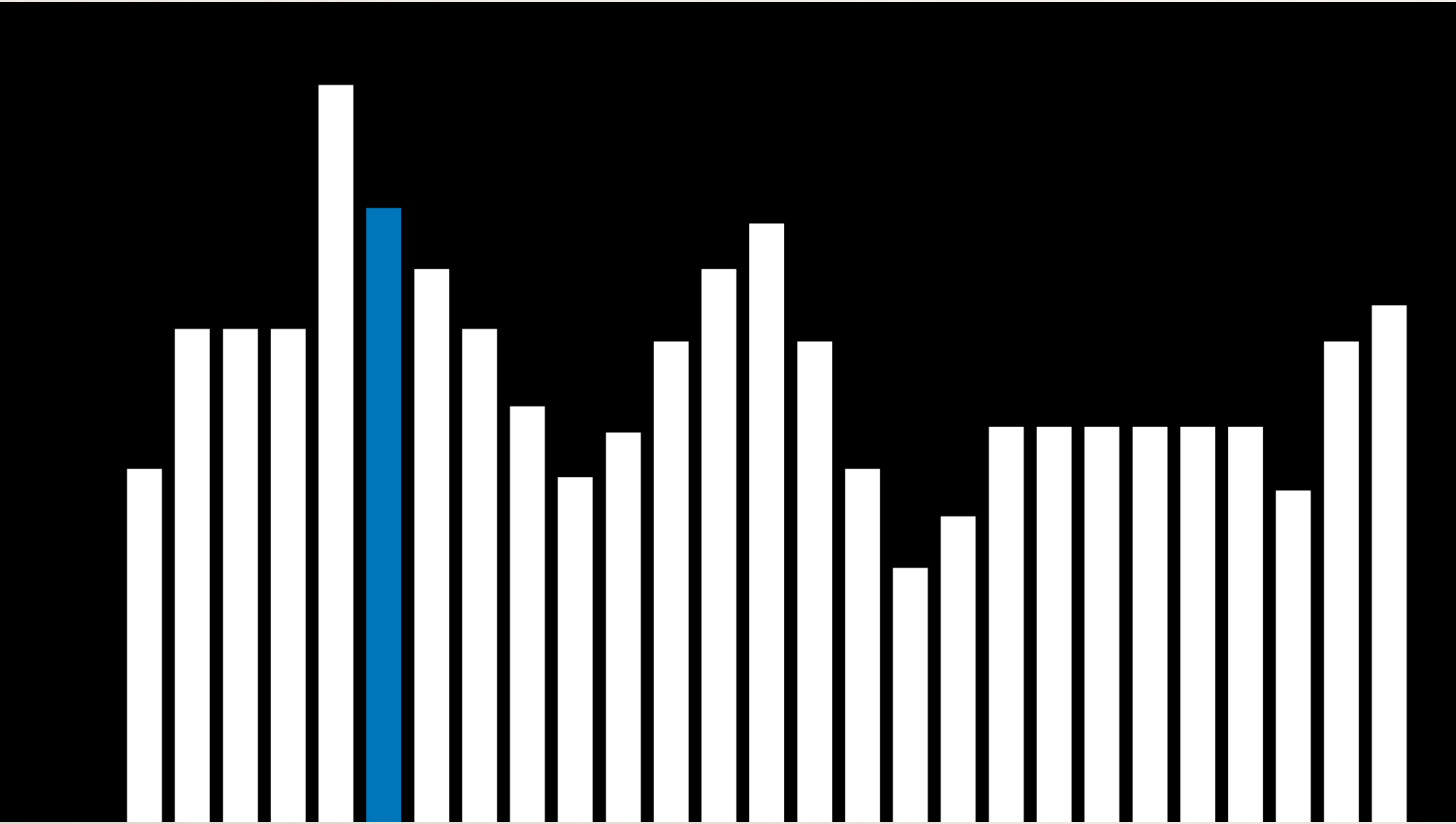
Hill-climbing search



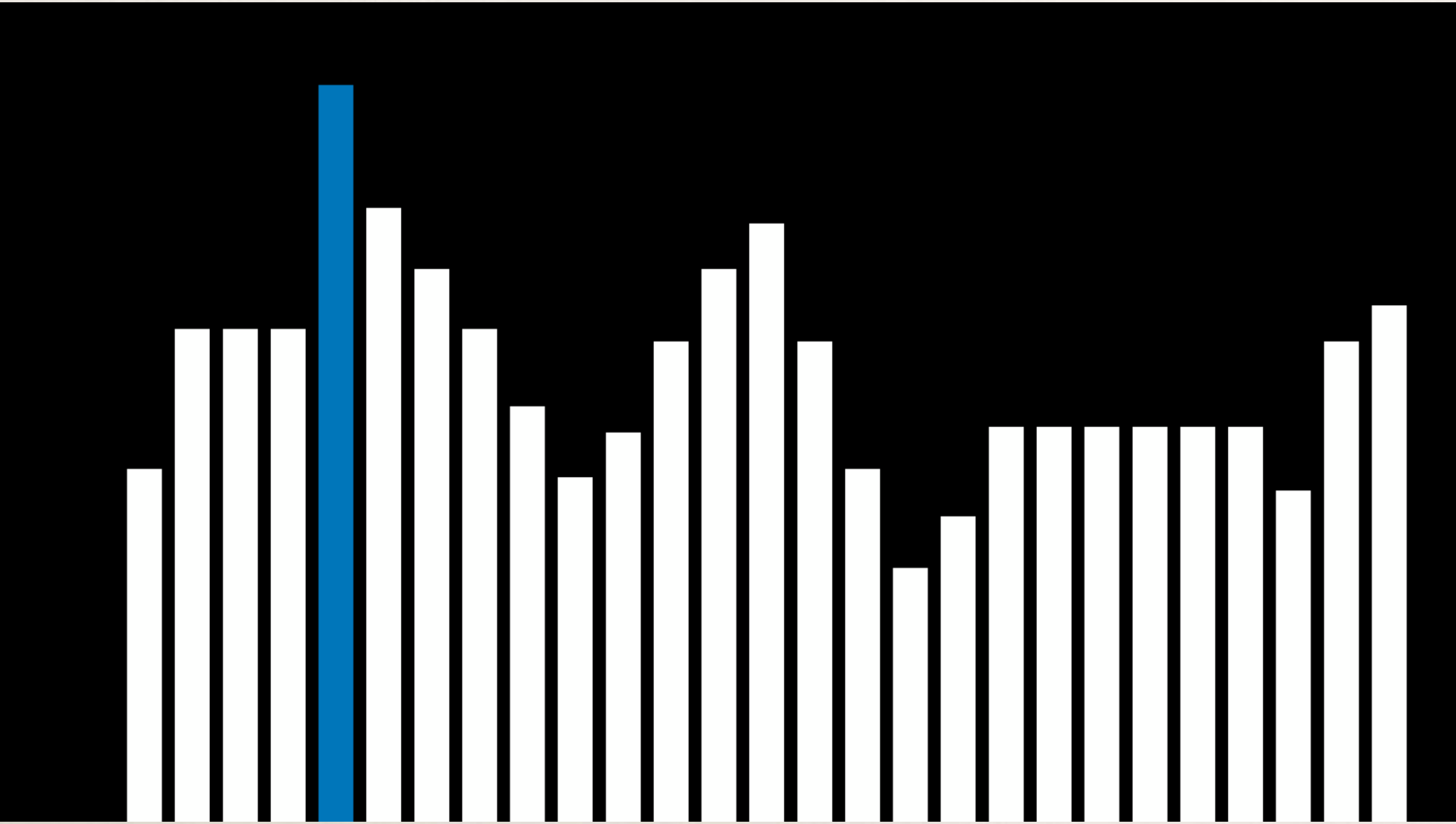
Hill-climbing search



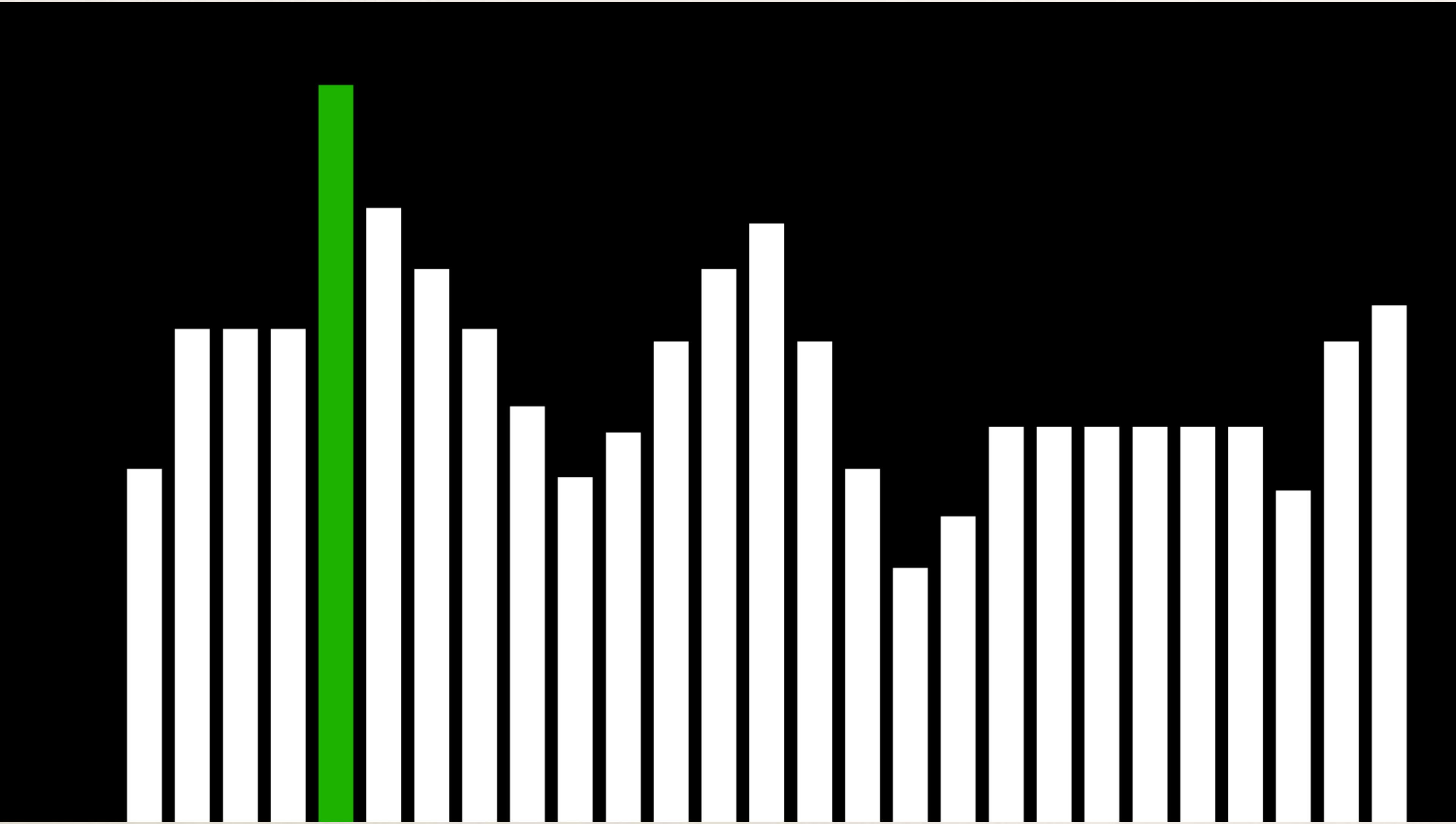
Hill-climbing search



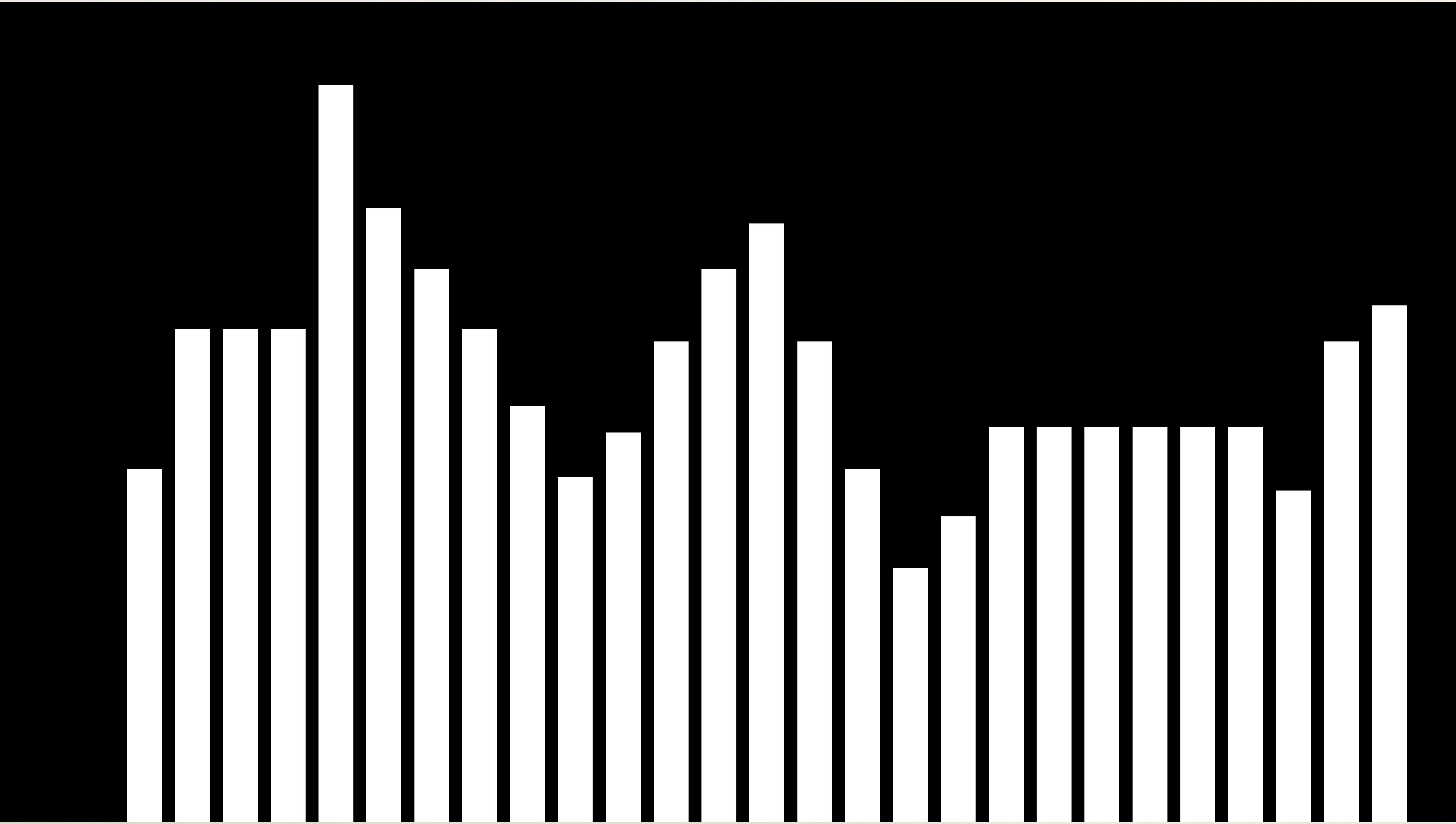
Hill-climbing search



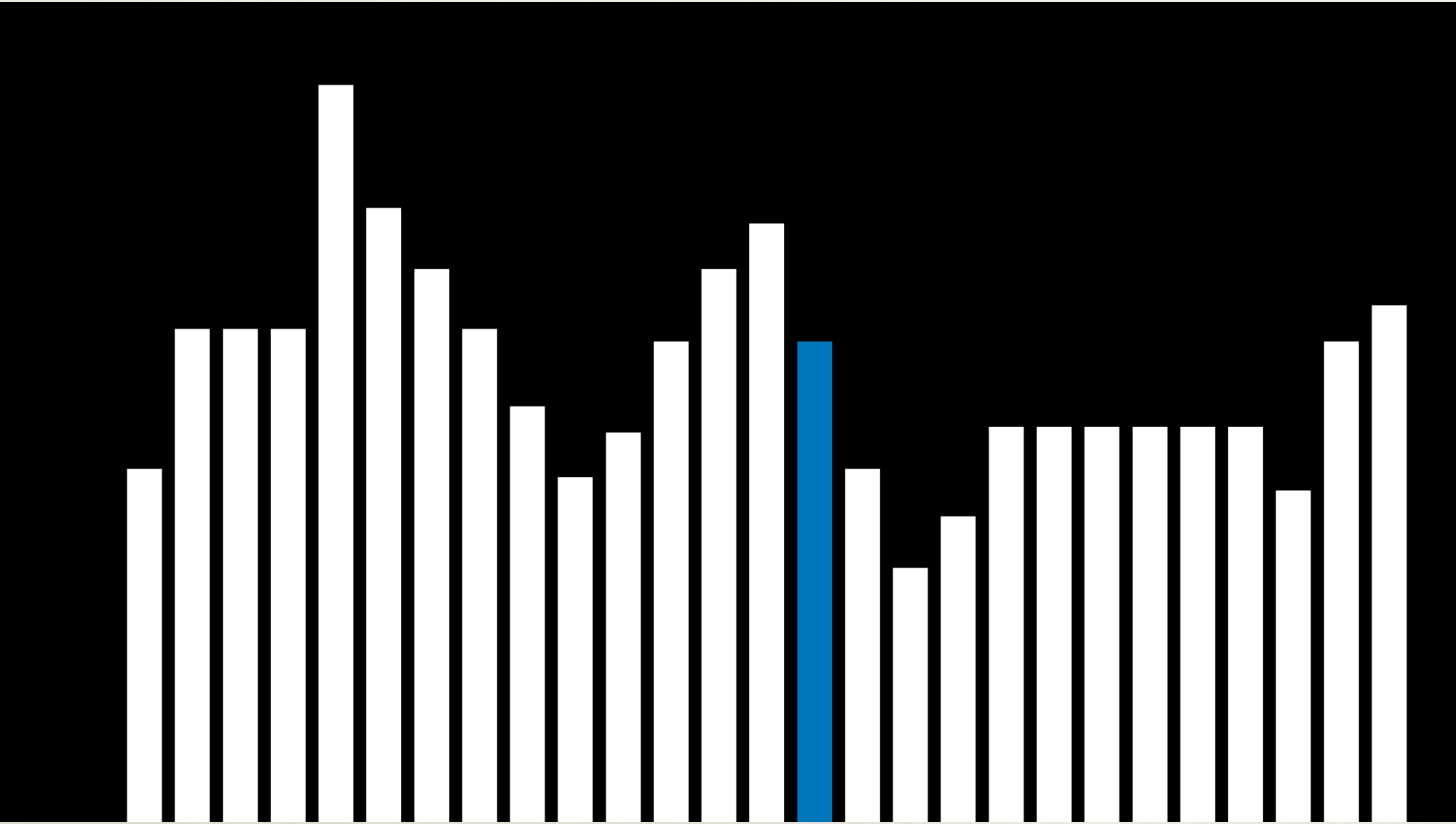
Hill-climbing search



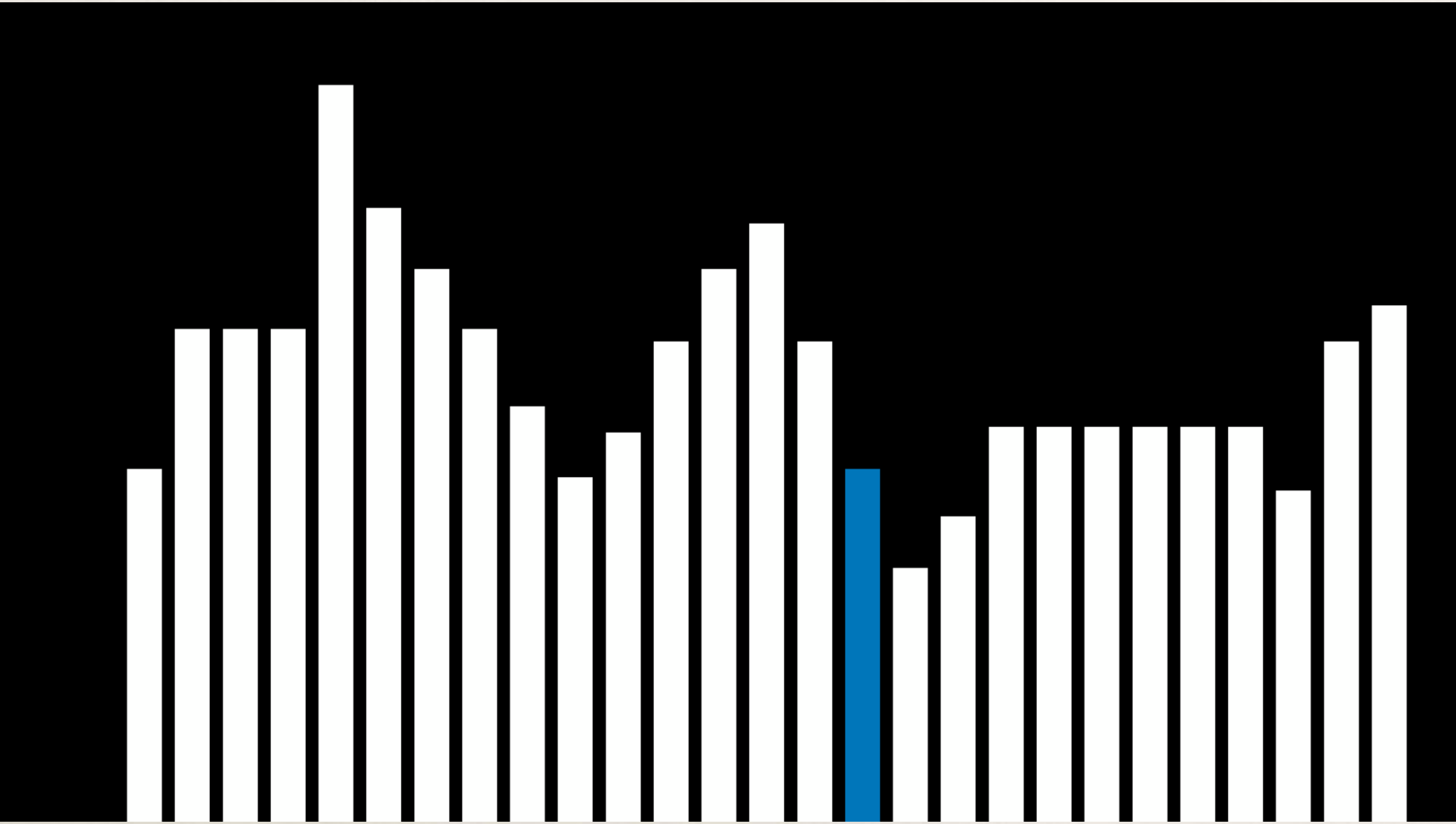
Hill-climbing search



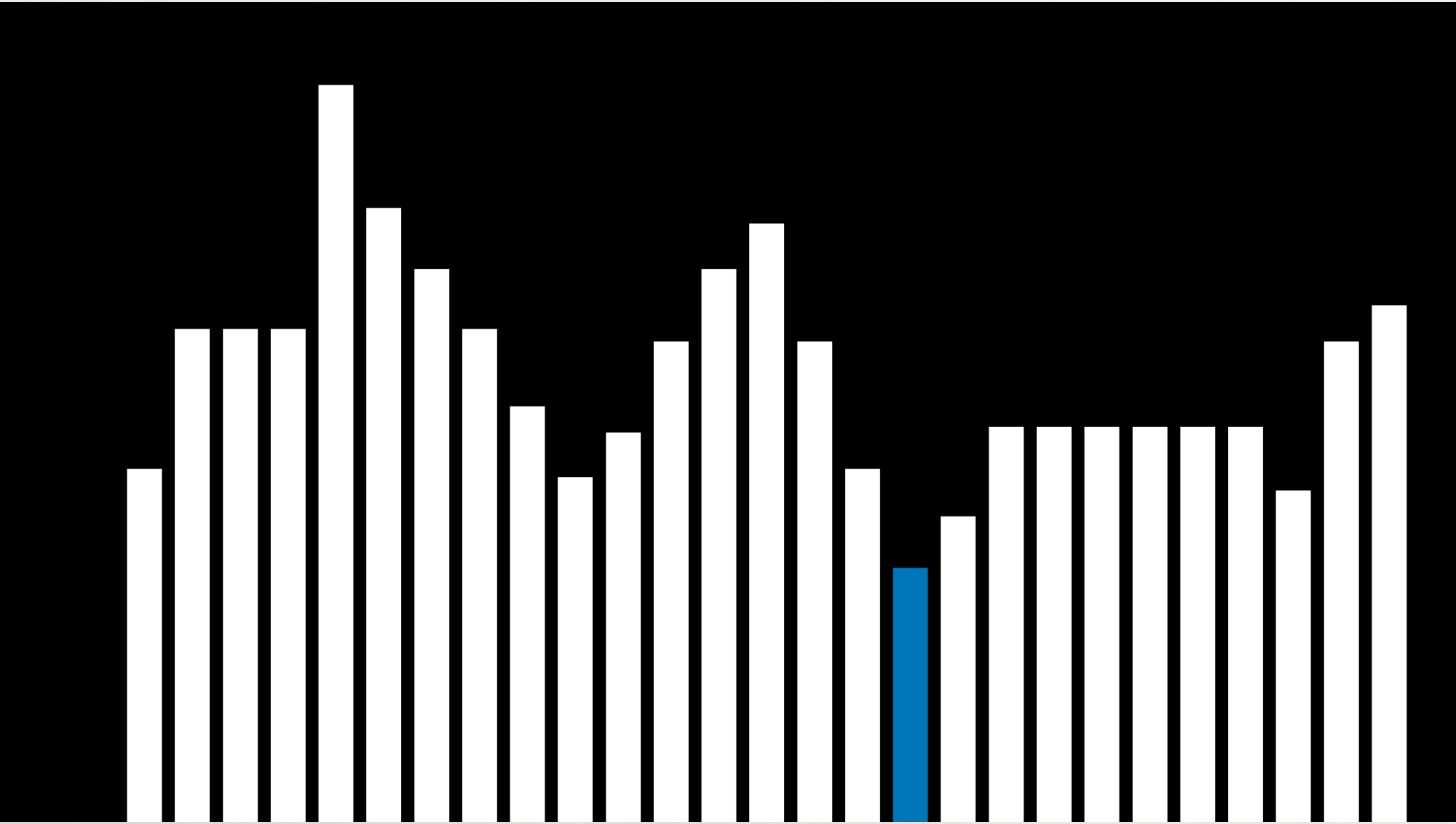
Hill-climbing search



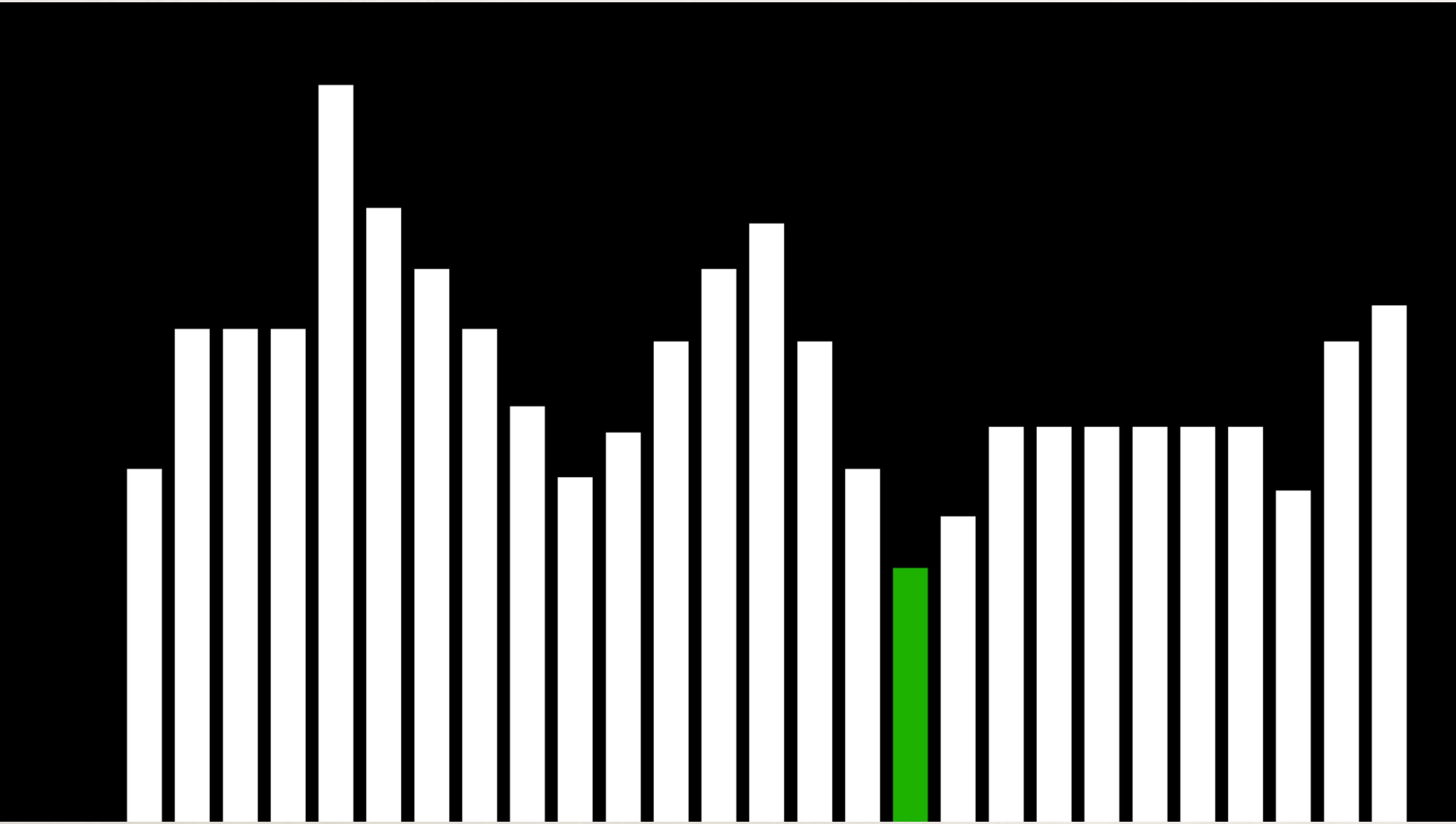
Hill-climbing search



Hill-climbing search



