Introduction to AI – 236501
Local Search & Optimization
[Chapter 4.1]

Oren Salzman and Sarah Keren

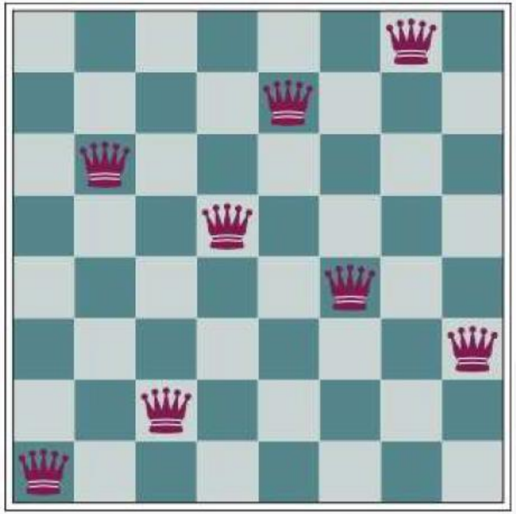
Slides adapted from Shaul Markovitz @ Technion

TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

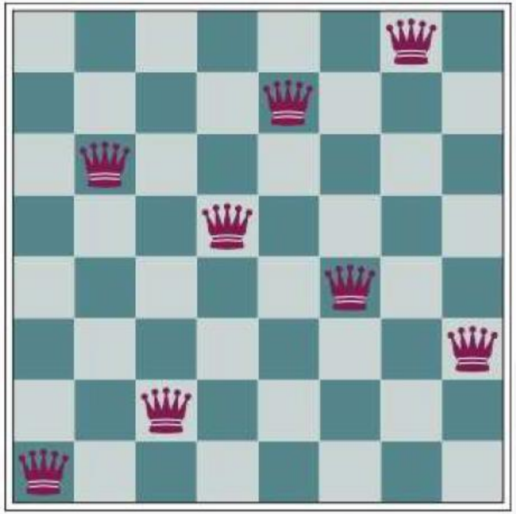
Local search – when the goal is all that matters

- In search problems we were interested in computing a **path** in the search space
- Sometimes we care about only finding a **state** that maximizes some utility function (or minimizes some cost function)
  - 8-queens problem
  - Integrated circuit design
  - Floor layout
  - Telecommunication network optimization
  - …



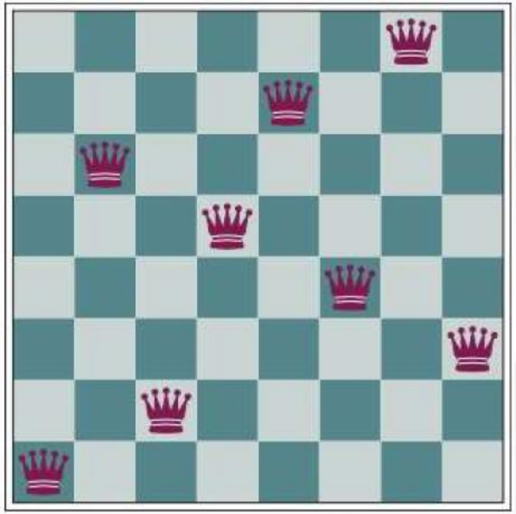
Local search – when the goal is all that matters

- In **local search** we typically store only one or a few states in memory (without the path to reach these states) nor the states that were already reached
  - No systematic coverage of the search space
  - Very low memory footprint
  - Can often find good solutions in reasonable times in huge search spaces
- **Heuristics** here don't measure the distance to the goal but how good a certain state is



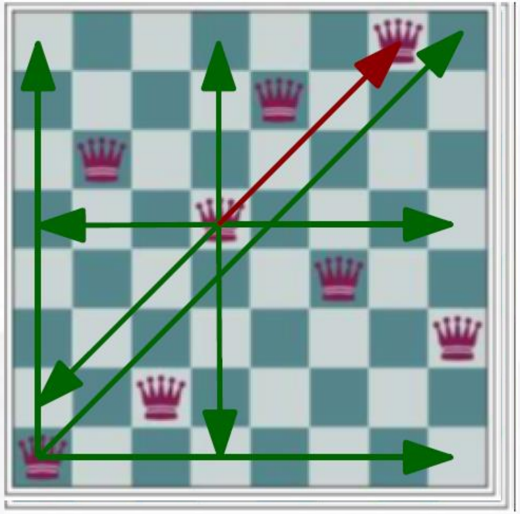
Example 8-Queen problem

- Given **n=8** queens, set them up on a chess board so no queen can attack another



Example 8-Queen problem

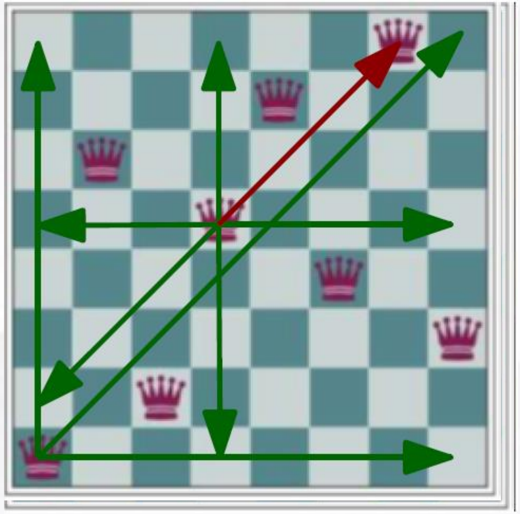
- Given **n=8** queens, set them up on a chess board so no queen can attack another



Example 8-Queen problem

- Given **n=8** queens, set them up on a chess board so no queen can attack another

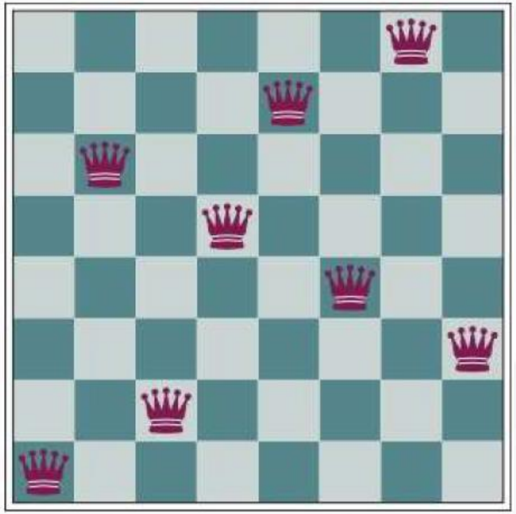
- Didn't we solve this in Intro to CS using recursion?
- Isn't that good enough?
  - Using recursion you can search up to **n!** states
  - OK when **n=8** and **n!** is around **40,000**
  - When **n=20**, **n!** is around **10^18**
  - Local algorithms like the one we will learn can handle in reasonable time instances with **n=3,000,000**



8-Queen problem - representation

- A state will be defined as an **8**-tuple
- The **i**'th element will denote the row of the queen placed in the **i**'th column
- An operator moving from one state to another is a pair (**i**,**j**) stating that the **i**'th queen is moved to the **j**'th row
  - => branching factor is **7X8=56**
- A heuristic can count the number of pairs of queens attacking each another

(8, 3, 7, 4, 2, 5, 1, 6)



8-Queen problem - representation

- A state will be defined as an **8**-tuple
- The **i**'th element will denote the row of the queen placed in the **i**'th column
- An operator moving from one state to another is a pair (**i**,**j**) stating that the **i**'th queen is moved to the **j**'th row
  - => branching factor is **7X8=56**
- A heuristic can count the number of pairs of queens attacking each another

