Tufts
UNIVERSITY

LOCAL SEARCH

Fabrizio Santini | COMP 131

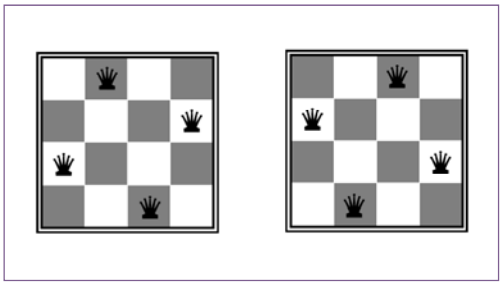
VERSION 1.1

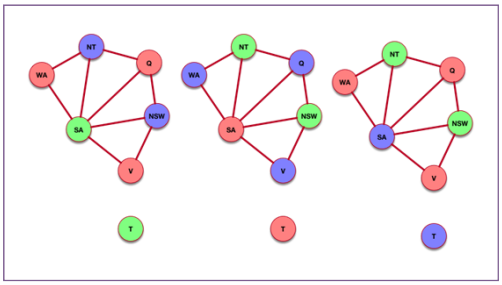
TODAY ON AI

- Hill climbing
- Simulated annealing
- Genetic algorithms
- Questions?

Local search algorithms
WHAT ARE THEY?

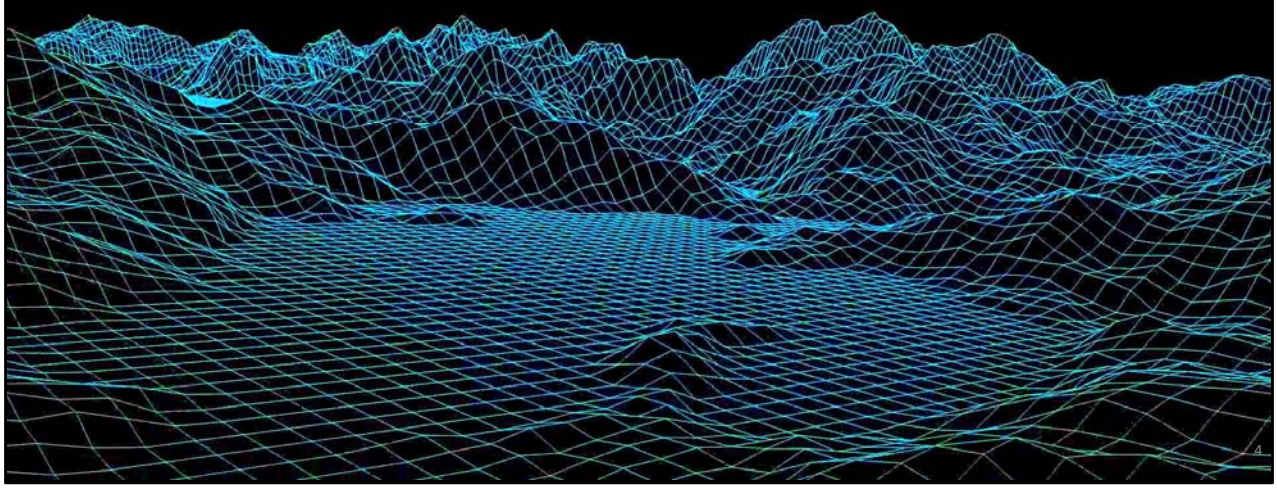
- **Local search algorithms** aim to solve Constraint Satisfaction Problems. Specifically, they are tailored to find a solution to problems whose search space is very big or infinite
- They can always provide an answer to the problem, even if it is both not definitive nor correct





ARTIFICIAL INTELLIGENCE

Local search algorithms work best when the **fitness landscape** is continuous.



Local search algorithms

- This class of algorithms operate on single current nodes that represent the complete state of the search
- The **current state** is the only thing that matter
- The state is evaluated with an **objective function**
- They **generally tend to move** through neighborhoods

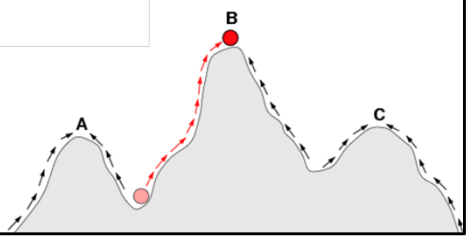
GOOD	Generally much faster for large or infinite state space. More memory efficient
BAD	Incomplete and suboptimal. Not systematic search