Hill-climbing search

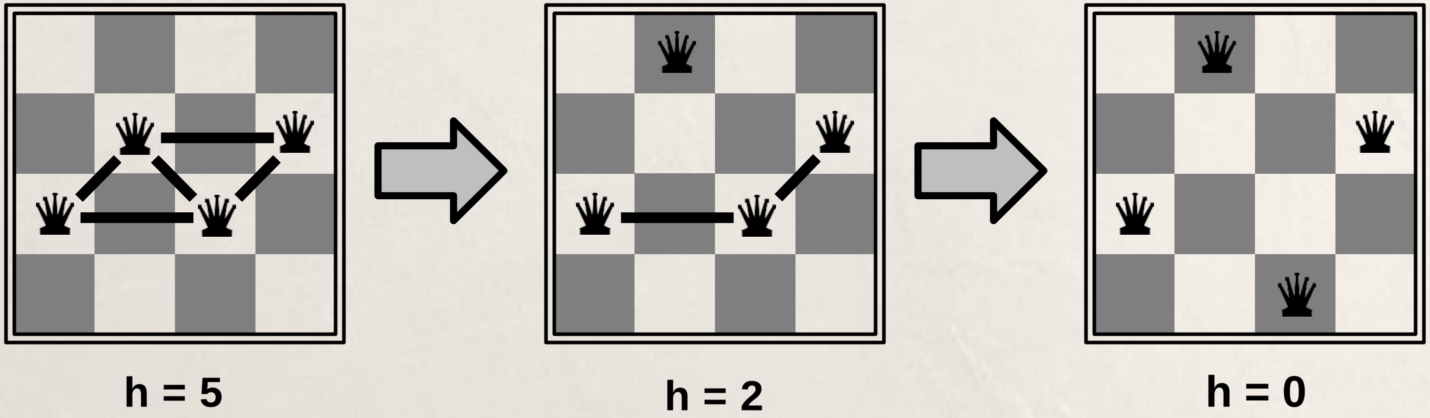


Hill-climbing search

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
    current ← *problem*.INITIAL
    while *true* **do**
        neighbor ← a highest-valued successor state of *current*
        if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
        current ← *neighbor*

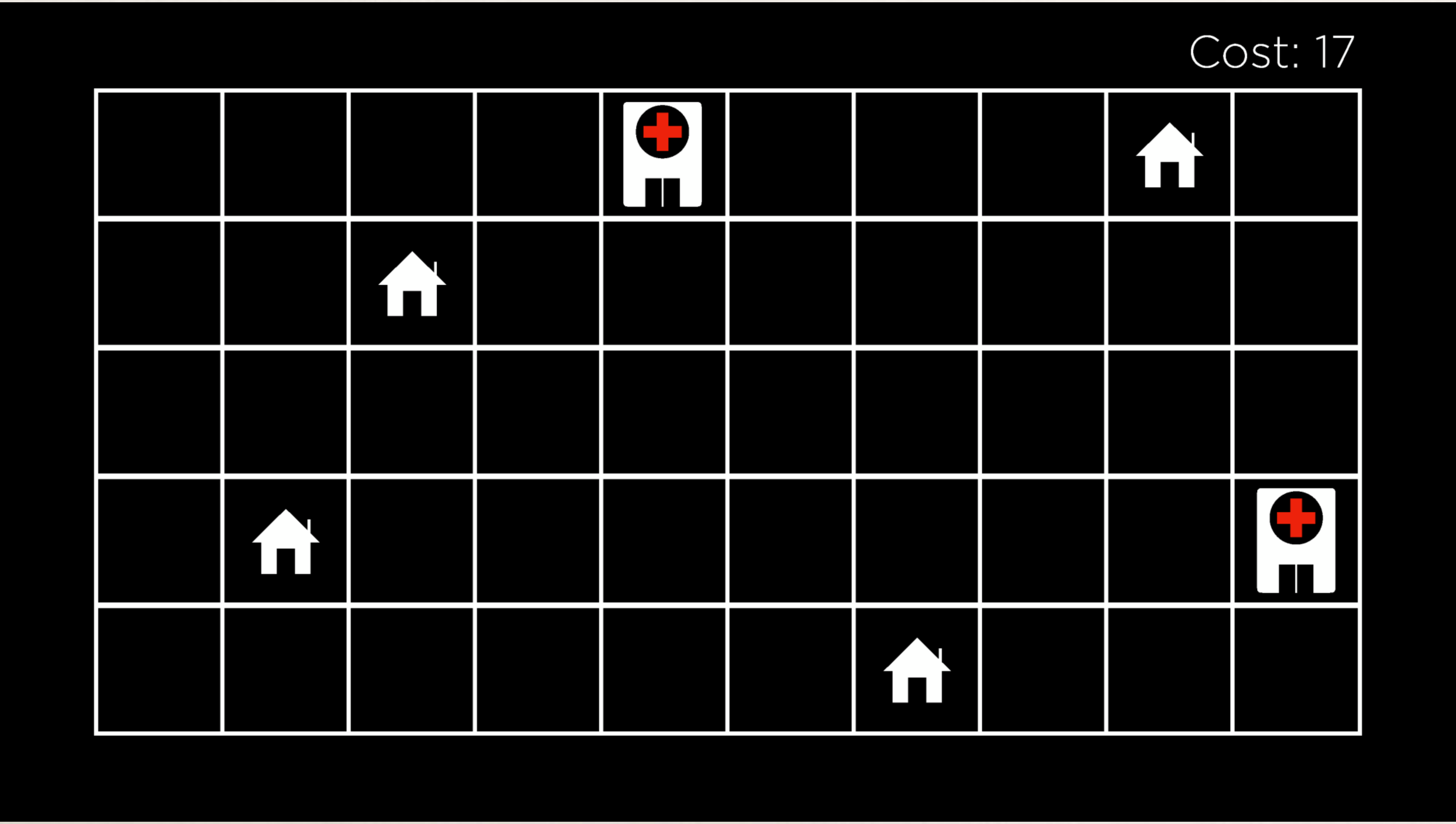
The n-queens problem

- The objective is to minimise the number of pairs of queens that attack each other
- Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts

