Artificial Intelligence

Local Search

Genetic Search

More Search Methods

- Local Search
  - Hill Climbing
  - Simulated Annealing
  - Beam Search
  - Genetic Search

Local Search Algorithms and Optimization Problems

- Complete state formulation
  - For example, for the 8 queens problem, all 8 queens are on the board and need to be moved around to get to a goal state
- Equivalent to optimization problems often found in science and engineering
- Start somewhere and try to get to the solution from there
- Local search around the current state to decide where to go next
- They are useful for solving optimization problems, in which the aim is to find the best state according to an objective function

Optimization algorithms

- Optimization algorithm that is not iterative and simply solves for one point
- Optimization algorithm that is iterative in nature and converges to acceptable solution regardless of the parameters initialization such as gradient descent applied to logistic regression
- Optimization algorithm that is iterative in nature and applied to a set of problems that have non-convex cost functions such as neural networks. Therefore, parameters’ initialization plays a critical role in speeding up convergence and achieving lower error rates

Iterative improvement algorithms

- In many optimization problems, the path is irrelevant;

  - the goal state itself is the solution

- Then the state space can be the set of “complete” configuration
  - e.g., for 8-queens, a configuration can be any board with 8 queens
  - e.g., for TSP, a configuration can be any complete tour
- In such cases, we can use iterative improvement algorithms, we keep a single “current” state, and try to improve it
  - e.g., for 8-queens, we gradually move some queen to a better place
  - e.g., for TSP, we start with any tour and gradually improve it

Iterative improvement algorithms

- The goal would be to find an optimal configuration
  - e.g., for 8-queens, an optimal configuration is where no queen is threatened
  - e.g., for TSP, an optimal configuration is the shortest route
- This takes constant space, and is suitable for online as well as offline search

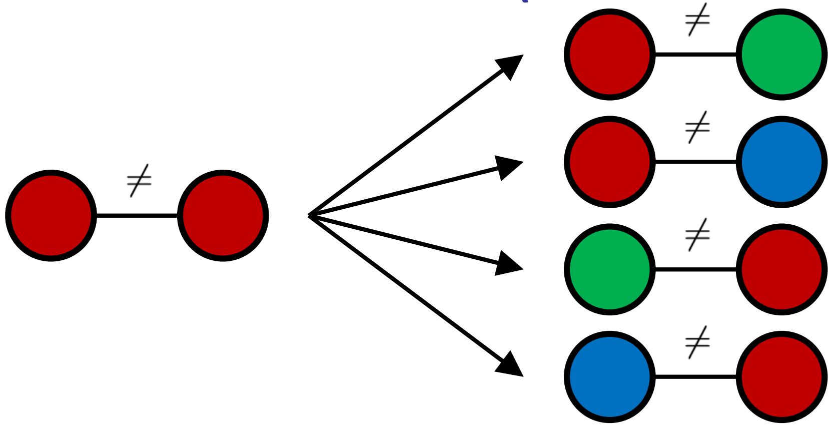
Local Search

Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)

- Local search: improve a single option until you can't make it better (no fringe!)

- New successor function: local changes



- Generally much faster and more memory efficient (but incomplete and suboptimal)