Hill climbing algorithms

SECTION 01

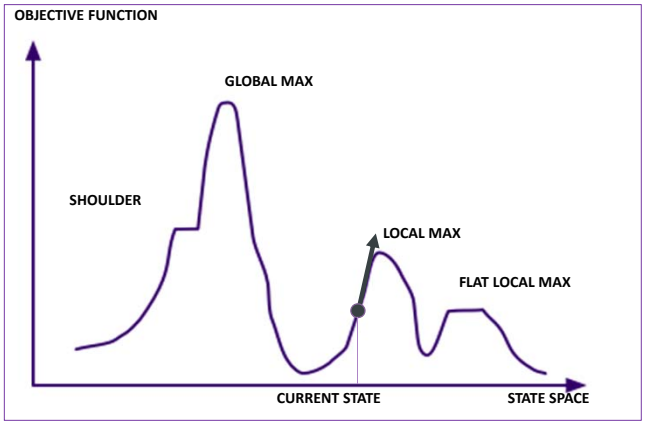
- Hill climbing algorithm is the **most basic local search** technique
- At each step, the current node is **replaced** by the best neighbor
- Hill climbing is **greedy** in nature

```
1  function Hill-climbing(PROBLEM) return SOLUTION, or FAILURE
2
3  current = Make-node(PROBLEM.initial-state)
4
5  loop do
6    neighbor = a highest-valued successor of current
7    if neighbor.value ≤ current.value then
8      return current.state
9    current = neighbor
```



Hill climbing

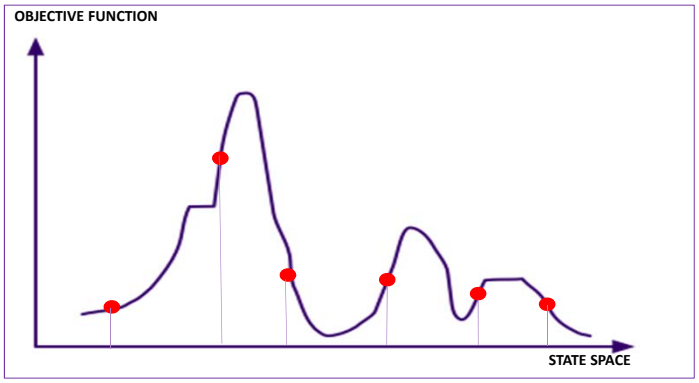
SIMPLEST FAMILY OF ALGORITHMS



GOOD Very simple to implement

BAD It can easily get stack in local maxima, ridges and plateau

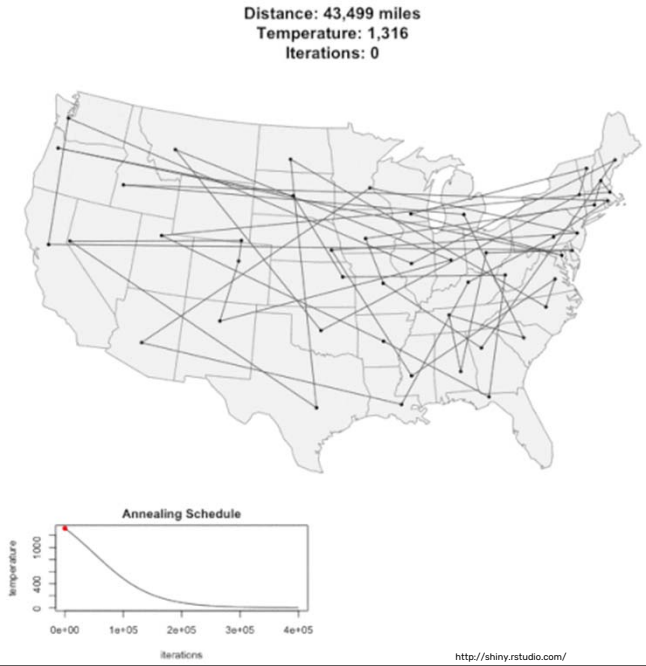
- **Randomly choose** to initialize several times
- Implement hill climbing for **each initialization** and find the optimal
- If each hill climbing search has a **probability p of success**, then the expected number of restarts required is $1/p$.



Simulated annealing

SECTION 02

- **Simulated annealing** is inspired by statistical physics
- **Annealing** is used in metal forging and glass making to aid the formation of crystal structures in the material
- The process **slowly reduces the temperature** the material to allow initial more random arrangements of atoms. At **lower temperatures** the crystallin structure is more stable



The **Traveling Salesman Problem** is a mathematical problem, formulated by W. R. Hamilton in the 1800s, in which one has to find which is the **shortest route** which passes through each of a set of points once and only once.

http://shiny.rstudio.com/

Animation by shiny.rstudio.com

- The basic idea follows the **annealing physical metaphor**: select random successors with decreasing probability, also known as **temperature**.
- A gradient ΔE is calculated:
  - If $\Delta E > 0$ the new state is **accepted immediately** as an improvement
  - If $\Delta E < 0$ the new state is **accepted only with a probability** that depends on ΔE and T
- If T decreases slowly enough, the algorithm will converge

ALGORITHM STRATEGY

```
function Simulated-annealing(PROBLEM, SCHEDULE) return SOLUTION, or FAILURE

current = Make-node(PROBLEM.initial-state)

loop do
  T = SCHEDULE(t)
  if T = 0 then
    return current.state
  next = a randomly selected successor of current
  ΔE = next.value – current.value
  if ΔE > 0 then
    current = next
  else
    current = next only with probability e^(ΔE/T)
```

Animation by Kingpin13, Wikimedia Commons.