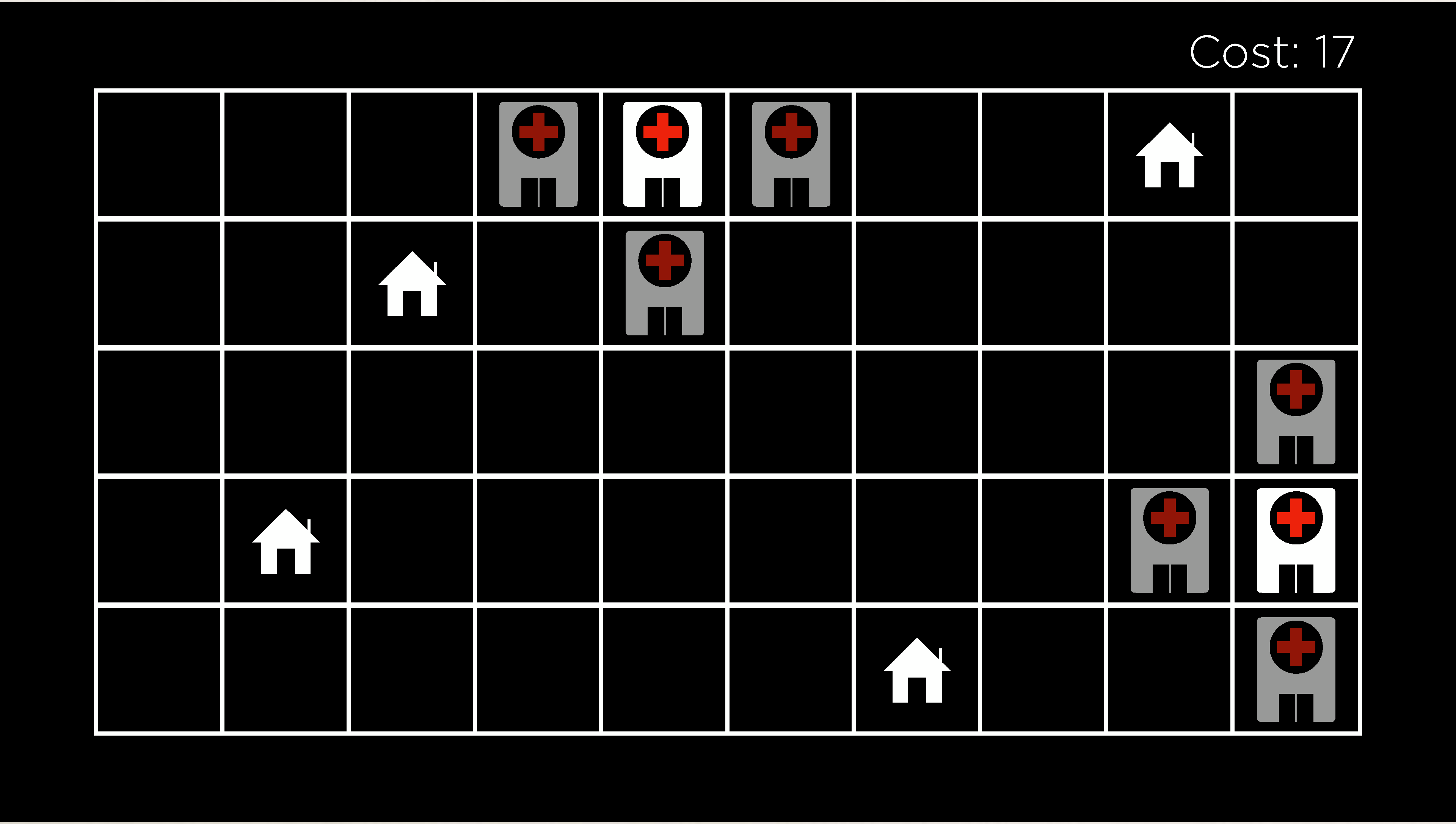


- Almost always solves *n*-queens problems almost instantaneously for very large *n*, e.g., $n = 1$ *million*