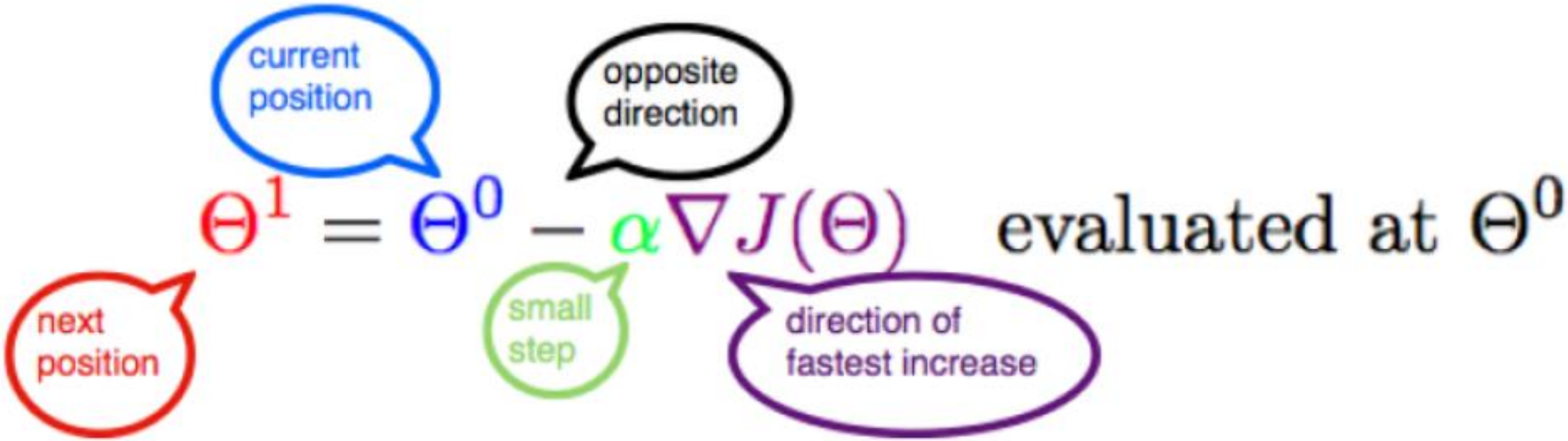
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
- What's bad about this approach?
  - Complete?
  - Optimal?
- What's good about it?

Gradient descent / accent

- Gradient descent / accent is the most common optimization algorithm
- It is a first-order optimization algorithm
  - it only takes into account the first derivative when performing the updates on the parameters.
  - on each iteration, the parameters are updated in the opposite direction of the gradient of the objective function where the gradient gives the direction of the steepest ascent / descent.

Gradient descent in a single line



$$\Theta^1 = \Theta^0 - \alpha \nabla J(\Theta) \quad \text{evaluated at } \Theta^0$$

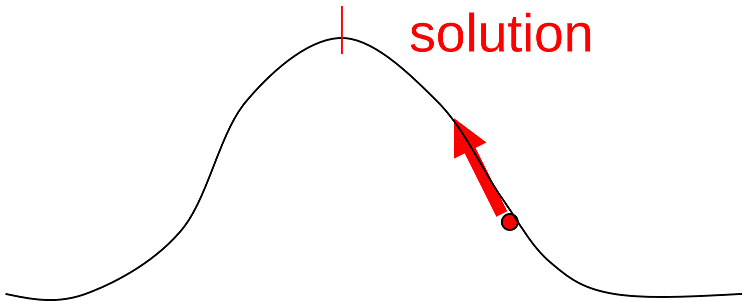
https://www.coursehero.com/file/27927651/Gradient-Descentpdf/

Gradient

- What is the gradient of a function?
- In 1D. Function *f(x)*. Gradient *f′(x)*, the derivative.
  - e.g. $f(x) = x^2$. $f'(x) = ?$
- In 2D. Function *f(x,y)*. Gradient $(\partial x, \partial y)$

Hill Climbing

“Gradient ascent”



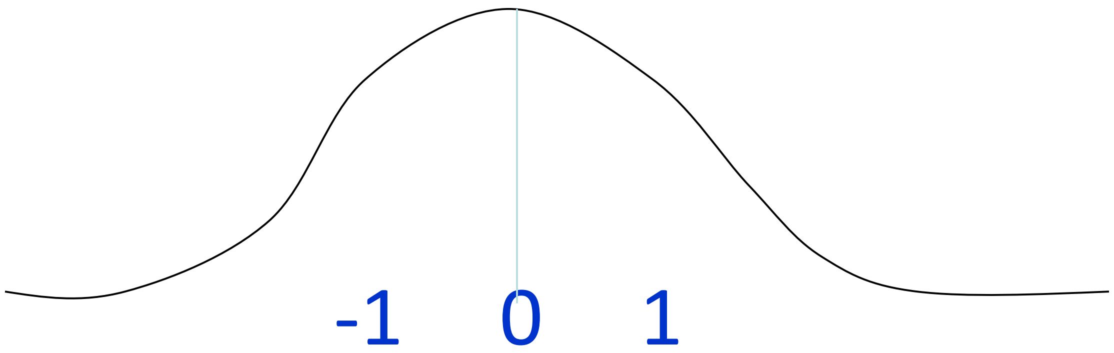
Note: solutions shown here as max not min.