Genetic algorithm

SECTION 03

- **Genetic algorithms** are a randomized heuristic search strategy
- They use a **natural selection metaphor** to find the best solution to a problem
- The selection process is applied to a population that is composed of **candidate solutions**
- The purpose is to **evolve a population** from which strong and diverse candidates can emerge via mutation and crossover, also known as mating

- An hypothesis is described by a **chromosome**
- Few successor functions are needed (also known as **fringe functions**):
  - Mutation
  - crossover
- A **fitness function** is used to implement a natural selection process
- A **solution test** is required if different from the fitness function
- Some general parameters guide the evolution of the population:
  - **Population size**
  - **Generation limit**

- Start with a random population
- Apply a **fitness function** to recognize the fittest individuals
- Keep N hypotheses at each step that have a high value of a fitness function
- Apply one or more **successor operations** to generate a new population
- Possibly **cull** the less fit individuals and remove them
- Apply the solution test to the best candidate
- Start over

- **Mutation** fringe operation: given a candidate, return a slightly different candidate
- **Crossover** fringe operation: given two candidates, produce one that has elements of each
- We don't always generate a successor for each individual. Rather, we generate a successor **population** based on the individuals in the current population, weighted by fitness

POPULATION GENERATION

- A new population can be generated by:
  - Given a population P, generate P' by performing crossover $|P|$ times, each time selecting candidates with probability **proportional to their fitness**
  - Get P'' by mutating each individual in P'
  - Return P''
- There are many approaches:
  - The previous approach doesn't explicitly allow individuals to survive more than one generation
  - Crossover is not necessary, though it can be helpful in escaping local maxima
  - Mutation is fundamental

GOOD

- Faster and with lower memory requirements
- It can explore a very large search space
- Easy to design

BAD

- Randomized – not optimal or even complete
- Can get stuck on local maxima, though crossover can help mitigate this
- It can be hard to design a chromosome

EXAMPLE: COLORING THE AUSTRALIA MAP

- **STATES**
  Color the Australia map so that neighboring regions do not match
- **CHROMOSOME**

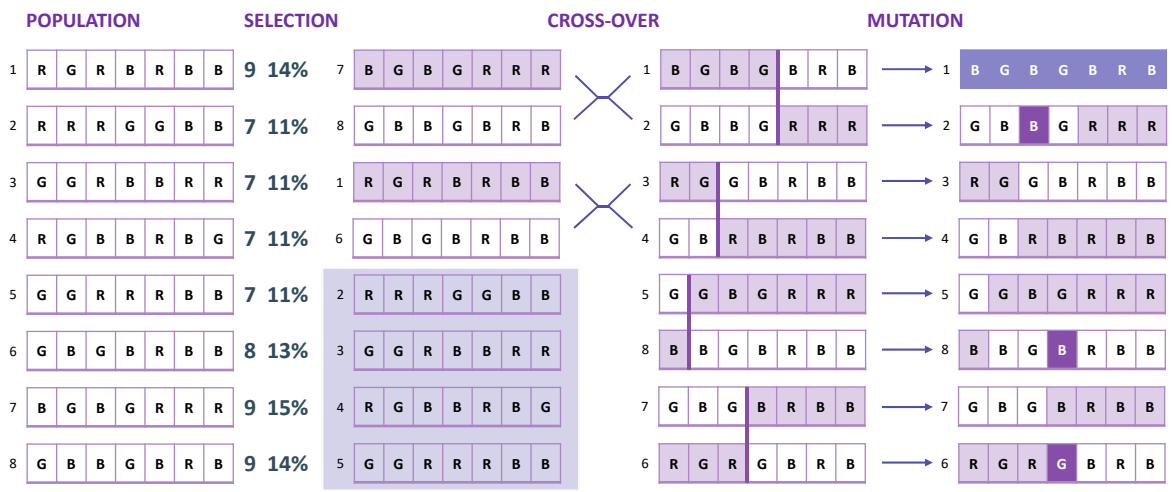
WA	NT	Q	NSW	V	SA	T

- **FITNESS FUNCTION**
  Number of pairs of regions that do not violate the constraint (max value 10)

1	R	G	R	B	R	B	B	9	14%
2	R	R	R	G	G	B	B	7	11%
3	G	G	R	B	B	R	R	7	11%
4	R	G	B	B	R	B	G	7	11%
5	G	G	R	R	R	B	B	7	11%
6	G	B	G	B	R	B	B	8	13%
7	B	G	B	G	R	R	R	9	15%
8	G	B	B	G	B	R	B	9	14%
								63	100%

Genetic algorithms

EXAMPLE: COLORING THE AUSTRALIA MAP



```
function GENETIC-ALGORITHM (population, FITNESS-FN) return an individual
        repeat
                new population ← empty set
                for i = 1 to SIZE(population) do
                        x ← RANDOM-SELECTION(population, FITNESS-FN)
                        y ← RANDOM-SELECTION(population, FITNESS-FN)
                        child ← REPRODUCE(x, y)
                        if (small random probability) then child ← MUTATE(child)
                        add child to new population
                population ← new population
        until some individual is fit enough, or enough time has elapsed
        return the best individual in population, according to FITNESS-FN

function  REPRODUCE(x, y) return an individual
        n ← LENGTH(x)
        c ← random number from 1 to n
        return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

QUESTIONS ?