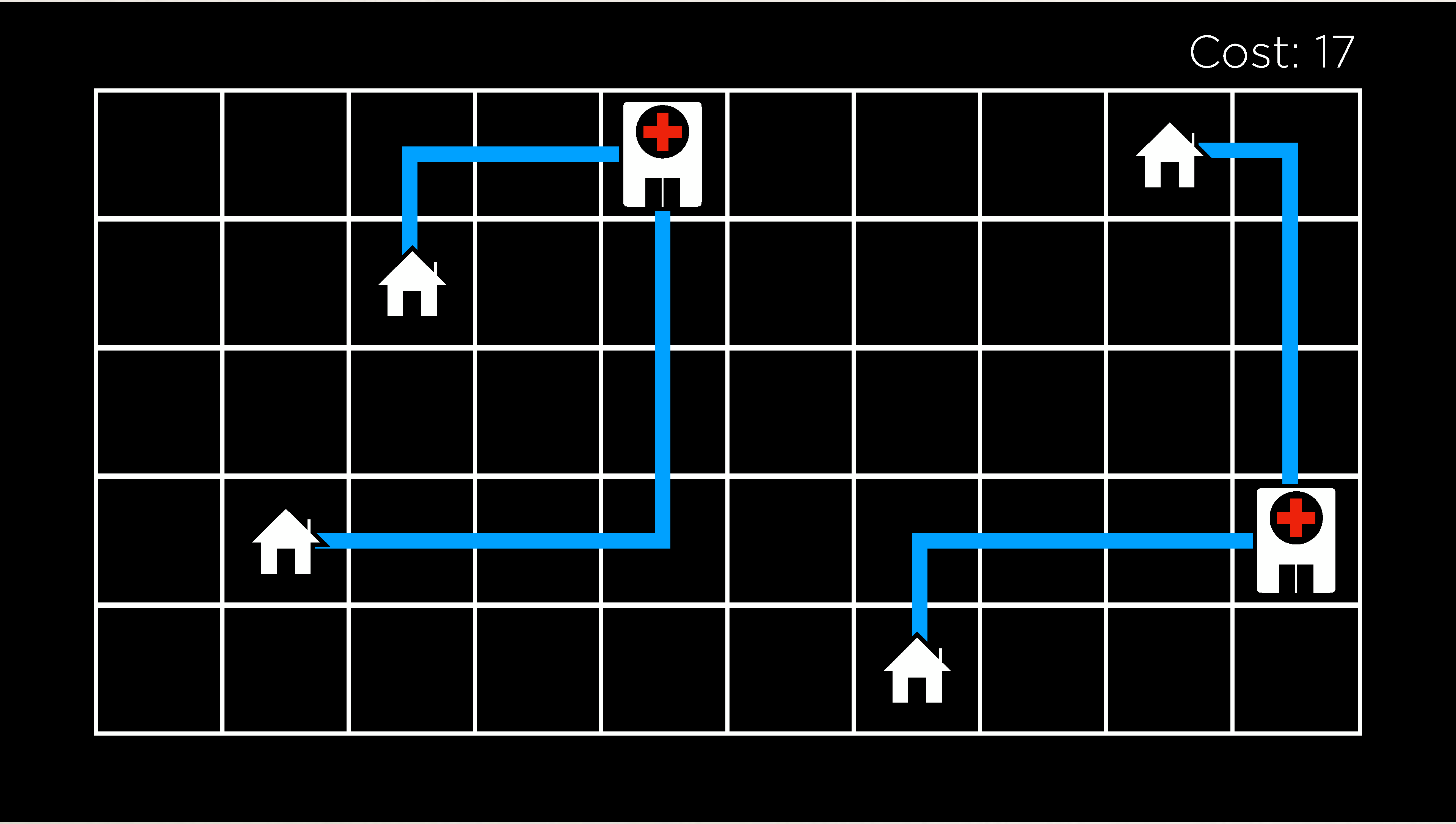
Hill-climbing search and optimisation



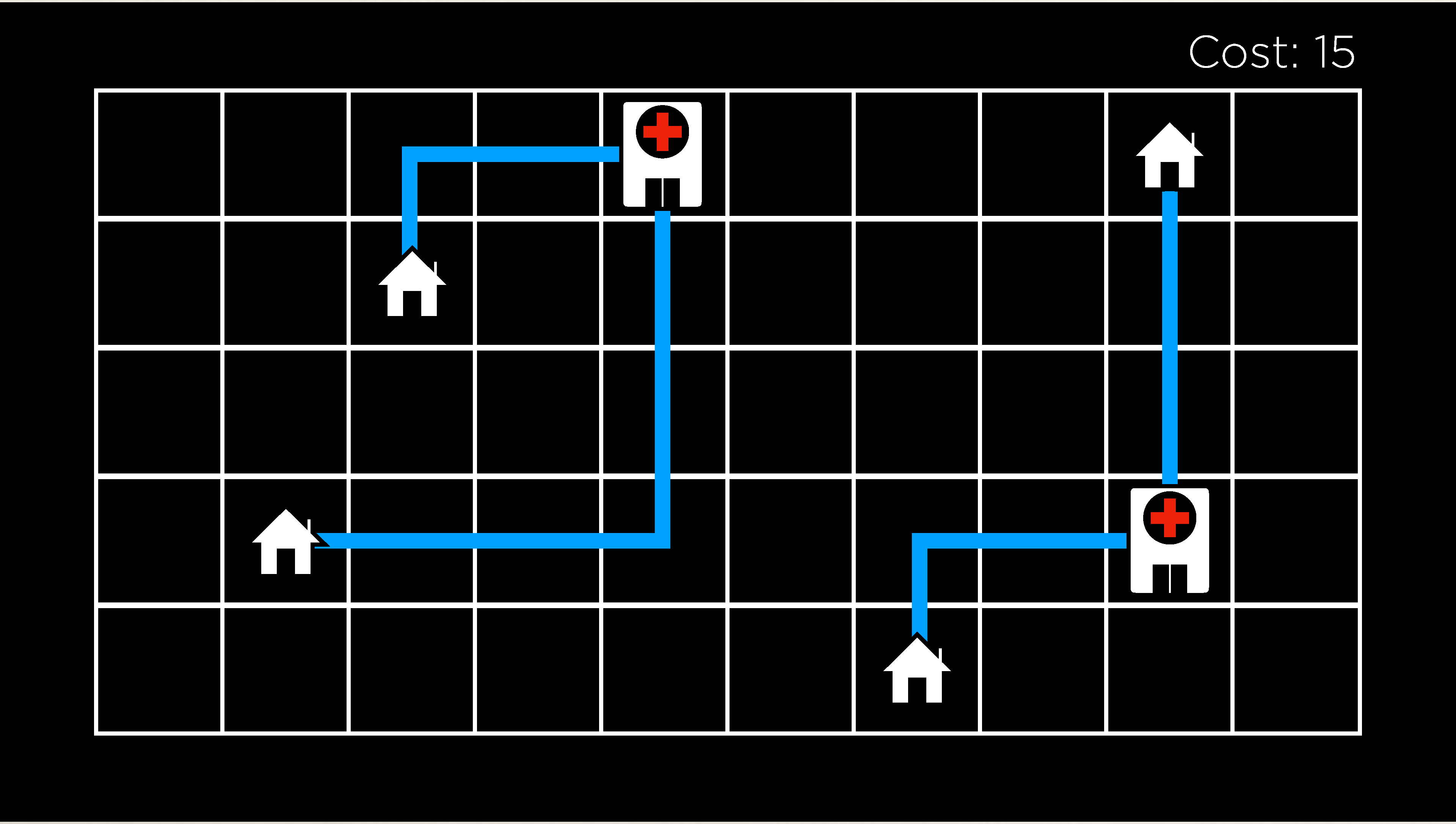
Hill-climbing search and optimisation



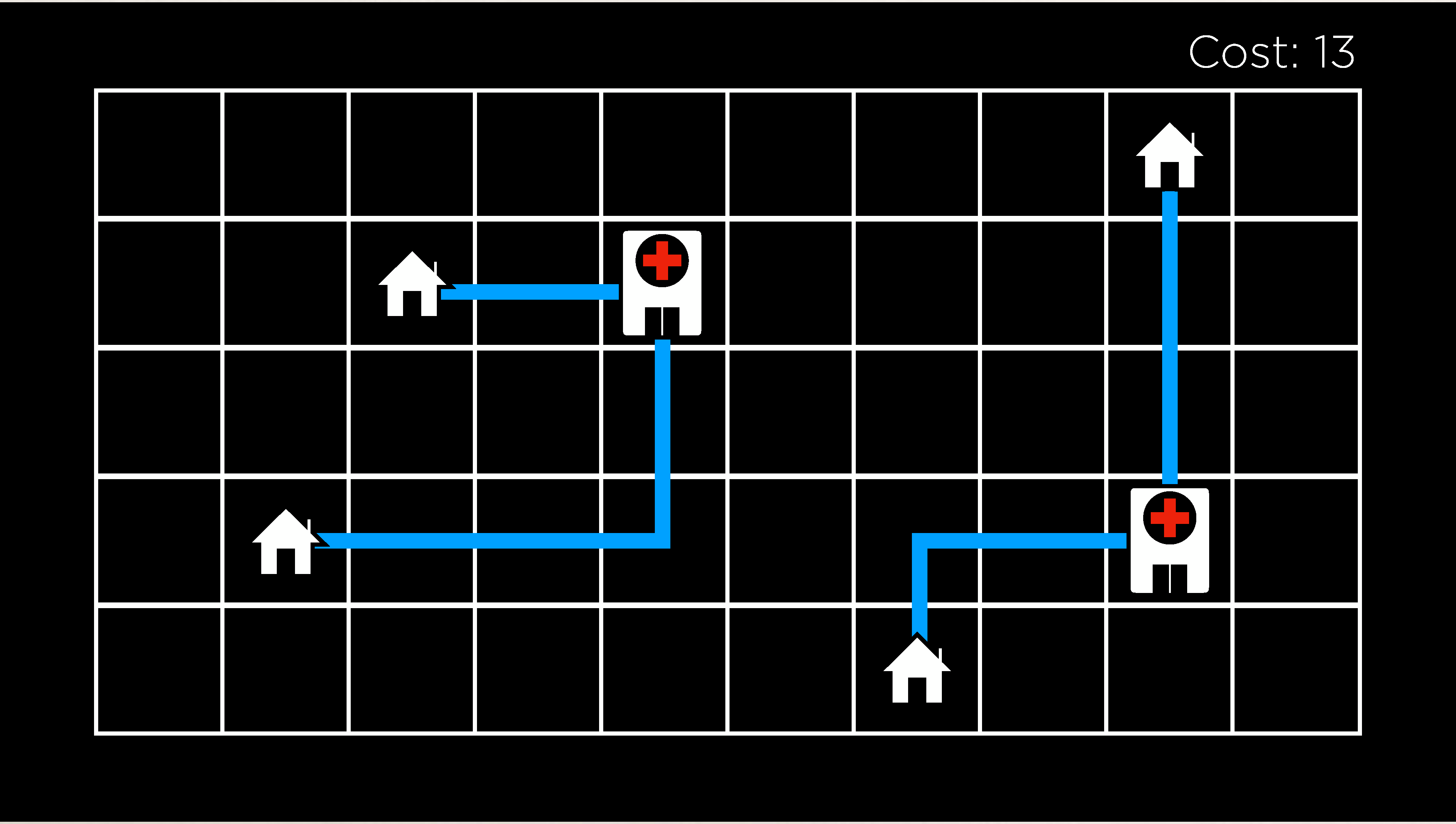
Hill-climbing search and optimisation



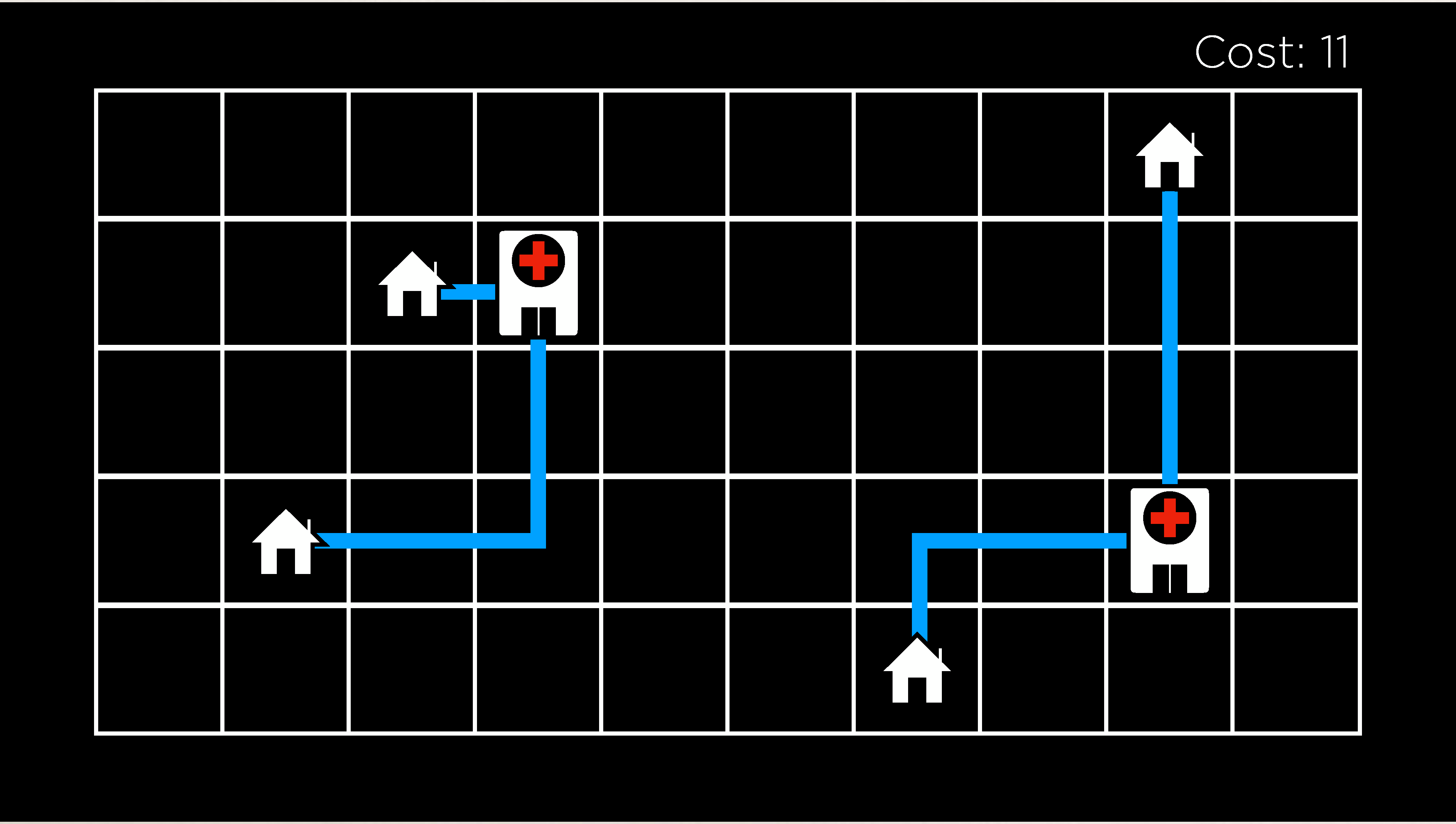
Hill-climbing search and optimisation



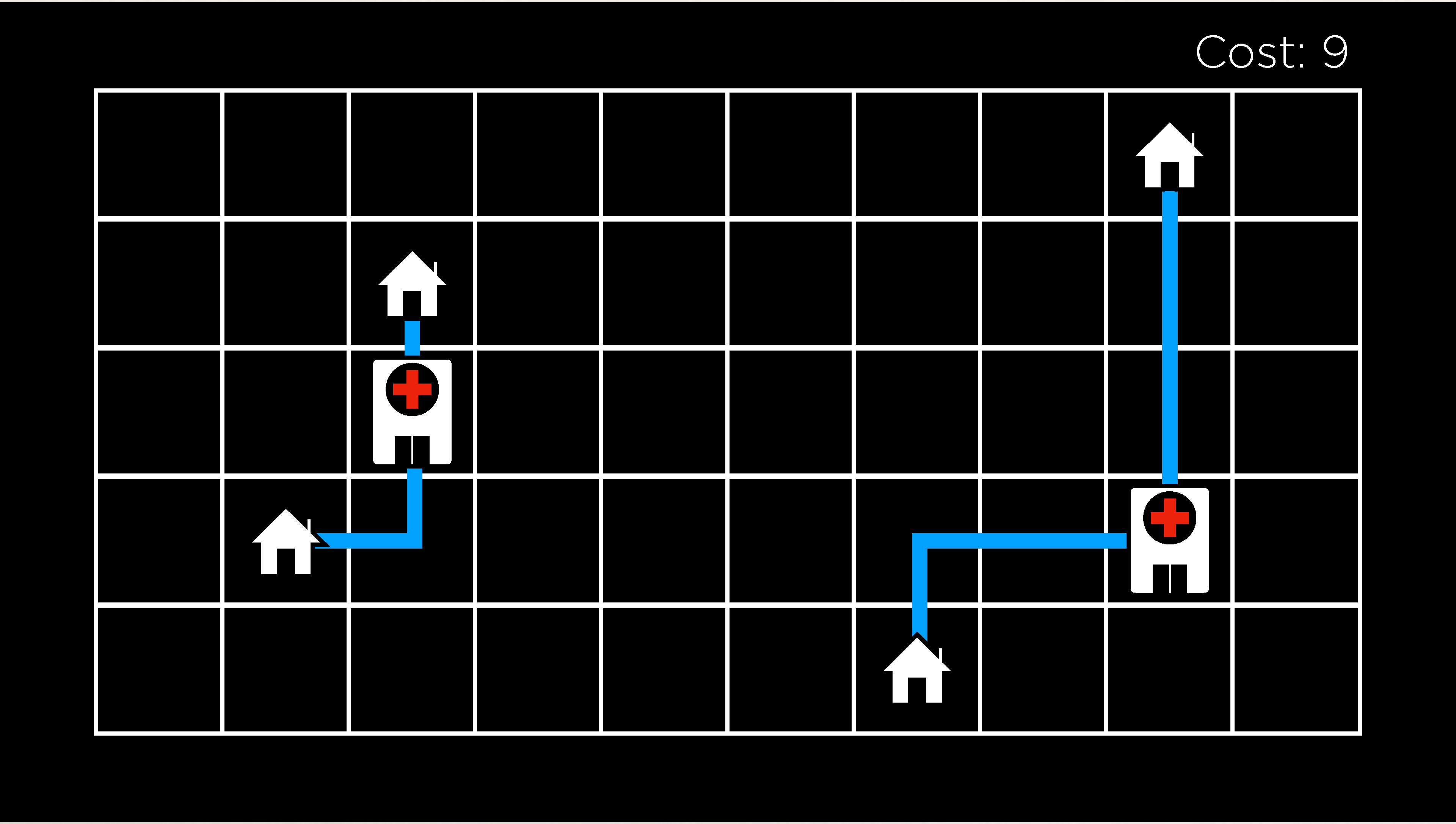
Hill-climbing search and optimisation



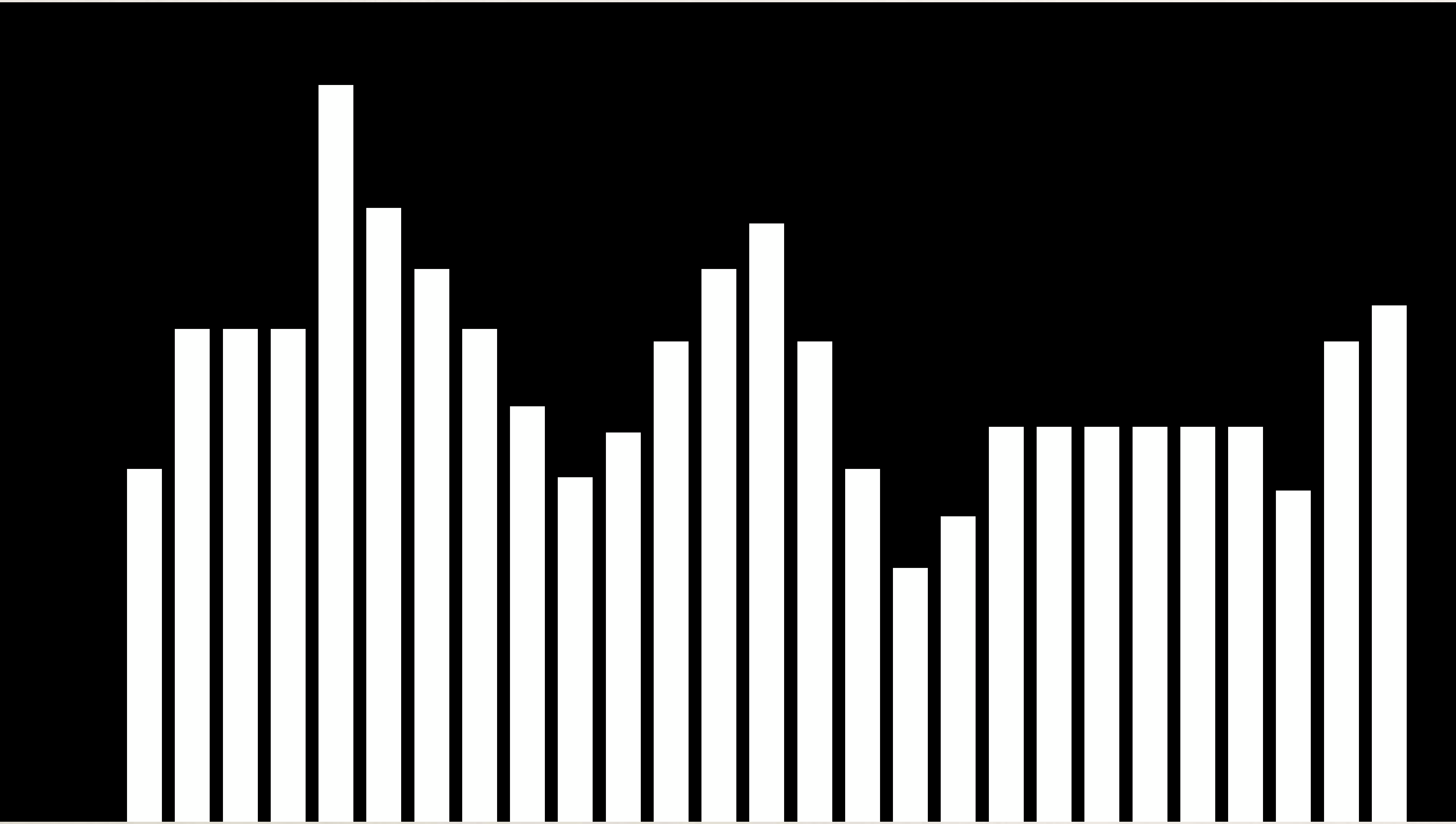
Hill-climbing search and optimisation



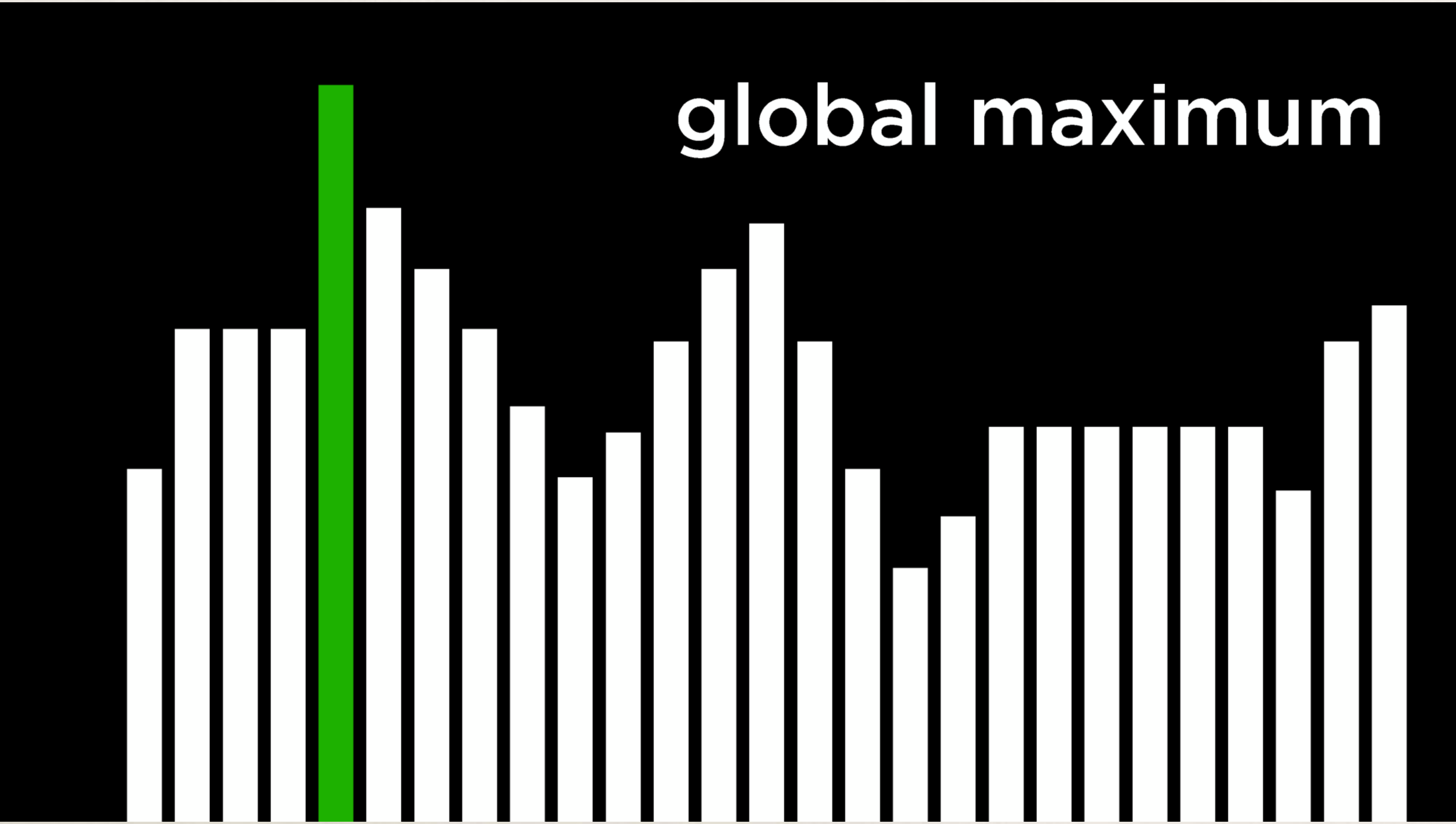
Hill-climbing search and optimisation



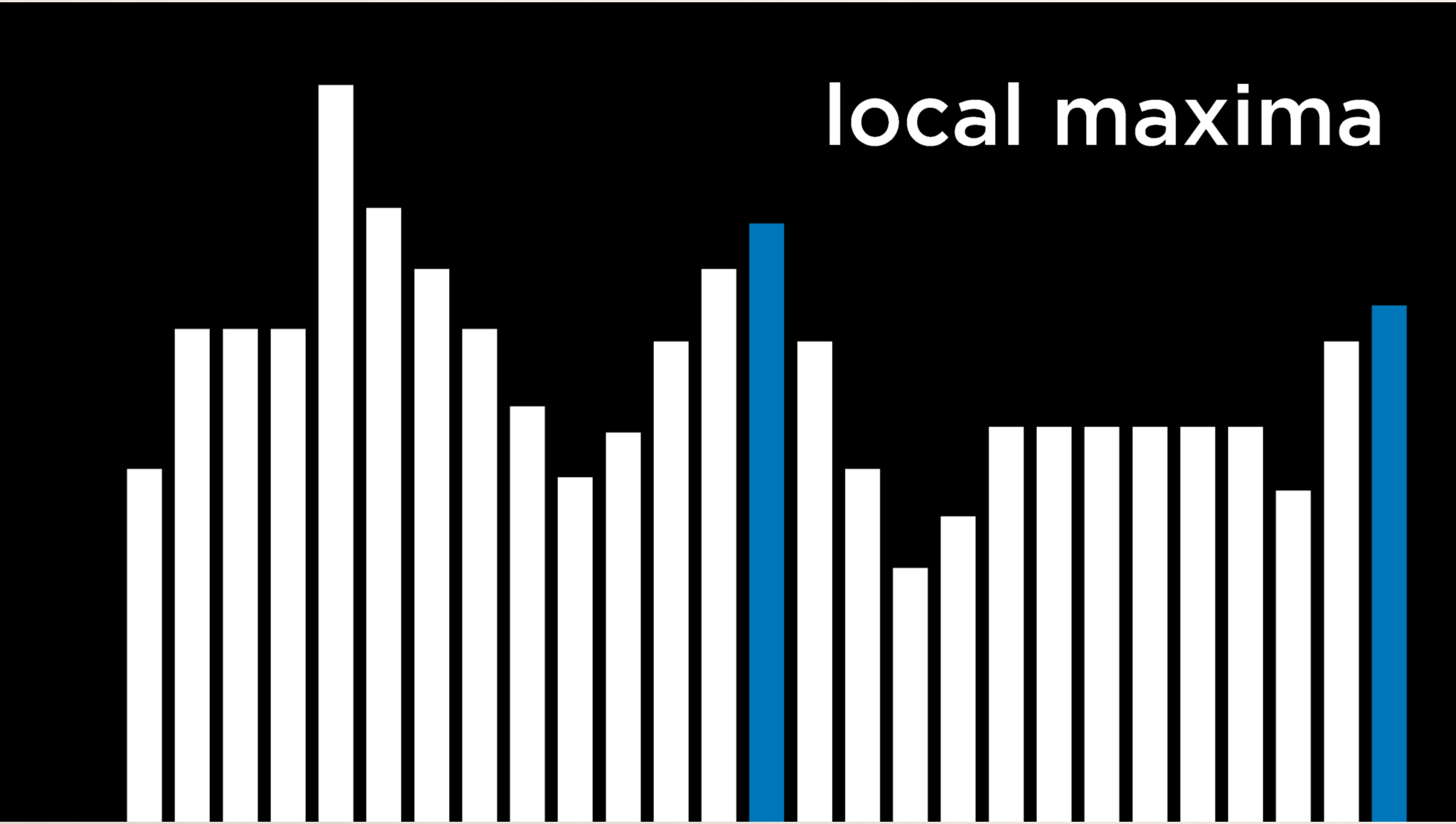
Hill-climbing search and optimisation



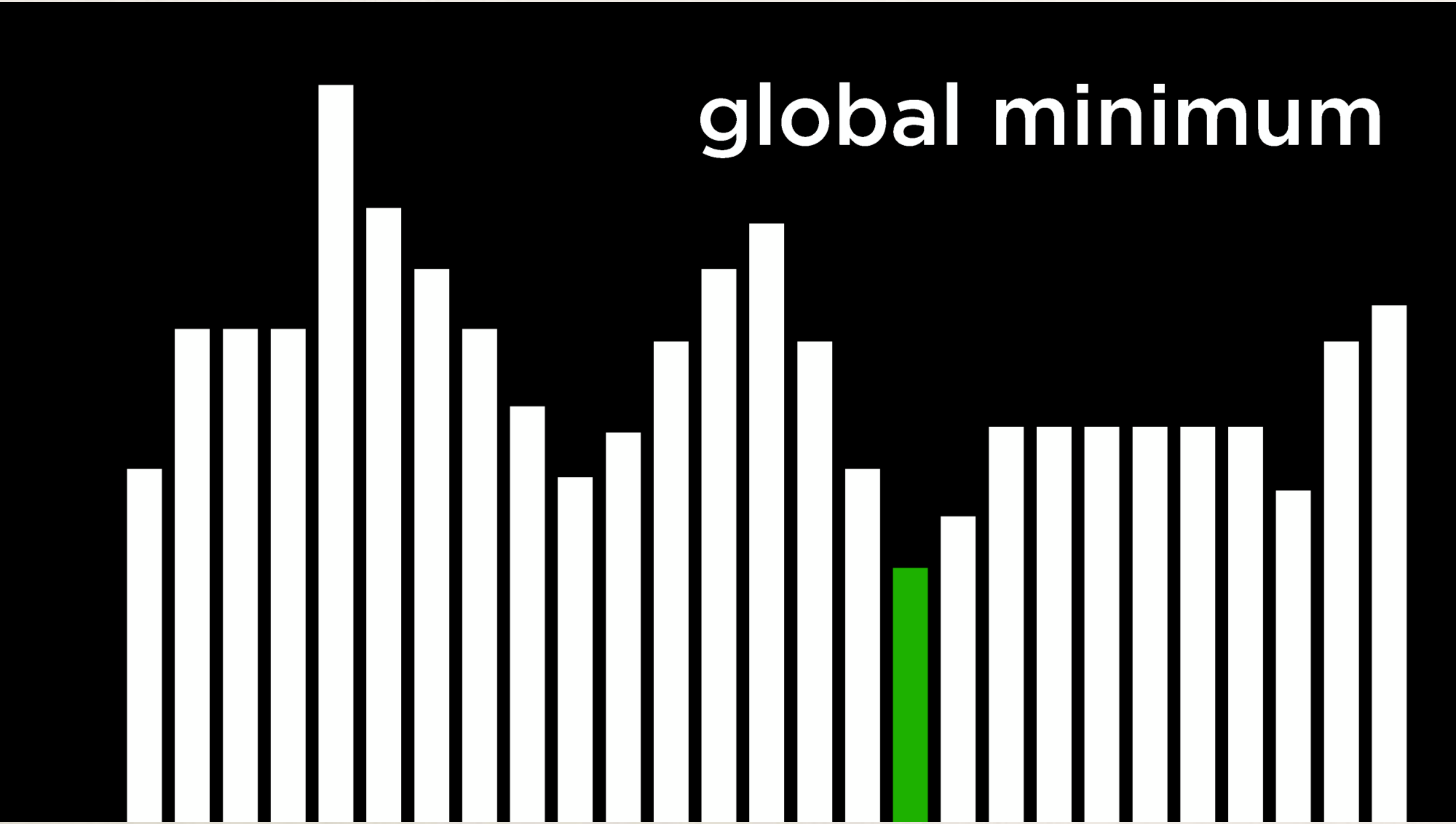