(**h=17: 3+4+2+3+2+2+1+0**)

Hill climbing

TECHNION
Israel Institute of Technology

The Henry and Marilyn Taub
Faculty of Computer Science

Steepest ascent hill-climbing search (SAHC)

- Search starts with a given **start state**
- Every step, **expand** all **neighbors**
- Choose neighbor with **best heuristic value**
- If several exist, choose one **randomly**
- **Terminate** when no neighbor can **improve** current state

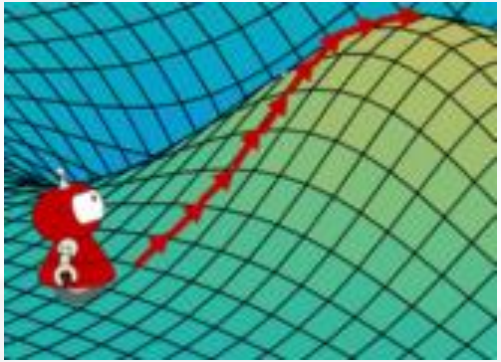
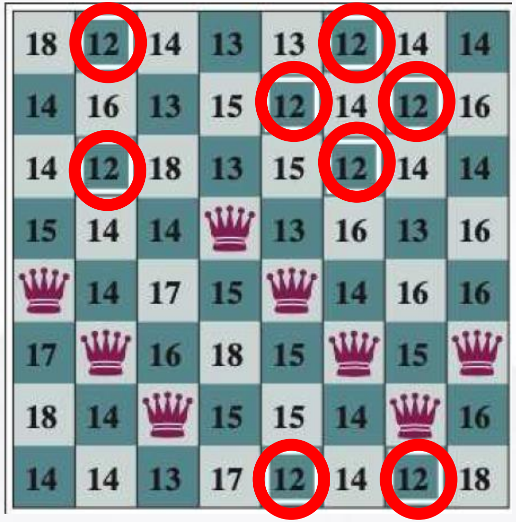


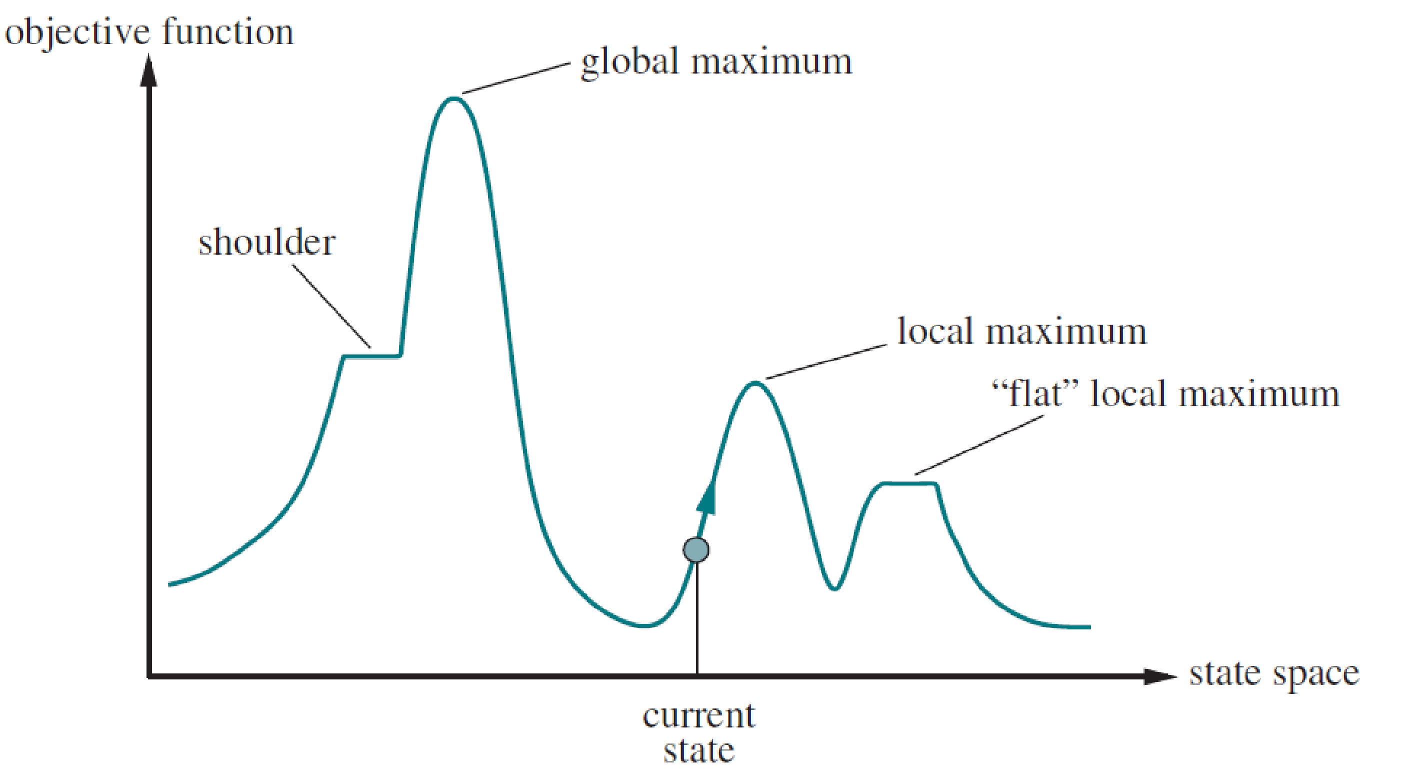
Figure adapted from https://tinyurl.com/2wbyrbta

Steepest ascent hill-climbing search

- In this example, we have
  - Heuristic value of 17
  - Branching factor is **56**
  - **8** neighbors with a value of **12** – one of them will be picked randomly

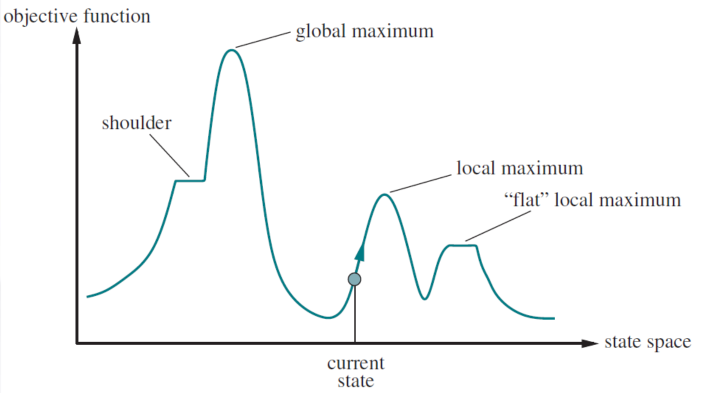


Topography of the objective function



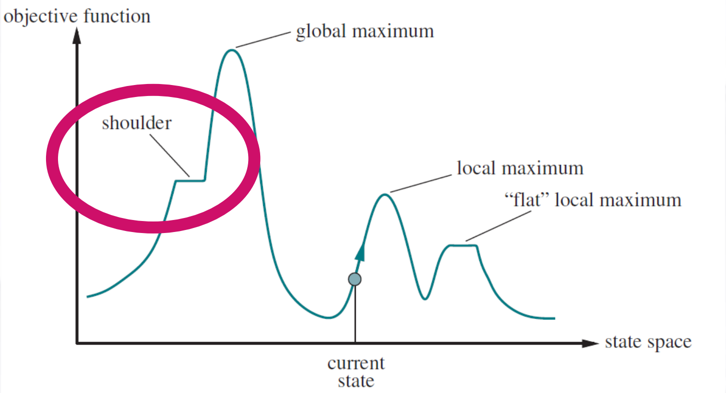
How will SAHC work on the 8-queen puzzle?

- After running the algorithm **1,000** times from random start locations
  - Success rate of **14%** (rest get stuck in a **local minimum**)
  - **4** iterations on average when succeeds
  - **3** iterations on average when fails



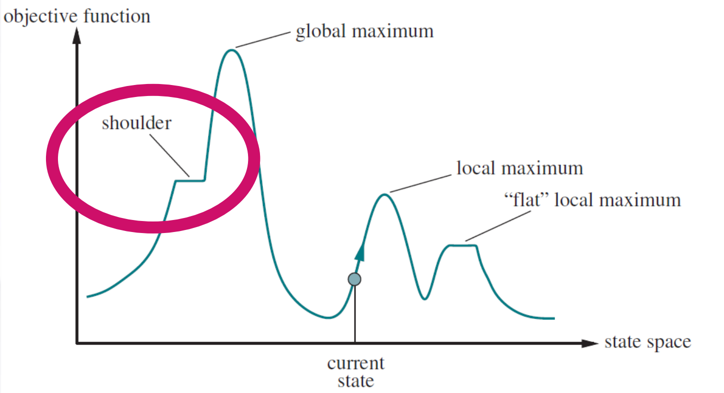
SAHC search with sideway moves

- In many settings, `SAHC` fails because it gets stuck on a shoulder where neighboring states have identical values
- We can overcome this by allowing to transition to a state with the same value
- Typically we add a limit to the number of times this is allowed

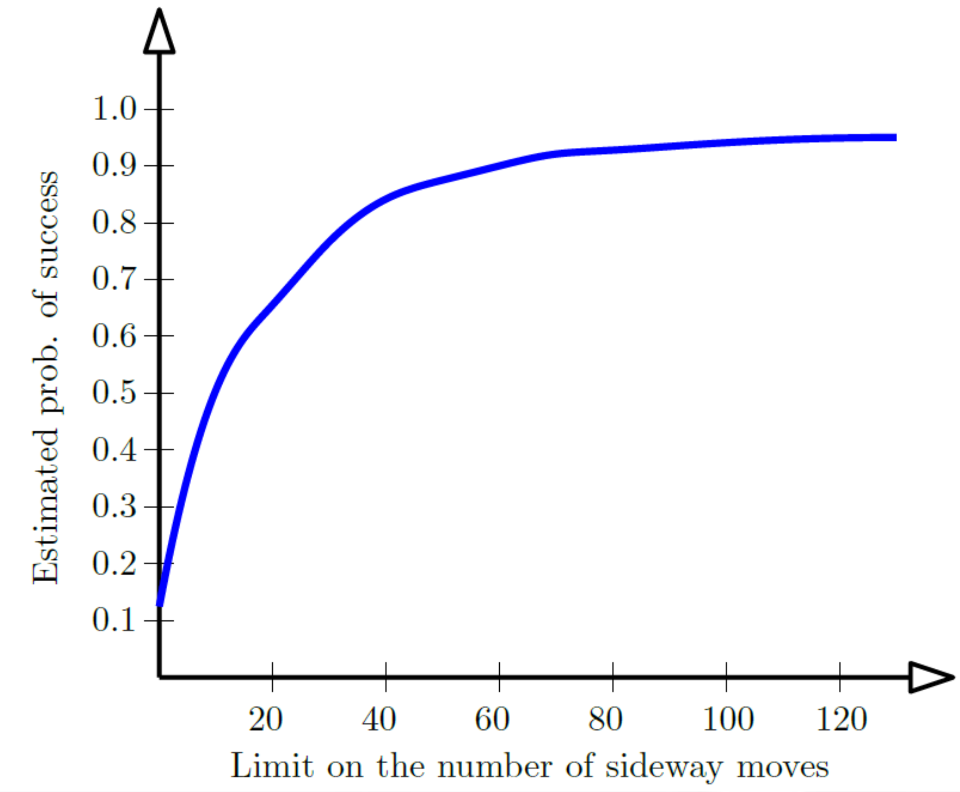


How will SAHC + sideway moves work on the 8-queen puzzle?

- After running the algorithm **1,000** times from random start locations
  - Success rate of **94%** (instead of **14%**)
  - **21** iterations on average when succeeds (instead of **4**)
  - **64** iterations on average when fails (instead of **3**)



How will SAHC + sideway moves work on the 8-queen puzzle?



Stochastic Hill Climbing

- The `SAHC` is a greedy algorithm that chooses at each step the most-promising neighbor
- In Stochastic Hill Climbing we allow other moves as well: choose a neighbor proportionally to the improvement it offers
  - Usually converges slower than SAHC but in some settings finds better solutions (when?)
  - If Δ_i denotes the improvement of moving to neighbor **i**
  - Stochastic Hill Climbing randomly picks neighbor **i** with probability

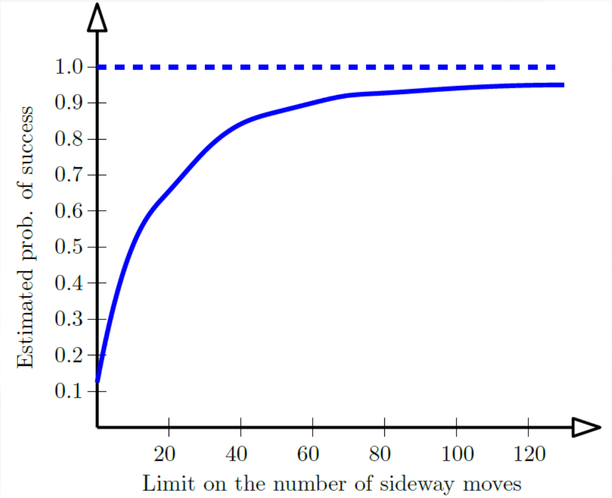
$$P_i = \frac{\Delta_i}{\sum_{i \ s.t. \Delta_i > 0} \Delta_i}$$

First-choice Hill Climbing

- In certain settings the branching factor may be huge (e.g., thousands of successors)
- In **first-choice hill climbing** we randomly generate successors and pick the first one that is better than the current state

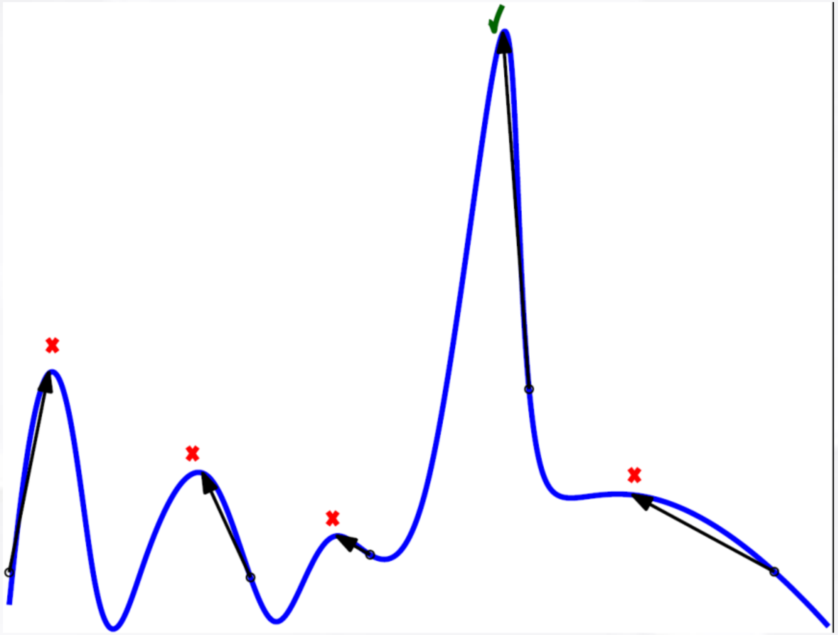
Random Restart Hill Climbing ("…if at first you don't succeed, lift yourself up and try again, try again")

- Conduct a series of hill climbing searches from randomly–generated initial states
- Complete with prob. **1** (why?)