Hill-climbing search and optimisation



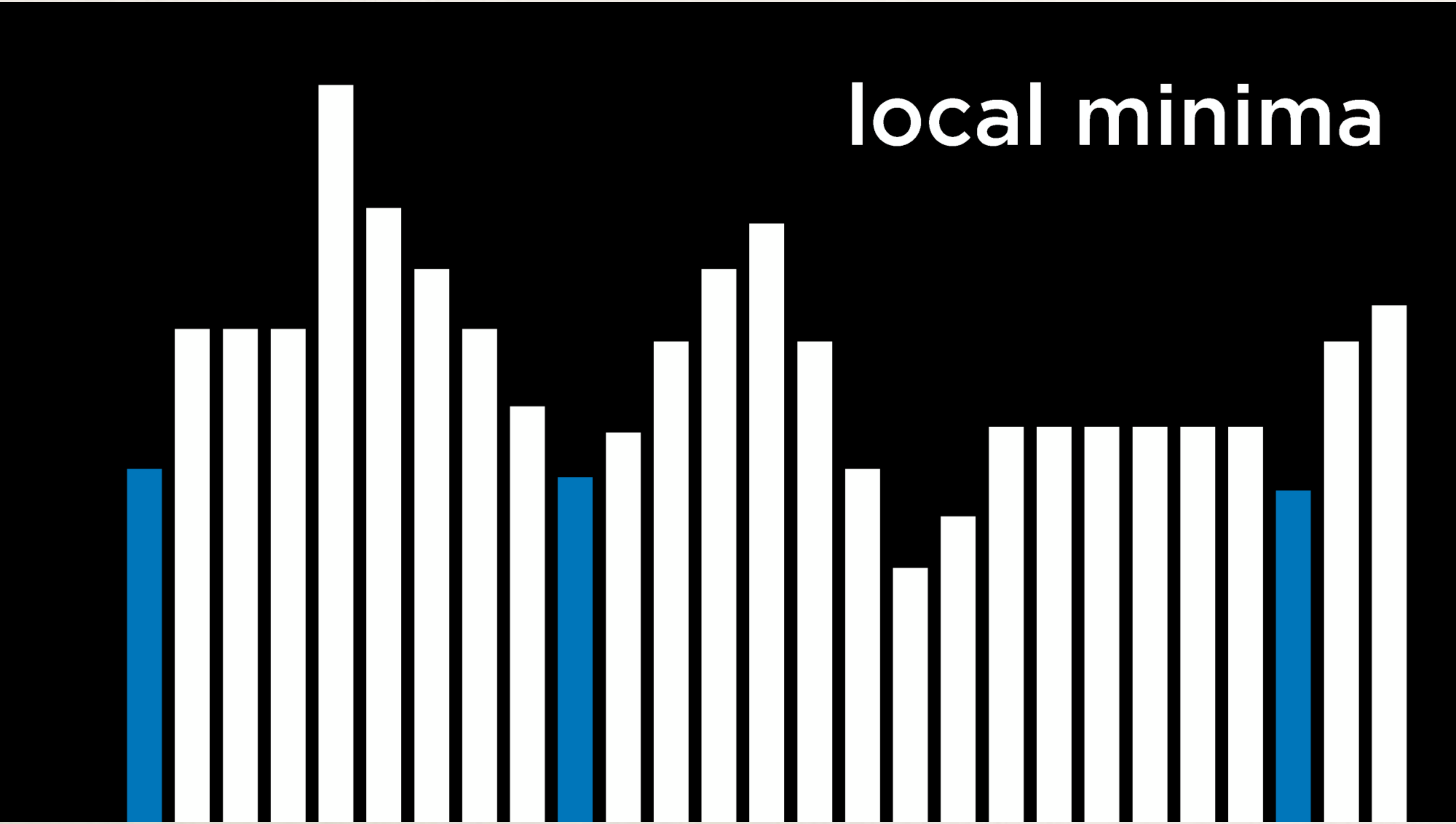
Hill-climbing search and optimisation



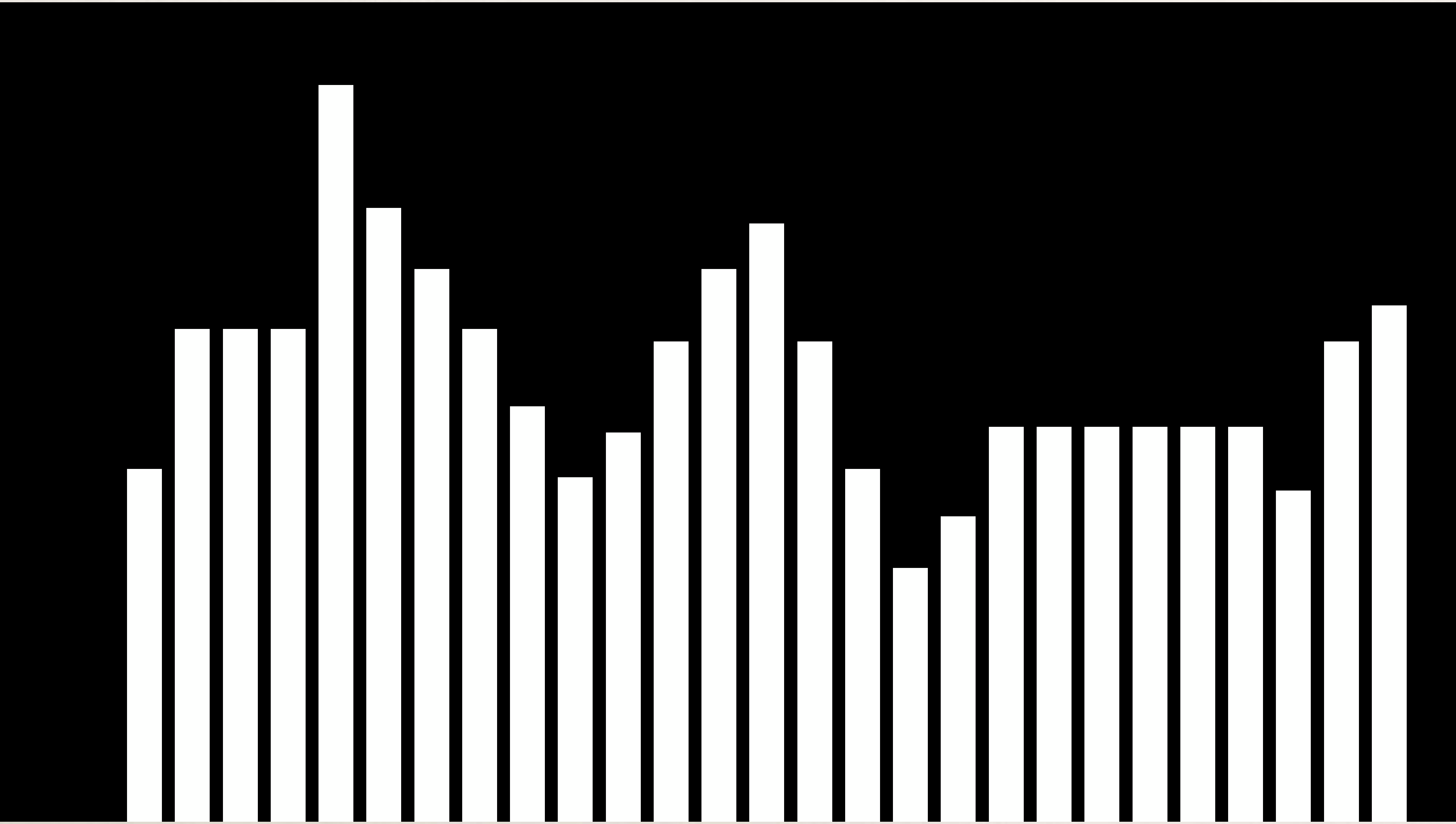
Hill-climbing search and optimisation



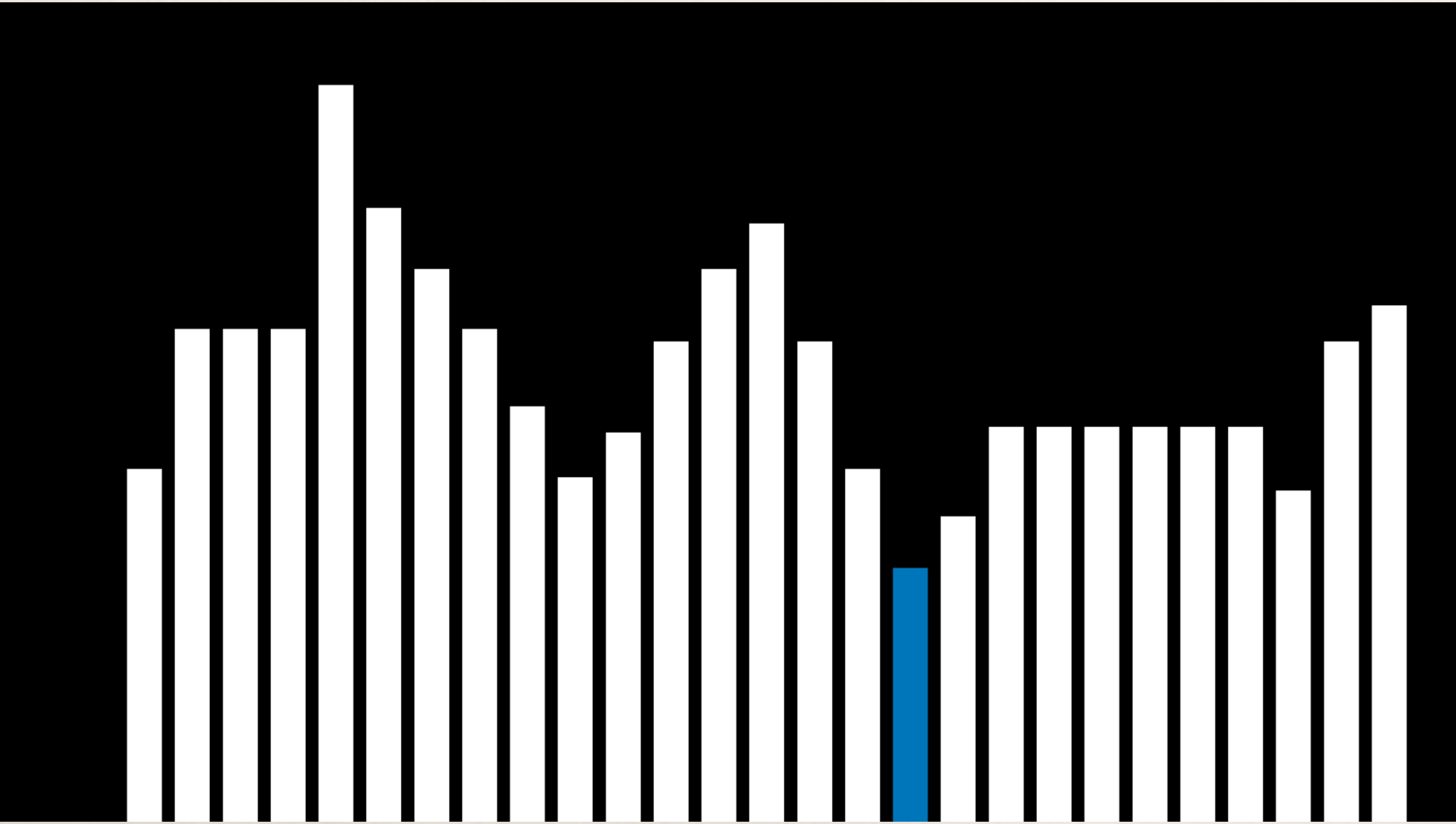
Hill-climbing search and optimisation



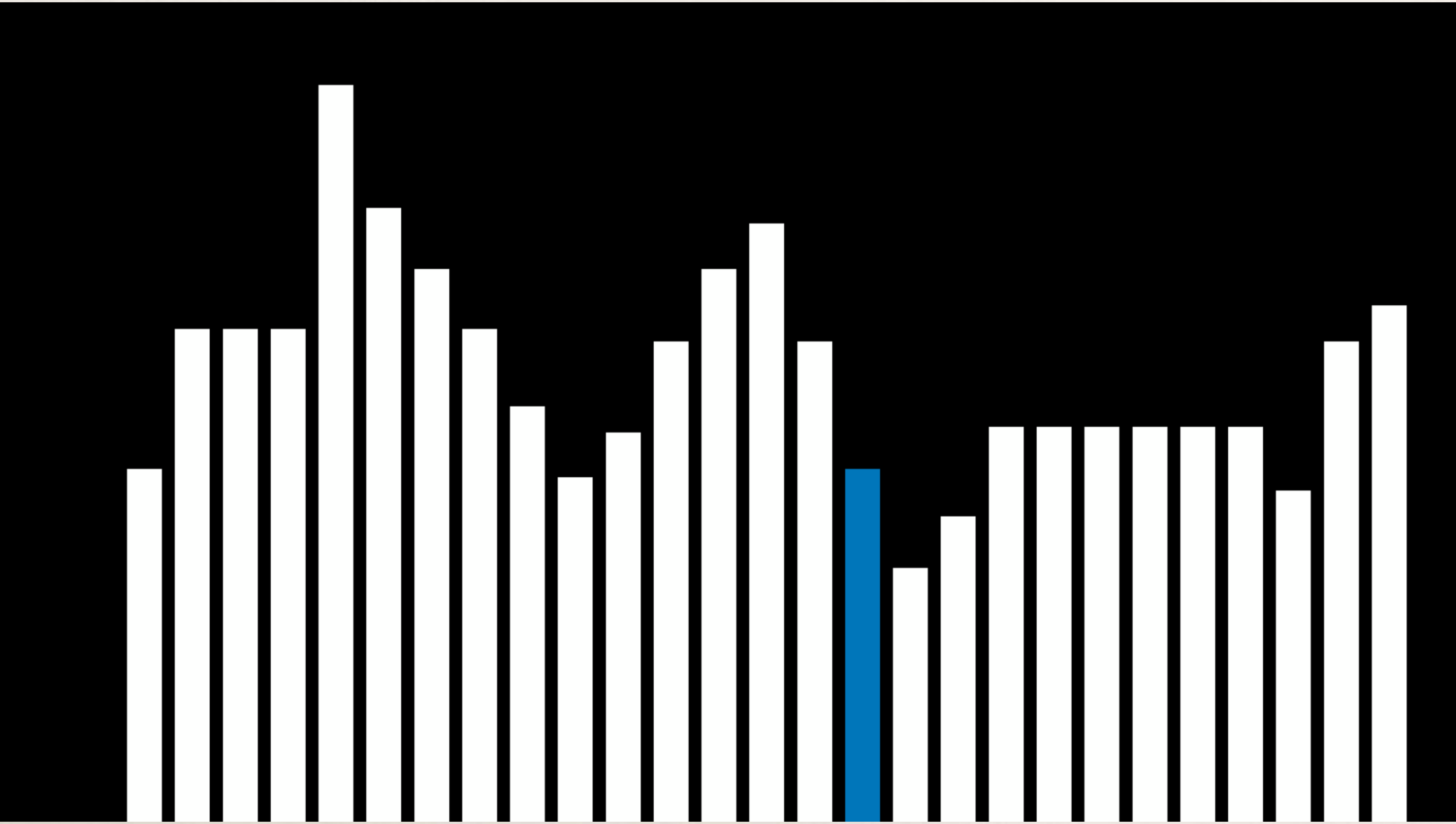
Hill-climbing search and optimisation



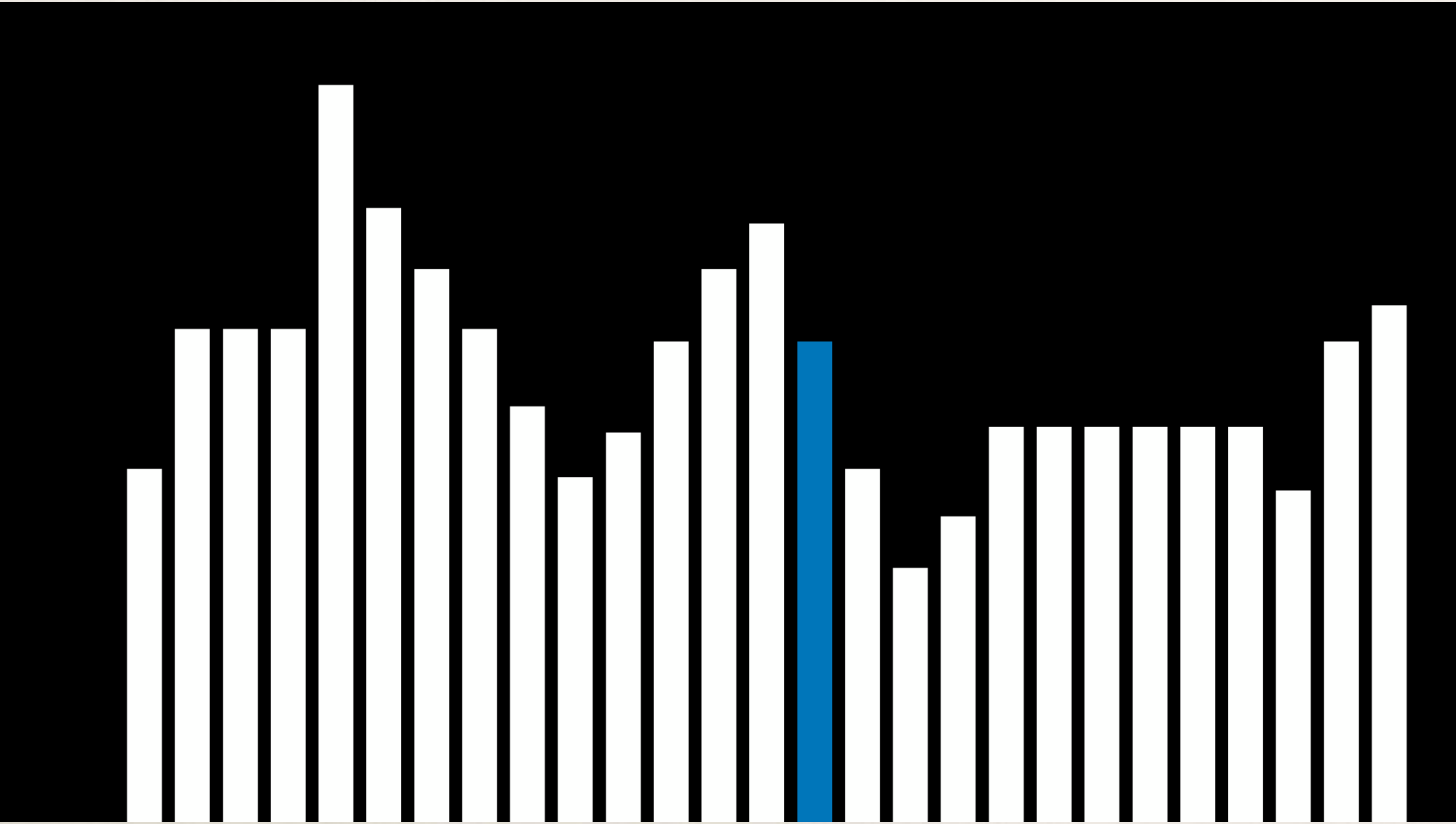
Hill-climbing search and optimisation



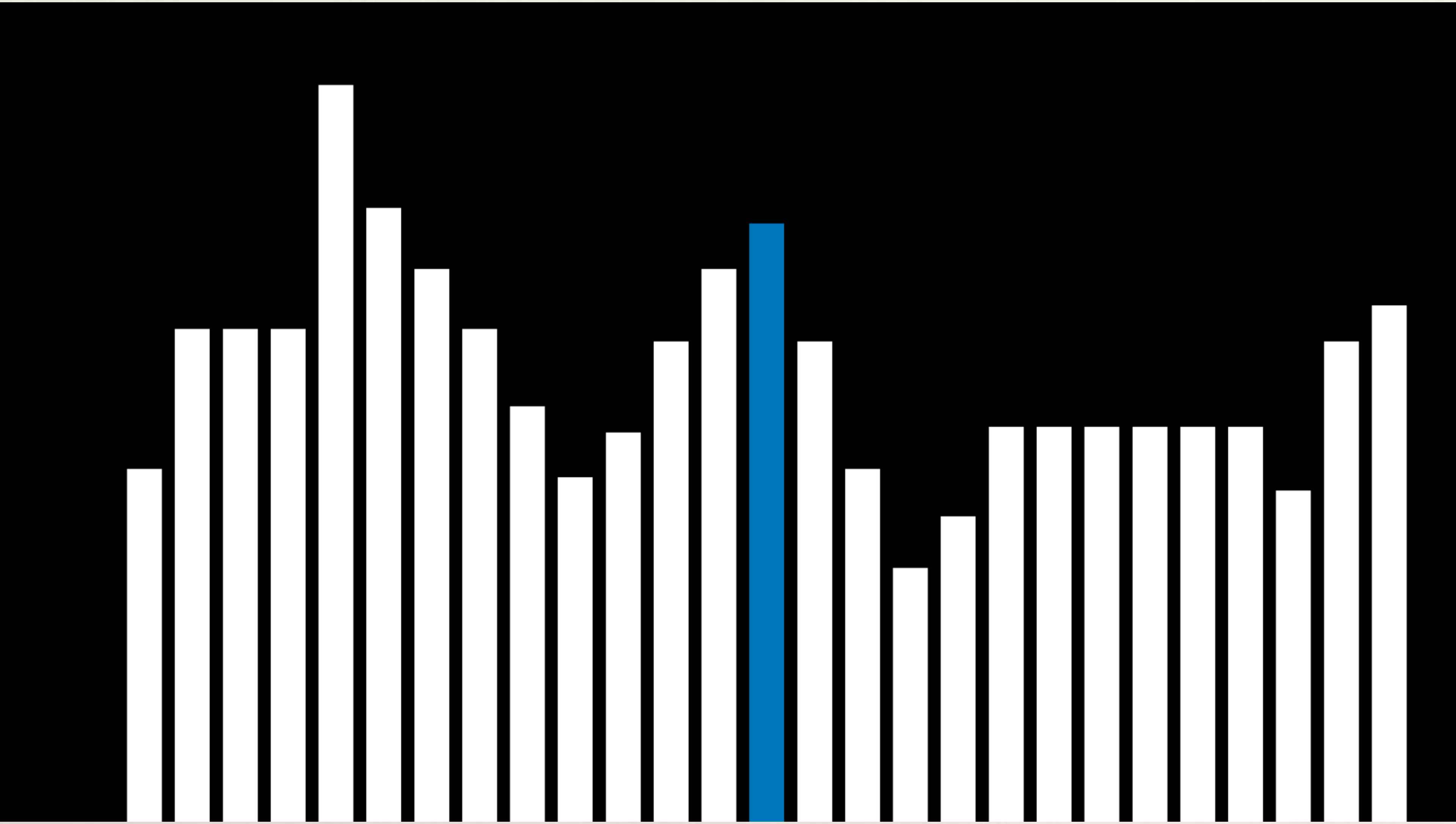
Hill-climbing search and optimisation



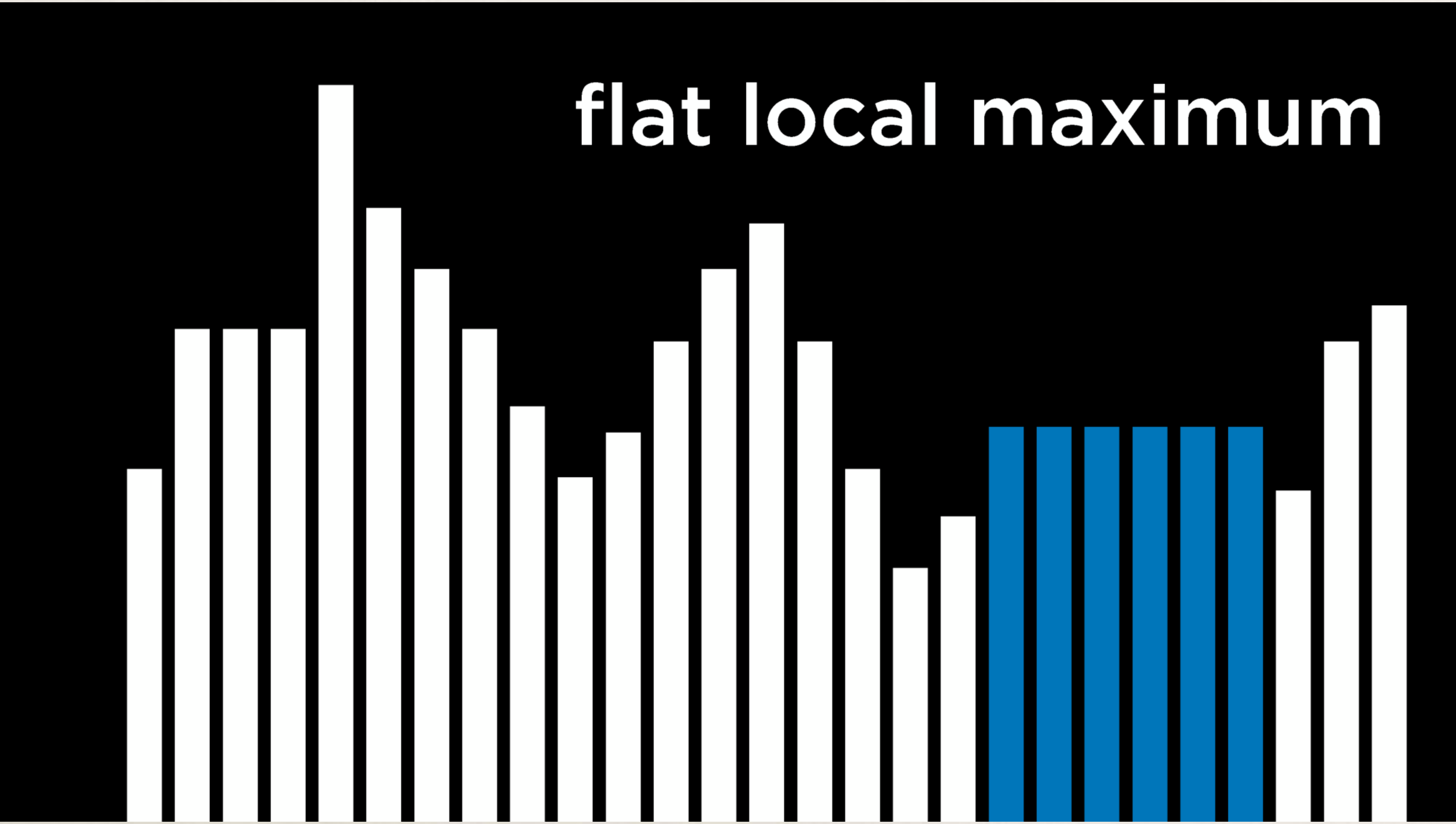
Hill-climbing search and optimisation



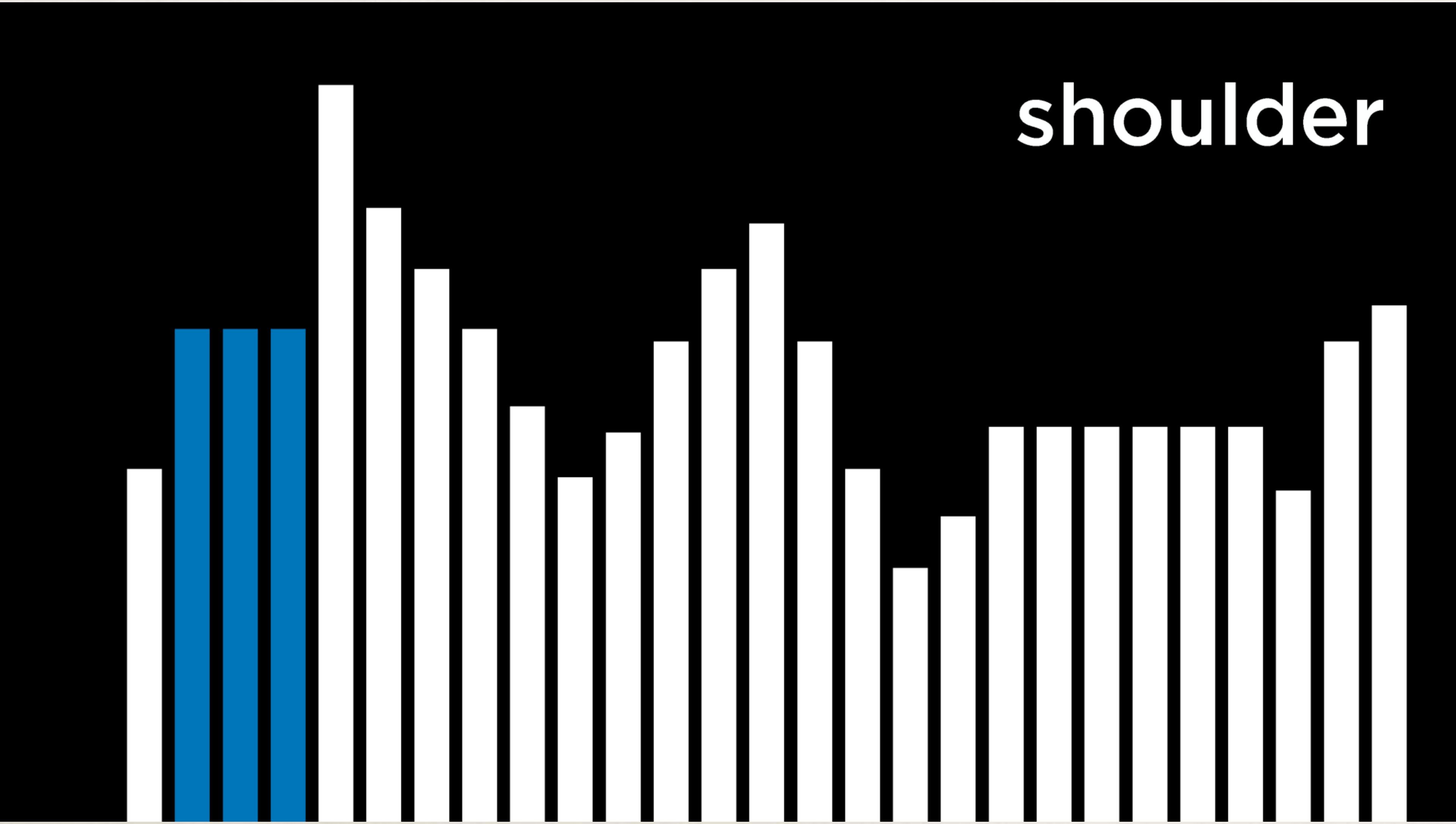
Hill-climbing search and optimisation



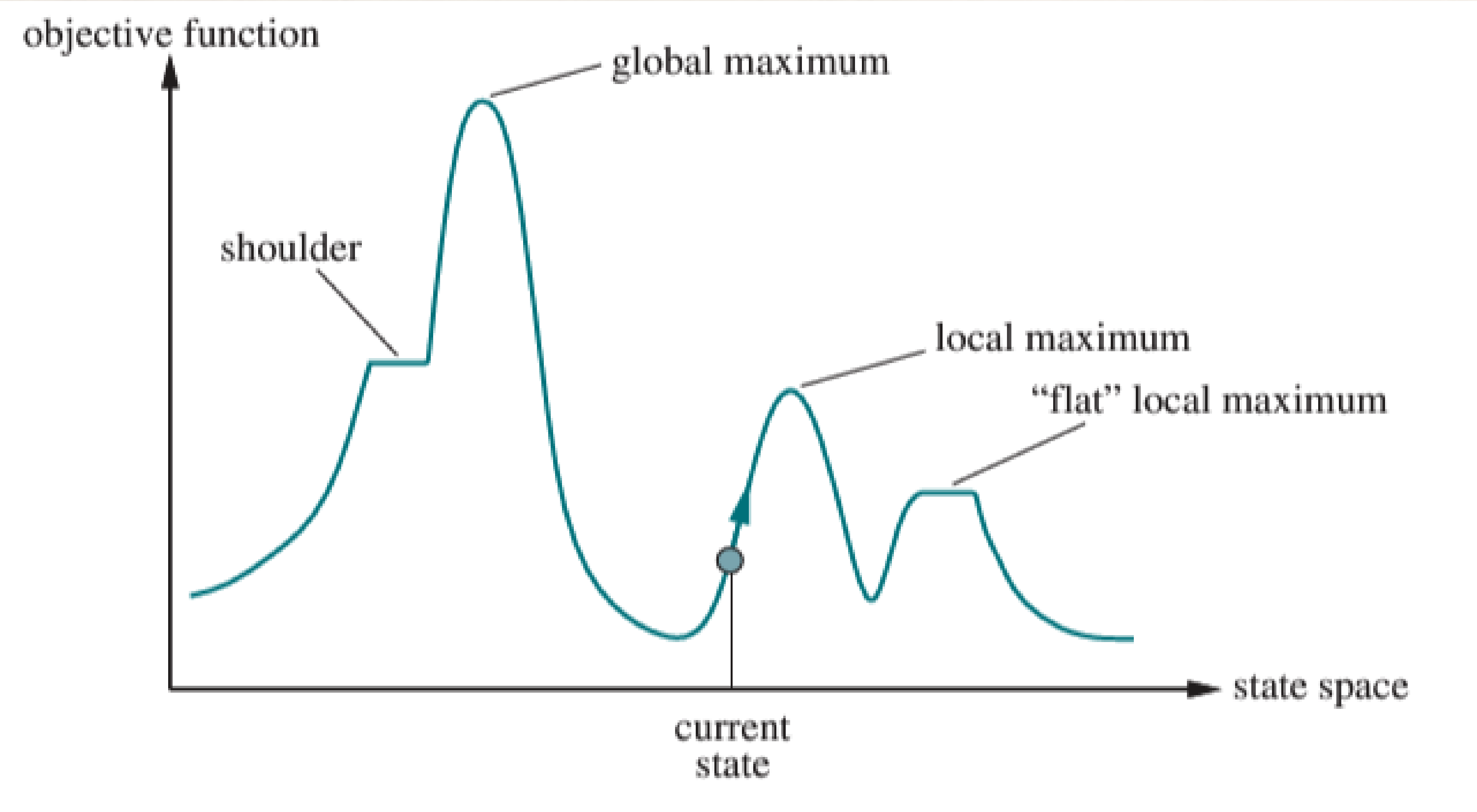
Hill-climbing search and optimisation



Hill-climbing search and optimisation



Hill-climbing search



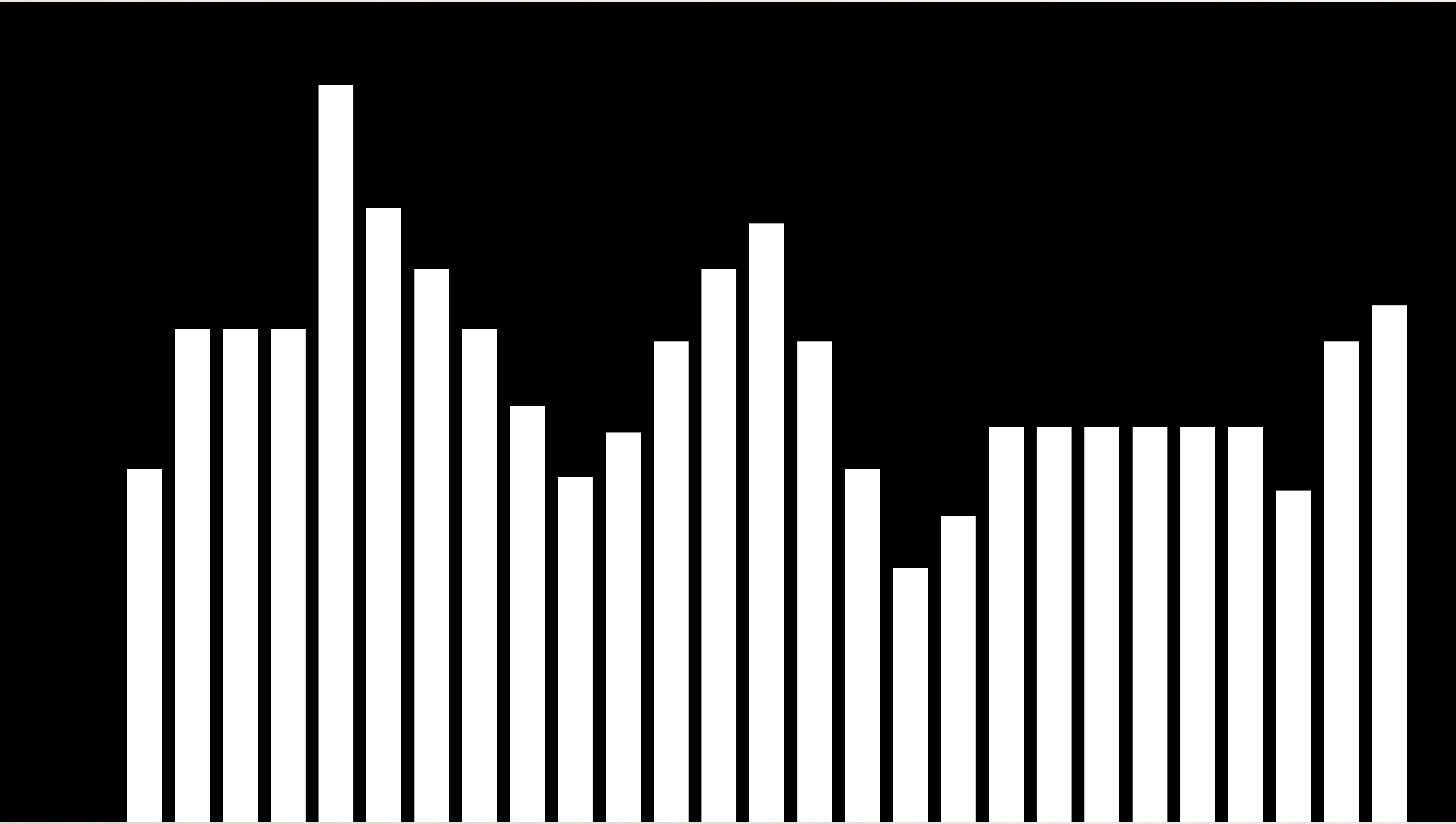
Variants of hill-climbing

- **Stochastic hill-climbing** chooses at random from uphill moves
  - Probability of selection may be based on size of improvement
- Stochastic hill-climbing finds good states more slowly but may find better solutions in some state spaces
- **First-choice hill-climbing** generates neighbours randomly and picks the first one that yields an improvement
  - Useful if the set of neighbours is very large
- Good, very simple option to improve on basic hill-climbing: **random-restart hill-climbing**