Random Restart Hill Climbing ("…if at first you don't succeed, lift yourself up and try again, try again")

- Conduct a series of hill climbing searches from randomly –generated initial states

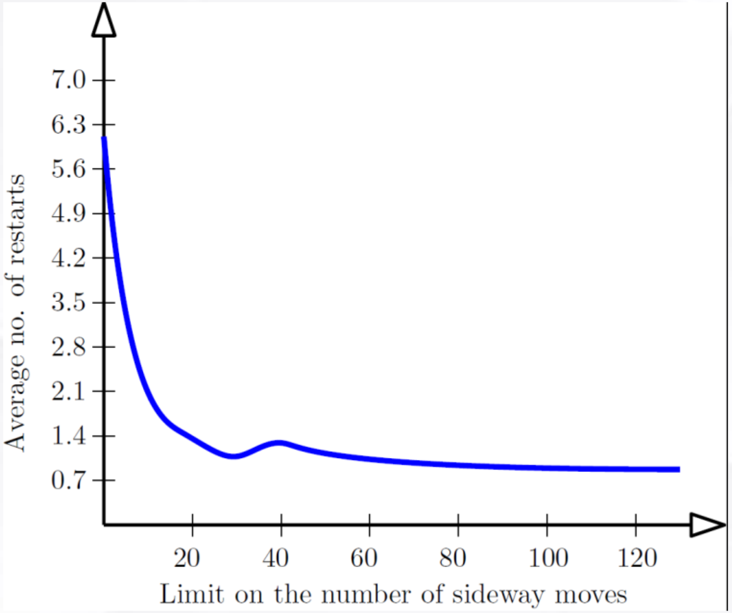


Random Restart Hill Climbing ("…if at first you don't succeed, lift yourself up and try again, try again")

- Conduct a series of hill climbing searches from randomly –generated initial states
- On the **8**-queen puzzle
  - Success rate of **100%** (instead of **14%** / **94%**)
  - **22** iterations on average w.o. sideway moves (instead of **4**)
  - **25** iterations on average w. sideway moves (instead of **21**)

Random Restart Hill Climbing ("...if at first you don't succeed, lift yourself up and try again, try again")

- Conduct a series of hill climbing searches from randomly –generated initial states
- On the 8-queen puzzle
  - Success rate of 100% (instead of 14% / 94%)
  - 22 iterations on average w.o. sideway moves (instead of 4)
  - 25 iterations on average w. sideway moves (instead of 21)

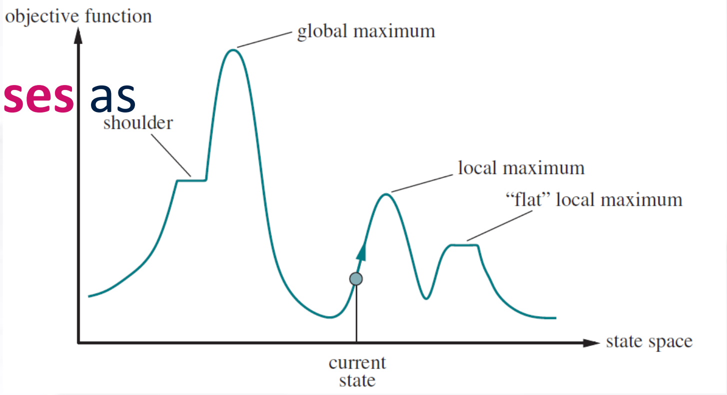


Simulated annealing

Simulated annealing

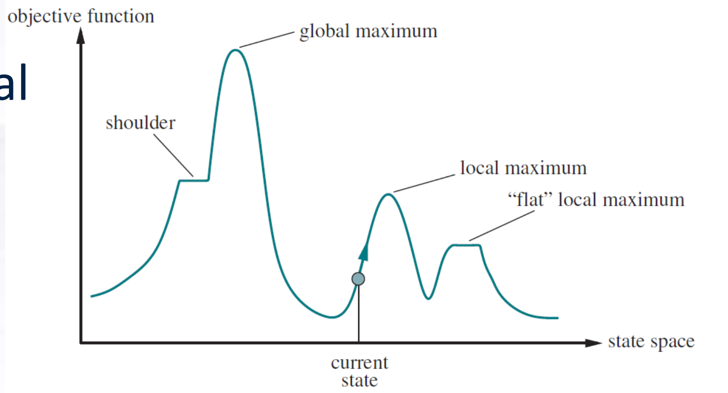
- All hill-climbing algorithms never makes “down-hill moves” towards states with lower value
- Can get stuck in a **local minimum**
- Simulated annealing allows moving to states with a **lower heuristic value**
- The **probability** to choose such a state **decreases** as the search **progresses**

  How?



Simulated annealing

- The **probability** to choose such a state **decreases** as the search **progresses**
- This is done by a parameter called the **temperature**
- The algorithm starts with a high temperature and decreases it gradually
- Multiple approaches on how to choose the initial temperature and how to schedule the decrease

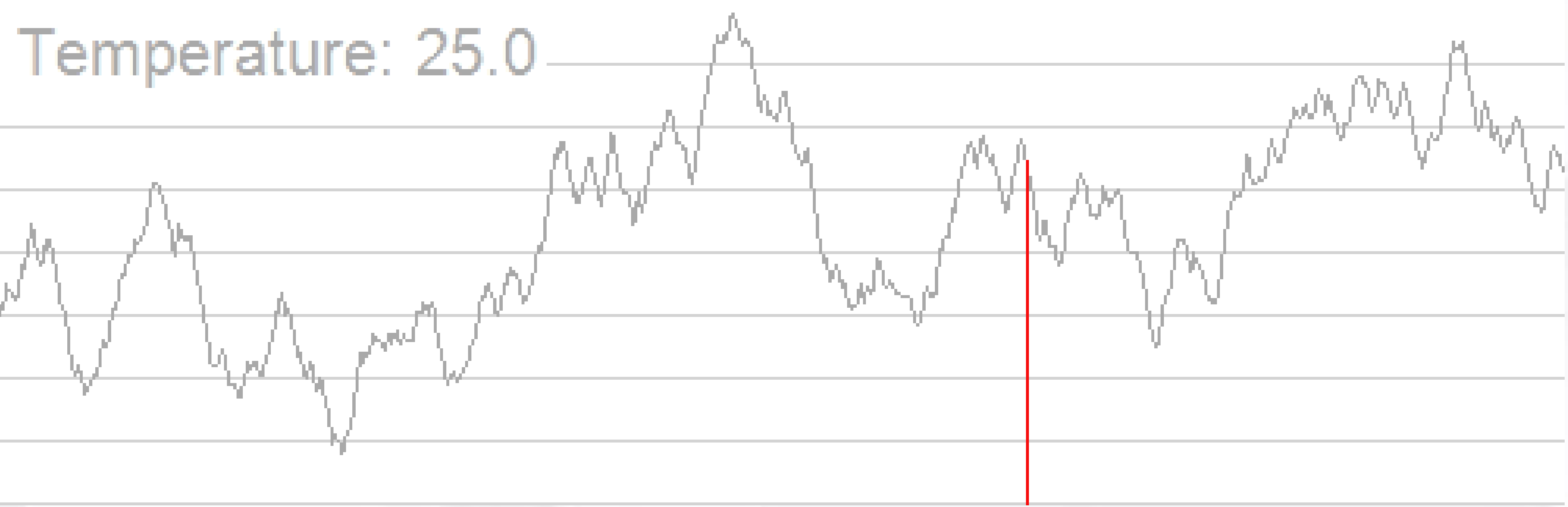


Simulated annealing

`Simulated-annealing` (**state**)