  - If each of n restarts has probability p of not finding the global maximum, the probability of failure is $1 - p^n$

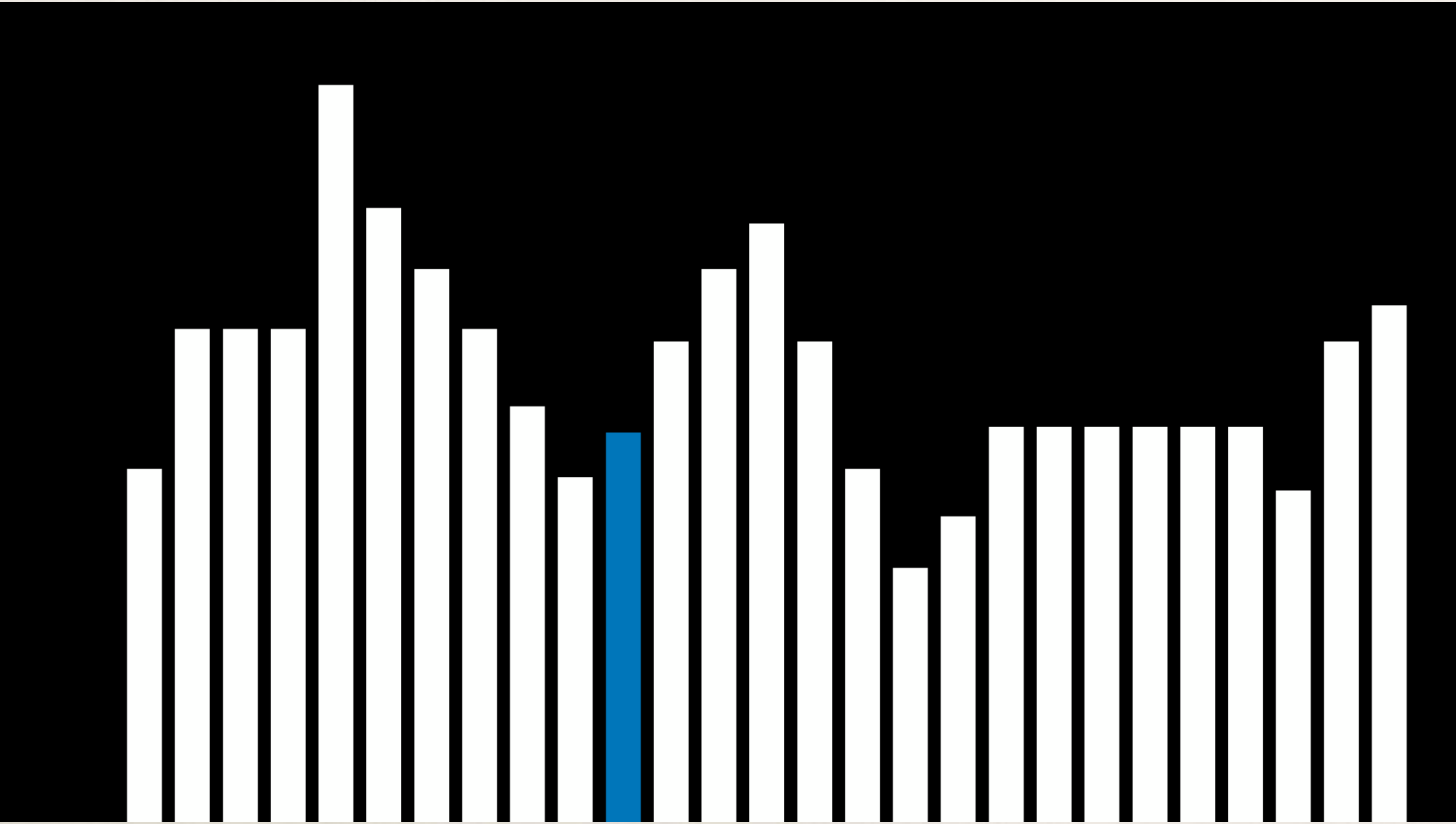
Simulated annealing

- In metallurgy, **annealing** is used to increase “ductility” and decrease hardness of a material to make it easier to work with
- Hot material’s temperature is gradually lowered according to a pre-defined scheduled to achieve this
- **Simulated annealing**: stochastic hill-climbing that allows downhill moves with a probability dependent on the temperature
- Early on, higher "temperature": more likely to accept neighbors that are worse than current state
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state