- **curr** <- **state**; **curr-val** <- **h**(**state**); **T**<- T_0
- **loop** until resources exhausted
  - **new** <- random-neighbor(**curr**)
  - **new-val**<-**h**(**new**)
  - **if** goal(**new**) **then return** (**new**)
  - ΔE <- **new-val** – **curr-val**
  - **if** $\Delta E > 0$ **then** **curr** <- **new**; **curr-val** <- **new-val**
  - **else** with prob. $e^{-|\Delta E|/T}$ **curr** <- **new**; **curr-val** <- **new-val**
  - **T**<- **0.95** * **T** `//simple scheduling scheme`
- **return curr**

Simple 1-D cost map (find max)



By Kingpin13 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=25010763

Travelling salesman problem (TSP)

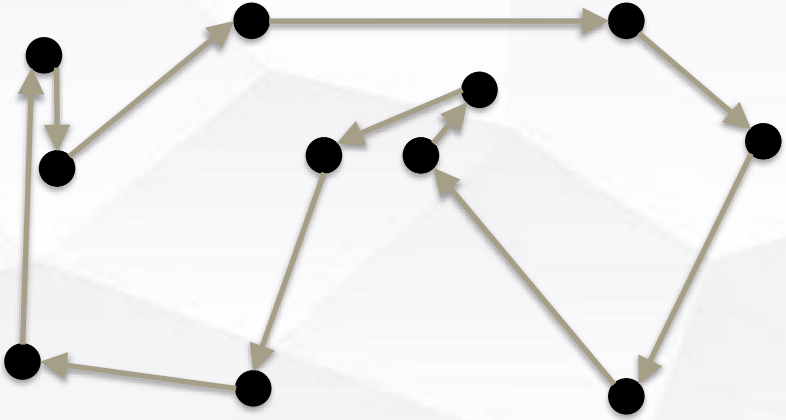
Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- Simplification: cities are points in 2D

Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

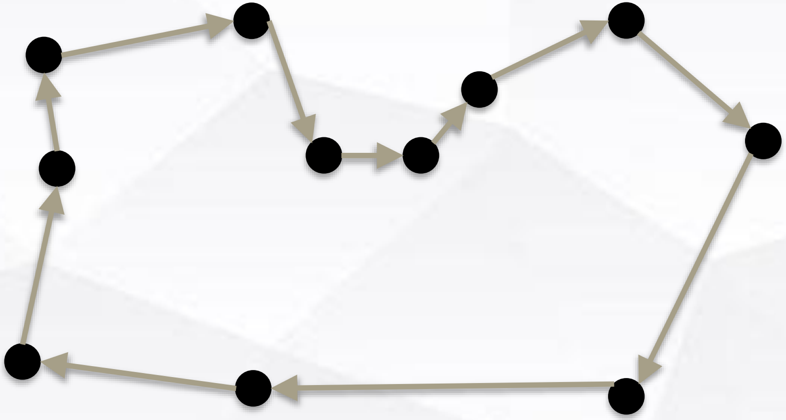
- Hamiltonian cycle in a graph
- Simplification: cities are points in 2D



Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- Simplification: cities are points in 2D



Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- Simplification: cities are points in 2D

Decision variables:

- Where to go next after every city

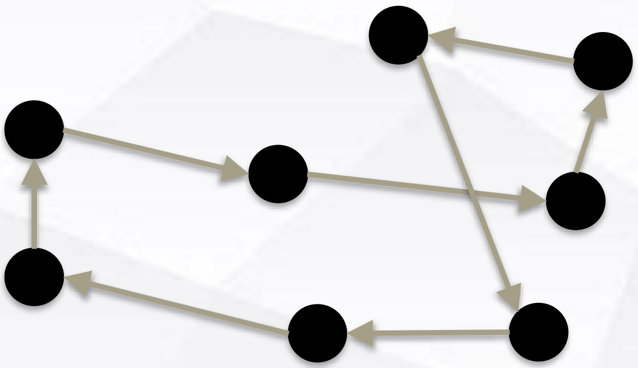
TSP is probably the most well-studied combinatorial problem!

Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- simplification: cities are points in 2D

Local move: select two edges and replace them by two other edges

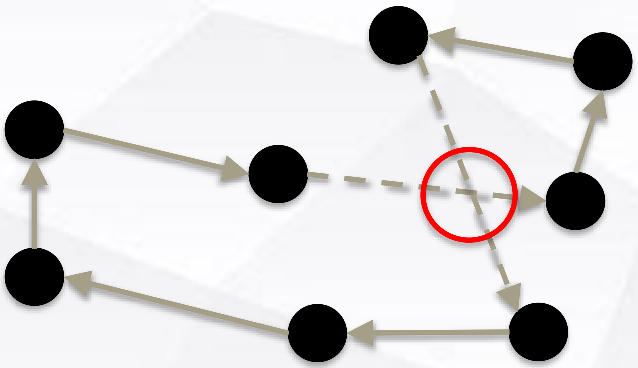


Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- simplification: cities are points in 2D

Local move: select two edges and replace them by two other edges



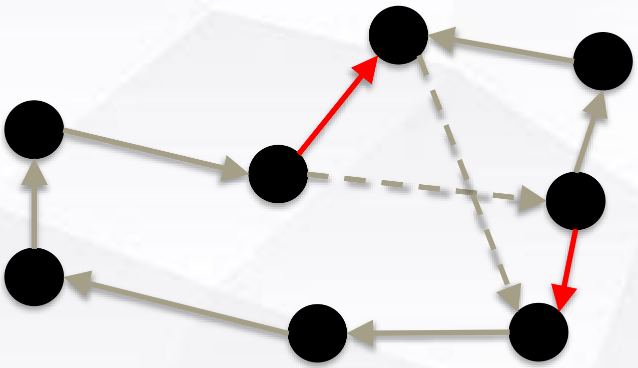
Crossings are bad!

Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- simplification: cities are points in 2D

Local move: select two edges and replace them by two other edges

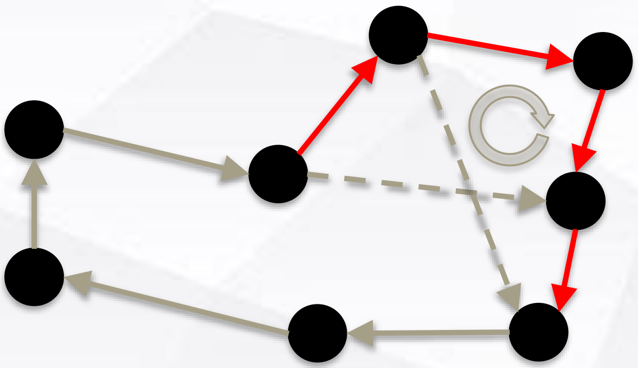


Local search for TSP: 2-OPT

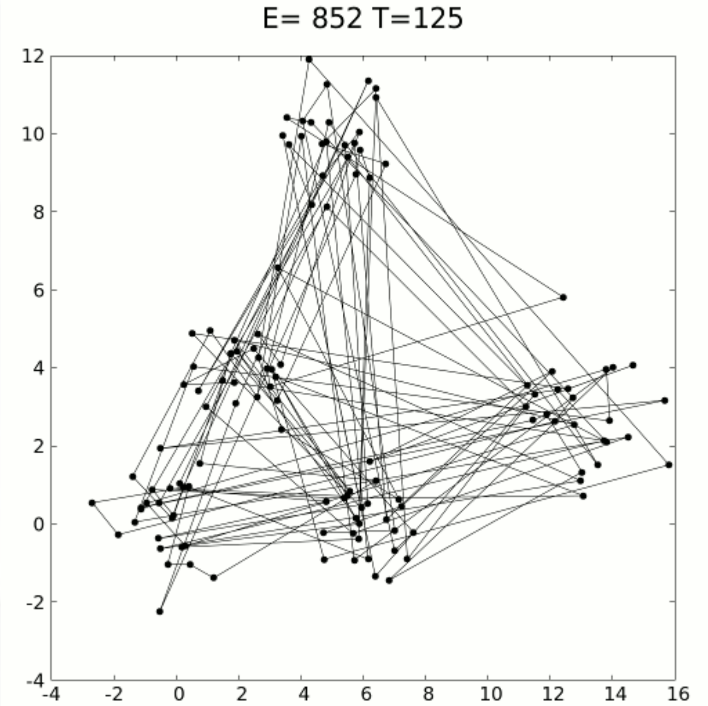
Task: find the shortest path to visit all cities exactly once

- Hamiltonian cycle in a graph
- simplification: cities are points in 2D

Local move: select two edges and replace them by two other edges



Application to TSP



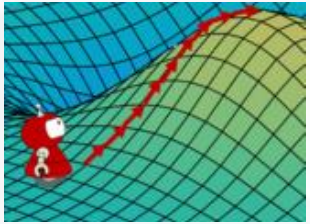
By Geodac - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=67988888

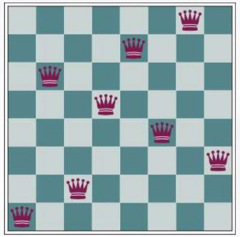
Application to TSP

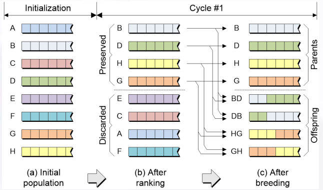
Adapted from https://www.youtube.com/watch?v=NPE3zncXA5s

Local search : conclusion

- We examined several approach for local that start from an initial solution and iteratively improve the solution.
- It is not a systematic search and therefore compromises completeness, but it is very efficient and practical.
- In the appendix you can find information about genetic\ evolutionary algorithms that use biologically inspired ideas such as mutation, crossover and selection to find solutions. They are beyond scope for the course.







Evolutionary algorithms (not part of the course material)

Genetic algorithm

- Local search algorithm that is inspired by **evolution**
- Main idea: take "good parts" from different states and "fuse" them together to obtain new states
- There are **endless** forms of genetic algorithms. We will only give a flavor of the approach

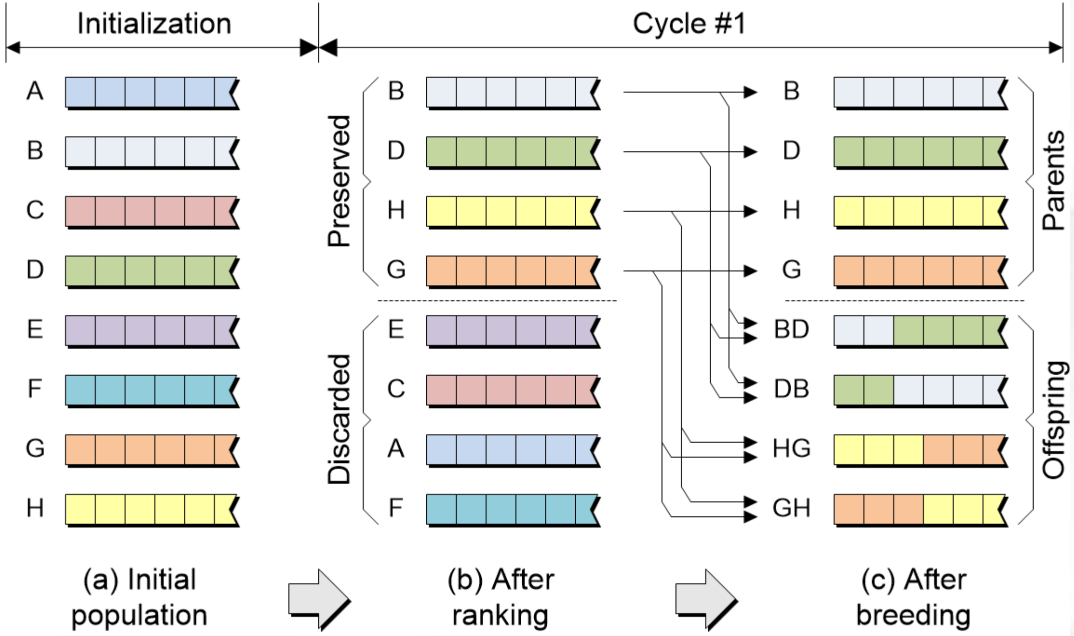


Image source: Max Maxfield

Genetic algorithm (cont.)

- The algorithm stores a set of solutions called the **population**
- Each solution, sometimes referred to as a "**chromosome**" is typically represented as a vector
- Each element in a vector is called a "**gene**"

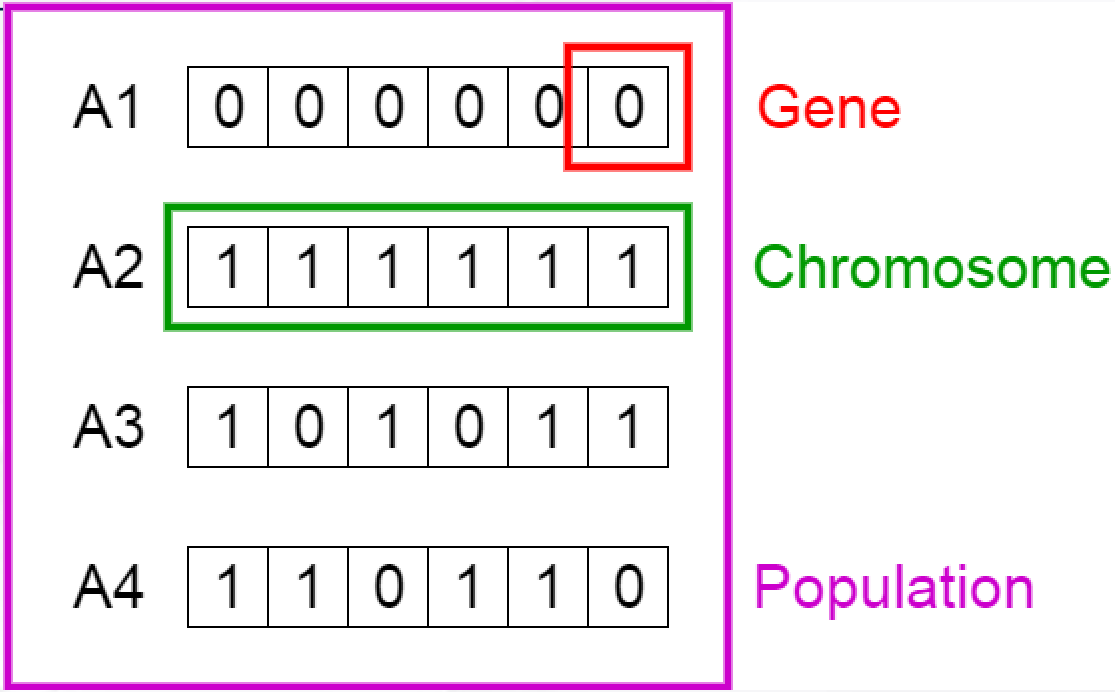


Image source: https://tinyurl.com/2n7nssbh

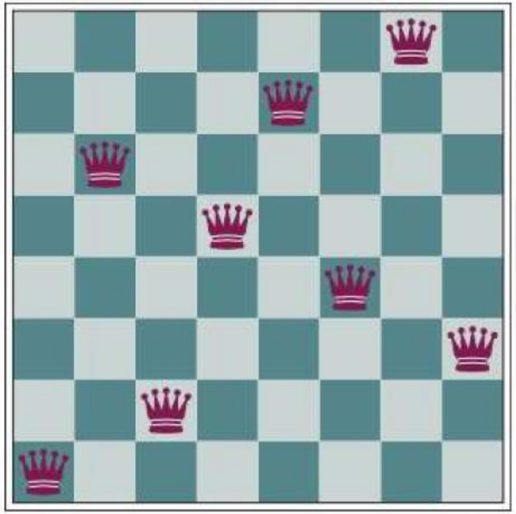
Genetic algorithm (cont.)

- We typically have a fixed **populations size**
- The "**mixing number**" ρ defines how many parent genes are used to create a new child node. Typically is **1** or **2**
- The **selection process** defines how we choose chromosomes that will
  - Move to the next iteration or
  - Serve as "parents" to create "children" for the next generation

  Often implemented by selecting individuals according to their fitness score
- The recombination process (assuming $\rho \geq 2$) typically chooses a random **crossover** point to split each parent string and then combine the two parts of each string
- The "**mutation rate**" defines how often we randomly flip a gene

Genetic algorithm – example (8-queen puzzle)

- A chromosome here will be a vector of 8 genes (digits)
- The **i**'th digit represents the row number of the queen in column **i**
- The fitness is the number of non-dominating pairs of queens



Genetic algorithm – example (8-queen puzzle)

- Assume we start with a **population** size of **4**

24748552

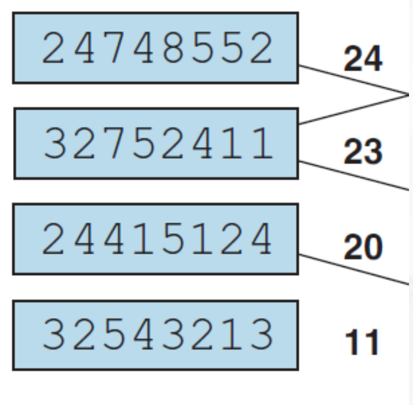
32752411

24415124

32543213

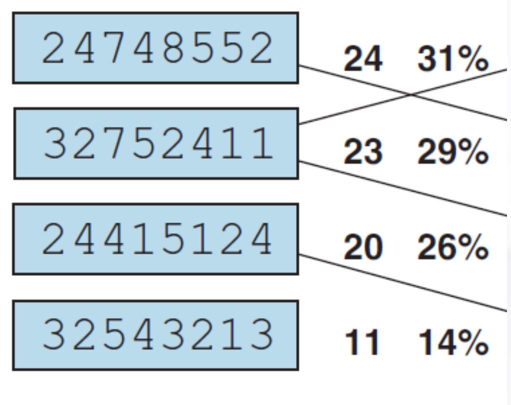
Genetic algorithm – example (8-queen puzzle)

- We compute the **fitness** of each state



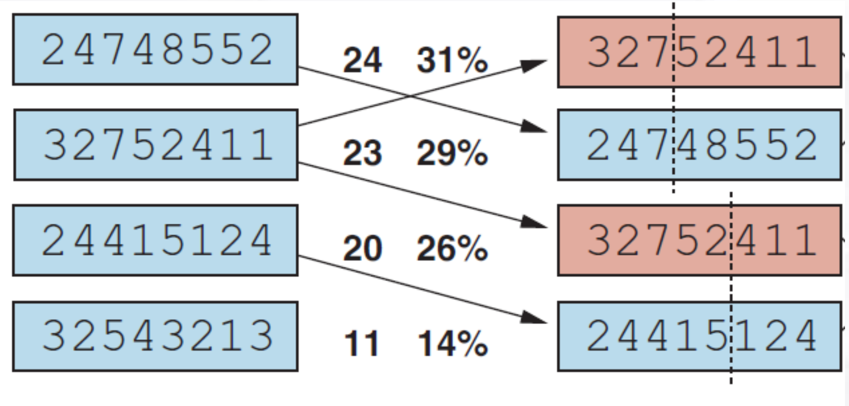
Genetic algorithm – example (8-queen puzzle)

- The **fitness** induces the probability to choose that state for the next step of the process (this is the **selection** process)



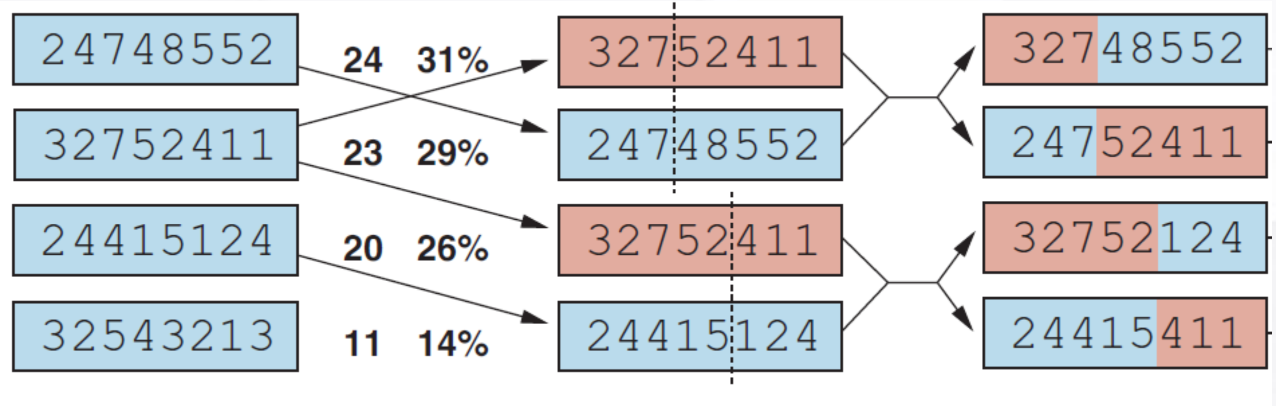
Genetic algorithm – example (8-queen puzzle)

- In this **selection** process, we choose **4** states
- Notice that one state was chosen twice and one was not chosen