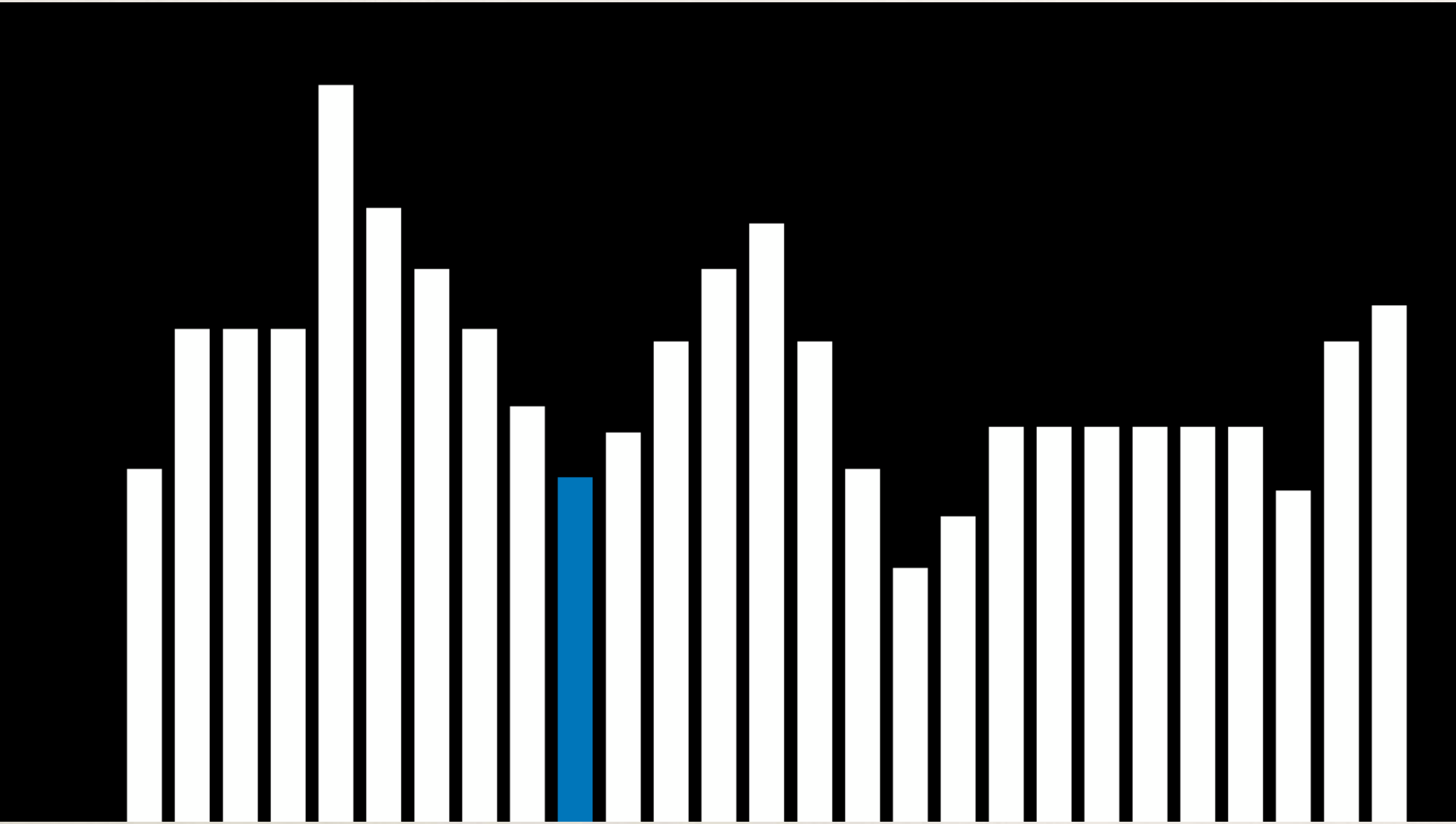
Simulated annealing



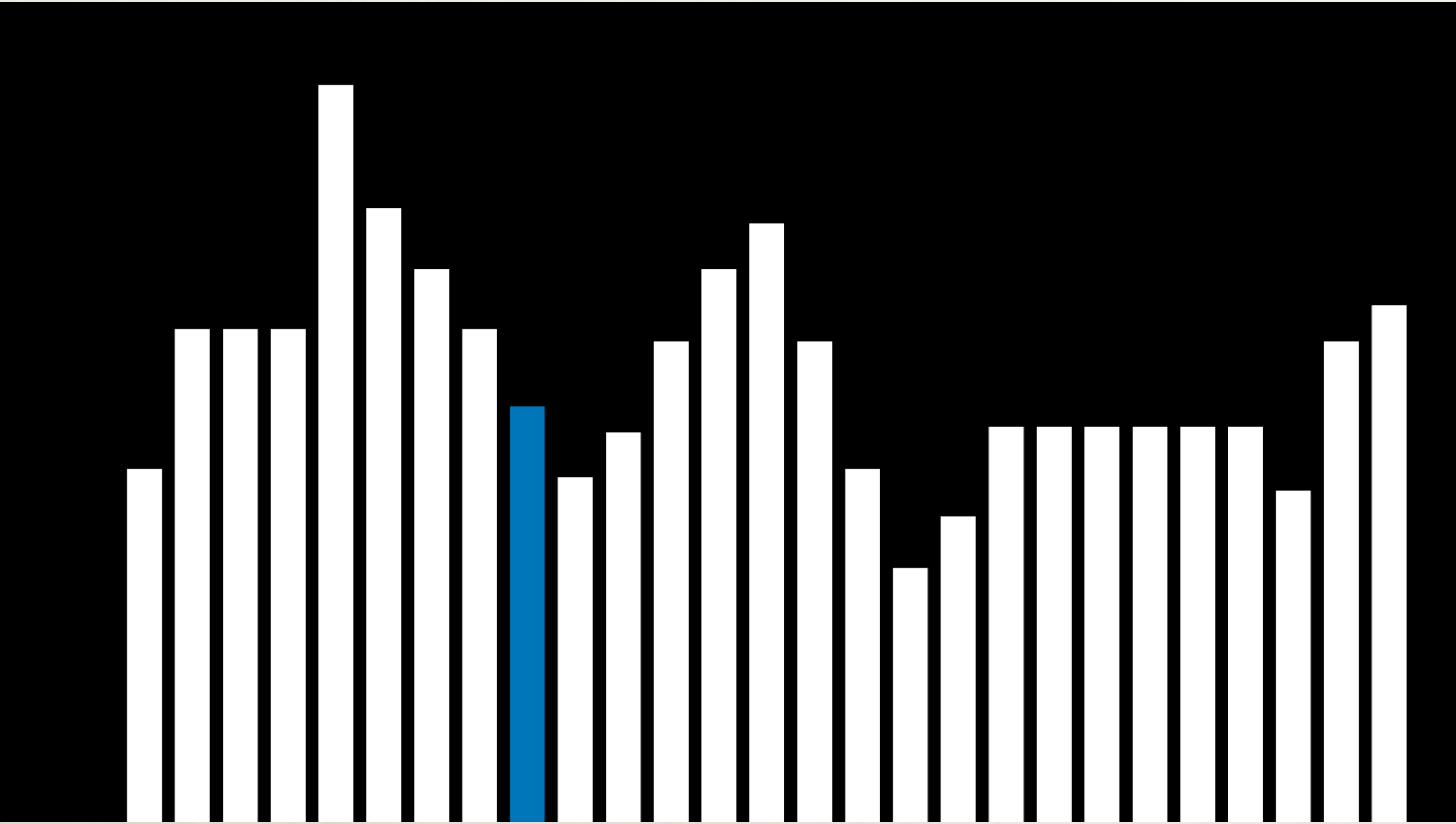
Simulated annealing



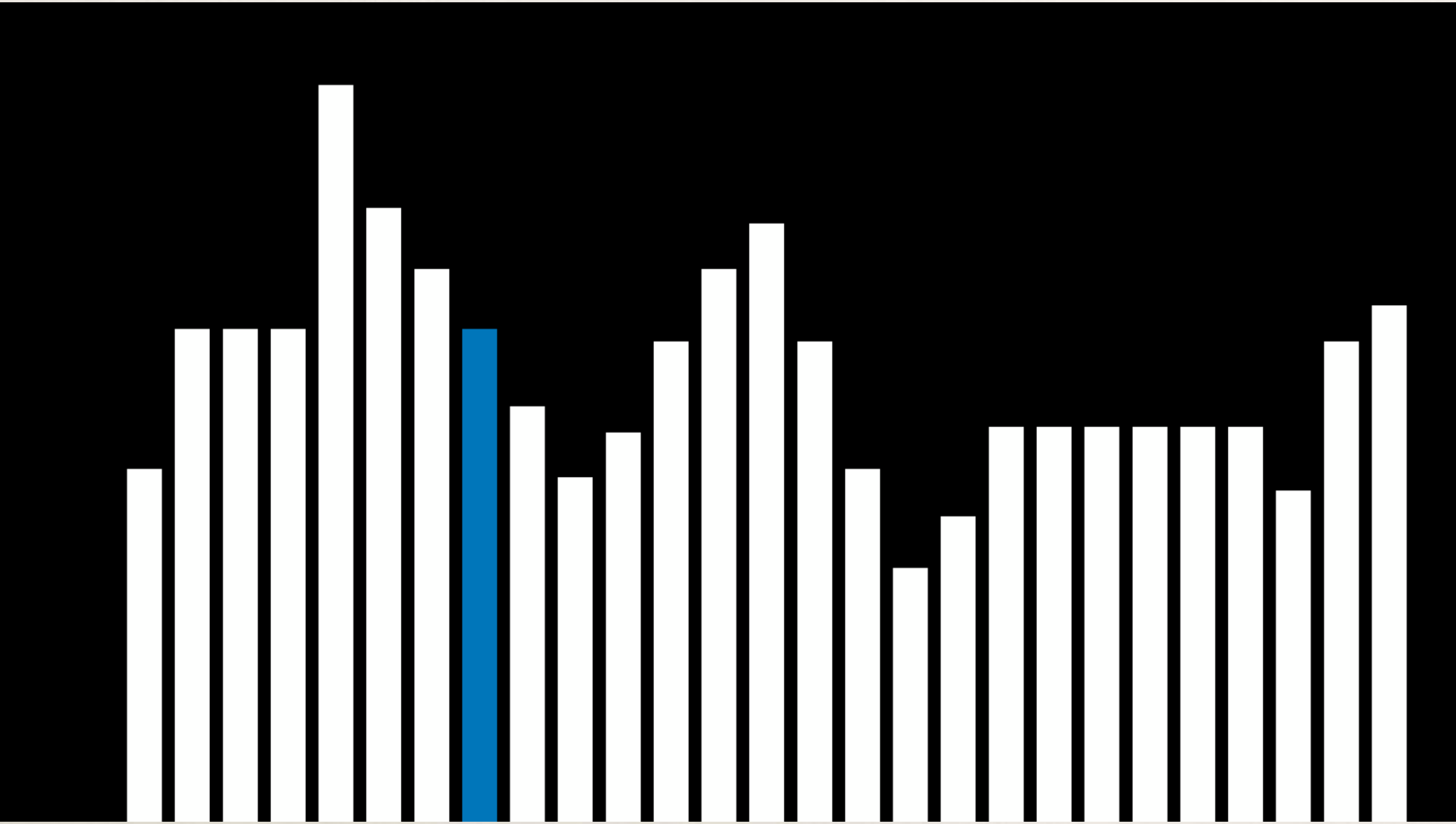
Simulated annealing



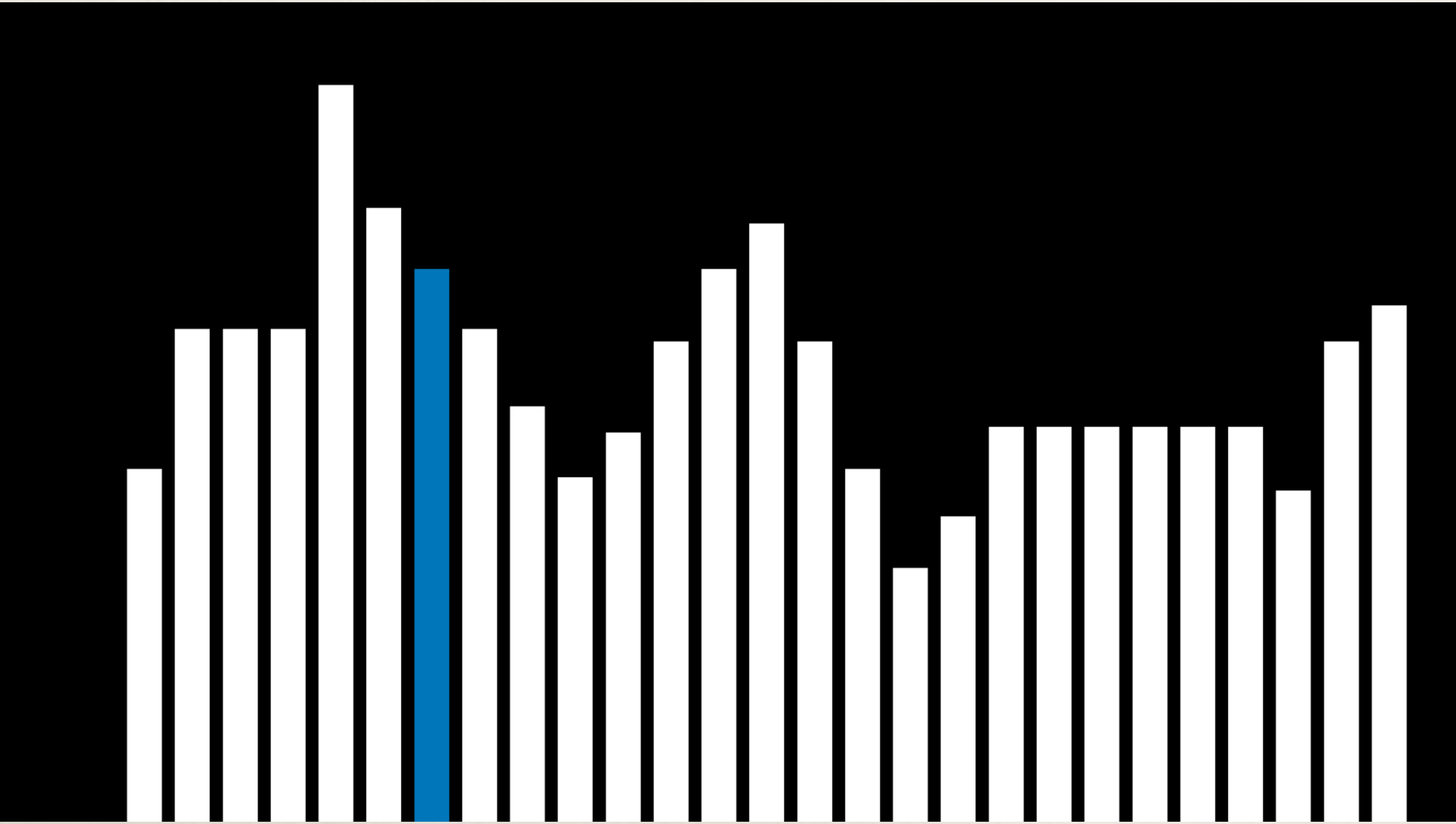
Simulated annealing



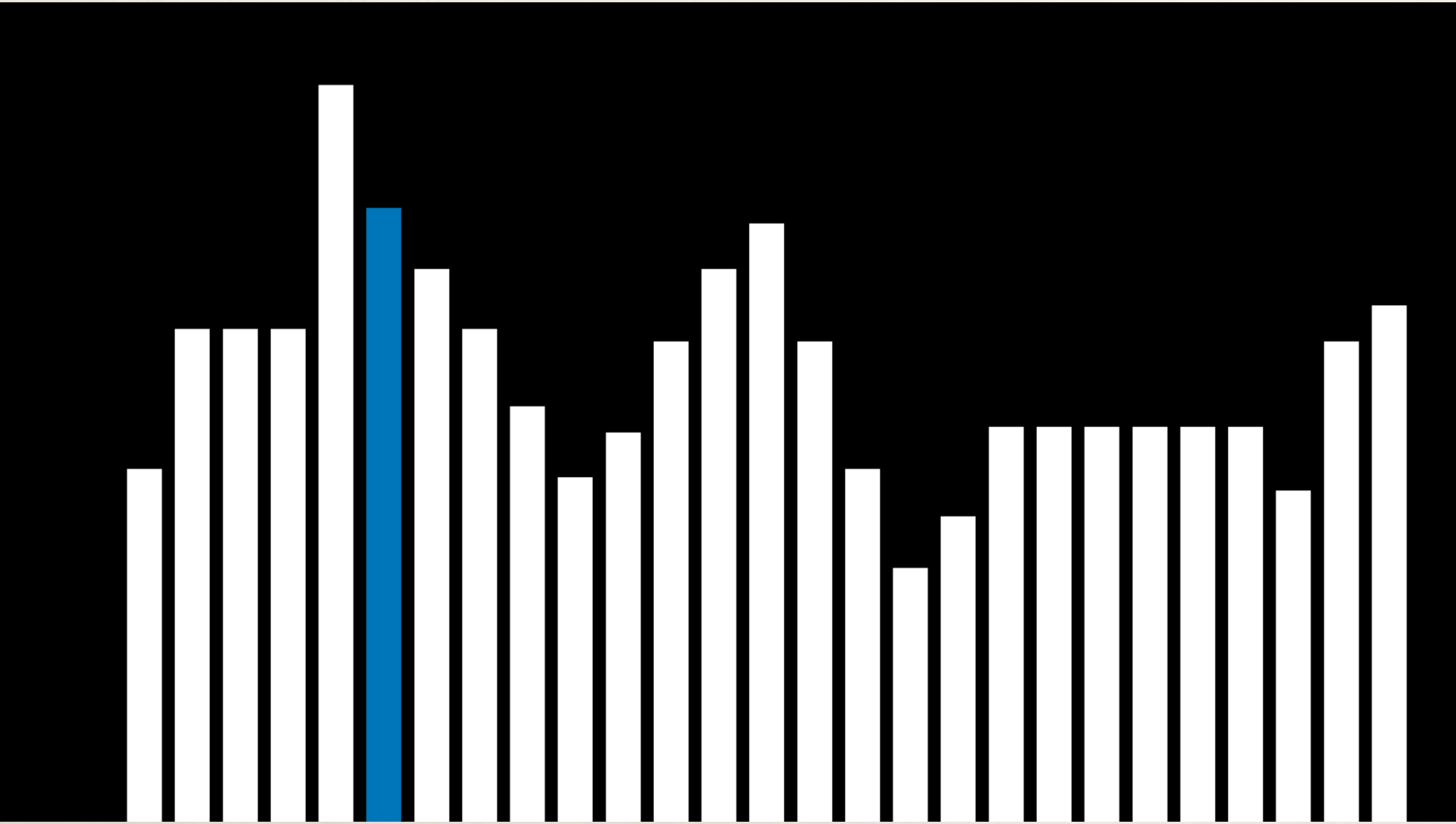
Simulated annealing



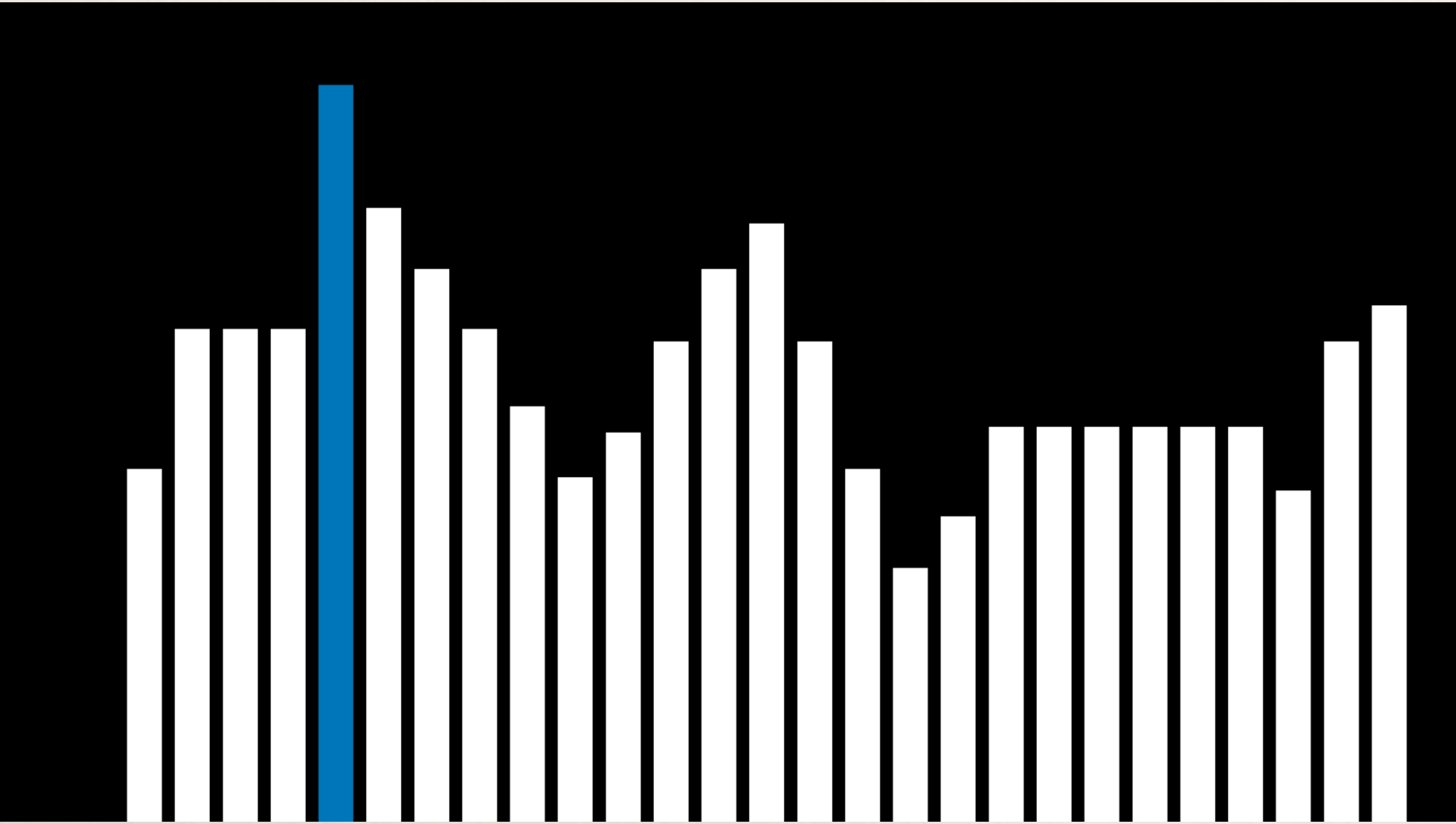
Simulated annealing



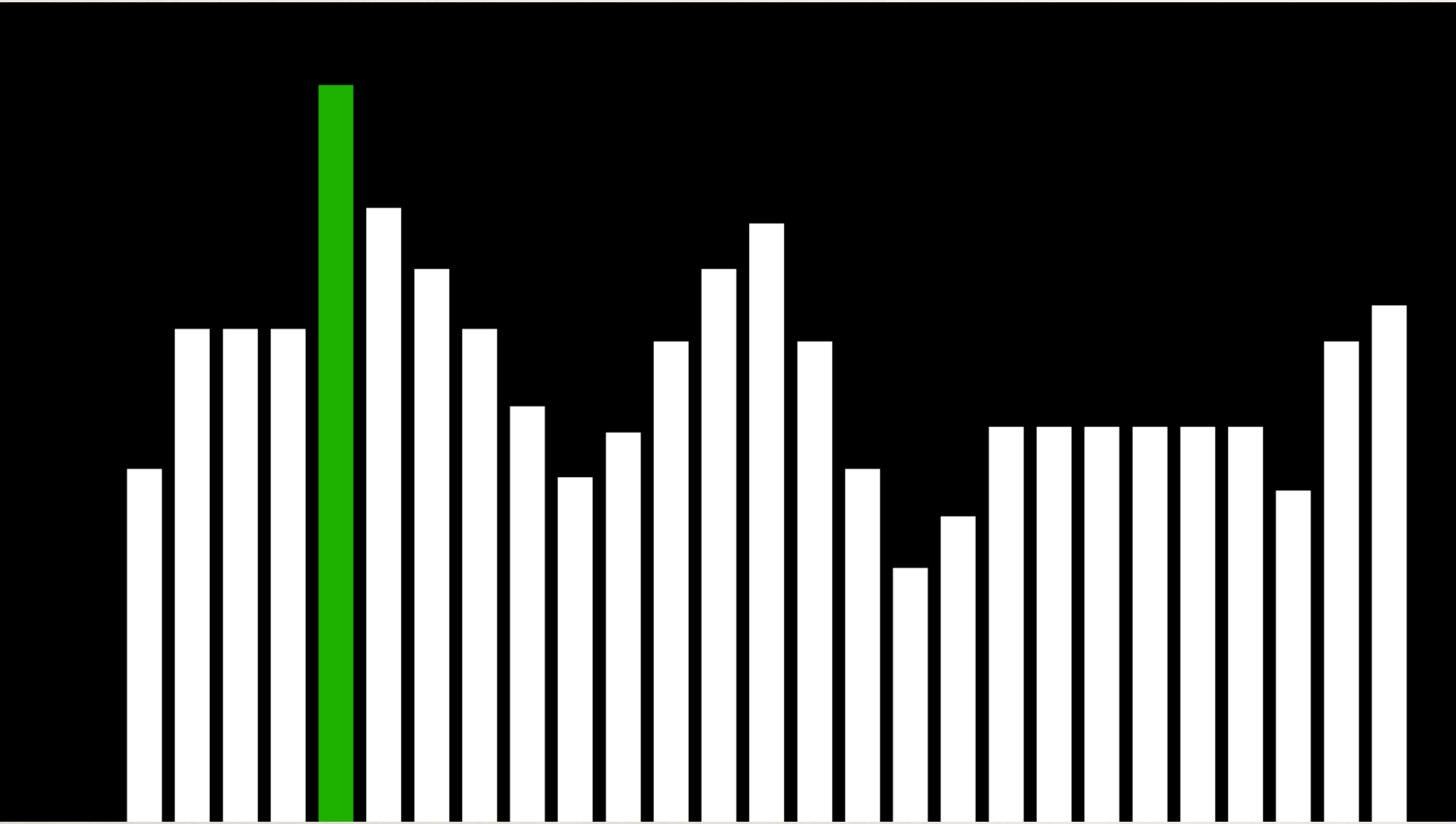
Simulated annealing



Simulated annealing



Simulated annealing



Simulated annealing

```
function SIMULATED-ANNEALING(problem, max):
    current = initial state of problem
    for t = 1 to max:
        T = TEMPERATURE(t)
        neighbor = random neighbor of current
        ΔE = how much better neighbor is than current
        if ΔE > 0:
            current = neighbor
        else
            with probability e^(ΔE/T) set current=neighbor
    return current
```

Temperature analogies

Searching for a New Apartment:

At the start, when looking for a **new apartment**, you might visit many different neighborhoods and consider various layouts (**high temperature**). Over time, you **narrow down** to a few best choices and finally commit to one (**low temperature**).

- **High temperature:** Willing to explore various locations, even ones that seem inconvenient.
- **Low temperature:** Focusing only on the best options and making a final choice.

https://propertyrecordsofmaryland.com/how-to-find-property-owner-information-in-maryland-tips-and-tools/

Temperature analogies

Decision-Making When Shopping

When you first enter a **shopping mall**, you're open to exploring multiple stores and trying out different products (**high temperature, more randomness**). As time passes and you get tired, you focus on finalizing a purchase (**low temperature, less randomness**), settling on the best option you've found.

- **Early stage (high temp):** Exploring many product options, including ones that might not seem great at first.
- **Later stage (low temp):** Narrowing down choices and making a final decision.

https://www.campaignasia.com/article/economic-hurdles-push-japanese-consumers-to-save-not-spend/452840

Local beam search

- The local search algorithms so far kept track of a single state
- **Local beam search** keeps track of *k* states instead
- In each iteration, all the successors of all *k* states are generated
- Unless a stopping condition is met, the best *k* successors are selected for the next iteration
  - Possible stopping condition: no improvement found
- This seems similar to random restart hill-climbing but is actually different: hill-climbing runs are executed *independently*
- Local beam search automatically concentrates the search on those parts of the explored search space where progress is most rapid
- We can increase exploration by adopting **stochastic beam search**
  - Chooses successors with probability proportional to their value

References

- CS50's Introduction to Artificial Intelligence with Python 2020 (https://cs50.harvard.edu/ai/2024/)