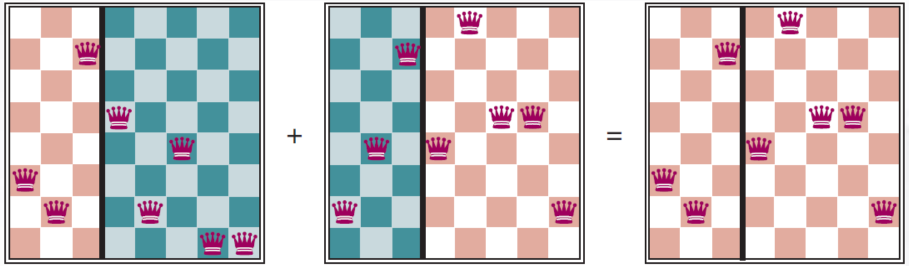
Genetic algorithm – example (8-queen puzzle)

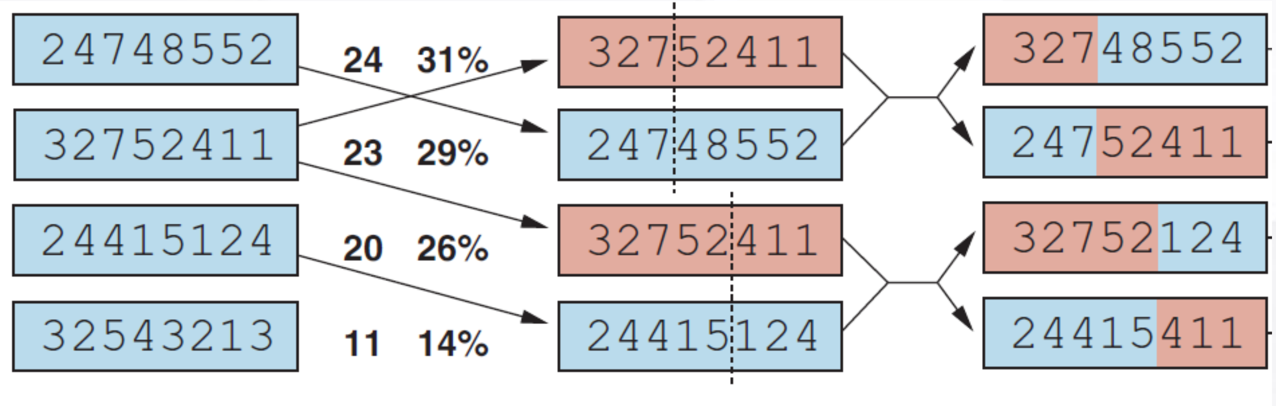
- For each pair of **parents** we randomly choose a **crossover point** and use it to generate a pair of new **child** states



Genetic algorithm – example (8-queen puzzle)

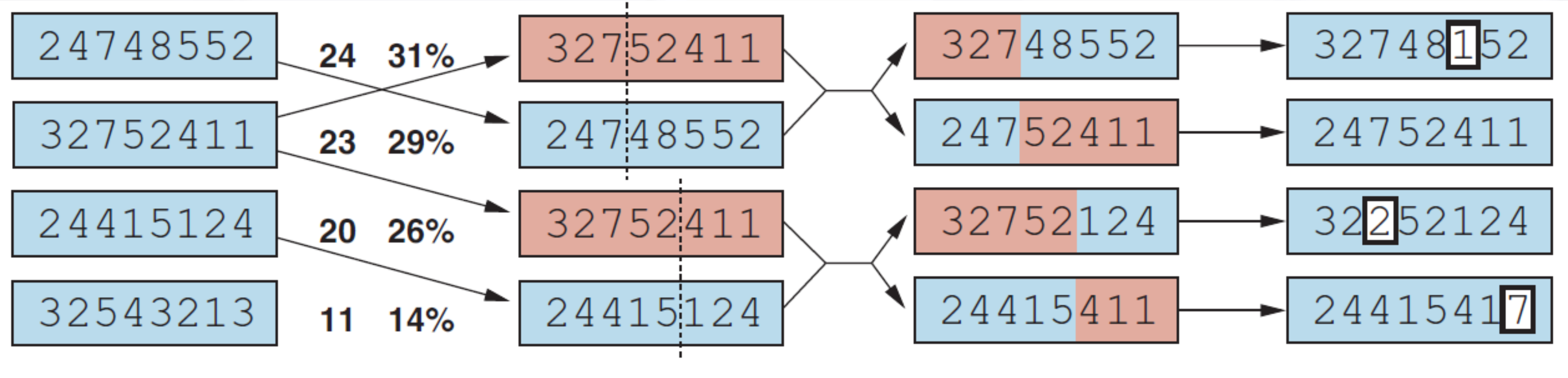
- For each pair of **parents** we randomly choose **a crossover point** and use it to generate a pair of new **child** states





Genetic algorithm – example (8-queen puzzle)

- These new child states are subject to **mutation** with some (small) probability



When to use GA?

- GA attracted a lot of attention in the 1990’s and early 200’s
- Not clear if this is because of the superiority of the method or the appealing metaphor of evolution

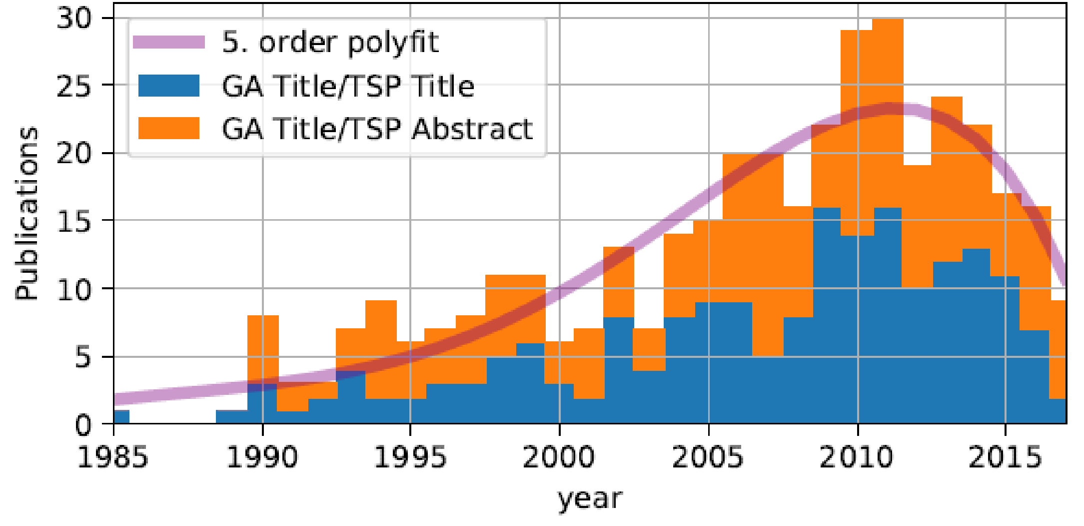


Figure adapted from [Scholz 2018]

When to use GA?

- When we can represent a solution as a vector of values
- Works well when parts of a good solution also exhibit good performance
- Especially works well on variants of the problem of choosing a good subset
  - Given a set of objects **S**
  - Given a function **U** that gives a score to every subset of **S**
  - Find a subset **S’** with maximal **U-**value

When to use GA? Examples

- Given some text **T** with **N** sentences. Find a set of sentences that will summarize **T** in the best way
- Given a query **Q** and **N** search results. Find a subset of **10** search results that maximize the prob. that a user will click on one.
- Given **N** courses a student can take. Find a subset for next semester that maximize some utility (diversity, workload, good exam period etc.)