Often used for numerical optimization problems.

How does it work?

In continuous space, the gradient tells you the direction in which to move uphill.

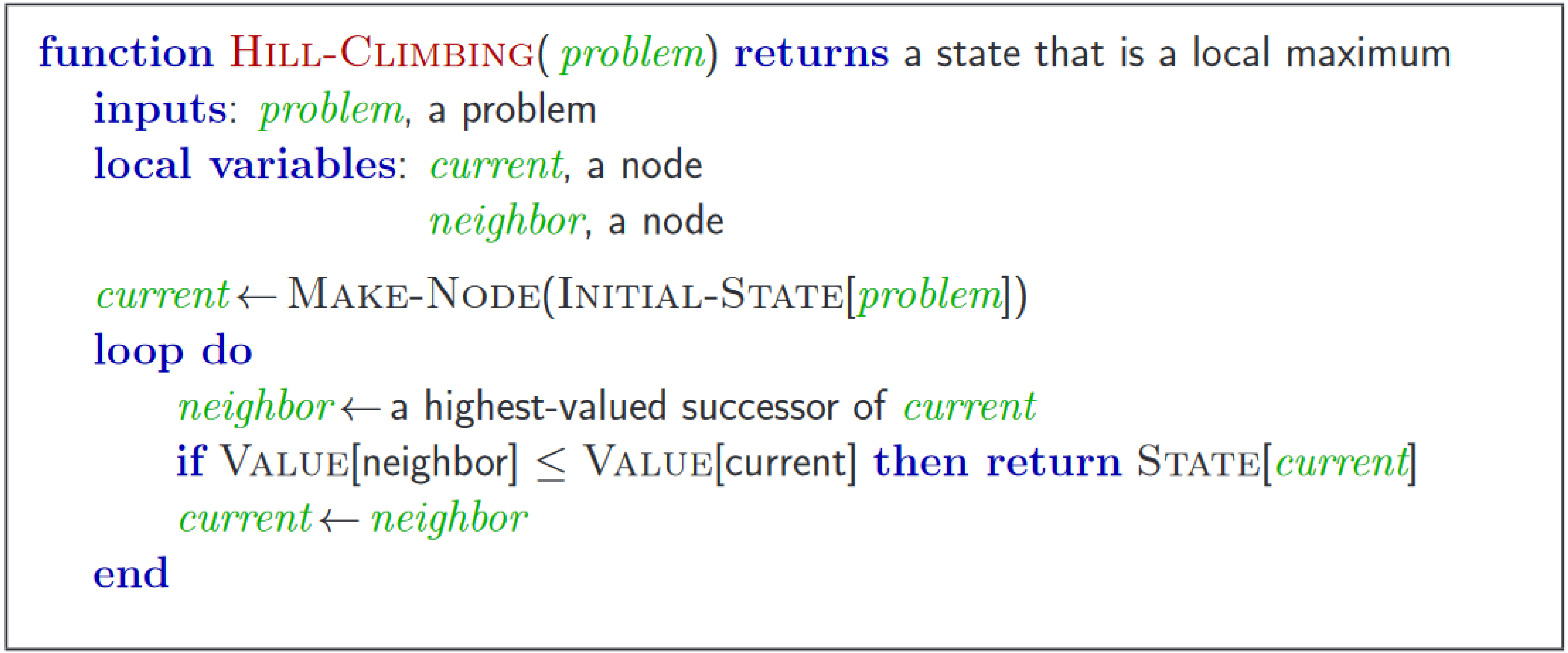
Numeric Example



- Normal distribution with 0 mean and 1 SD
- *f(x)* = c e ^ (-1/2)x^2
- *f′(x)* = -x c e ^ (-1/2)x^2
- *f′(1)* comes out negative, i.e. move backward
- *f′(-1)* comes out positive, i.e. move forward.

Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in thick fog with amnesia"



```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

AI Hill Climbing

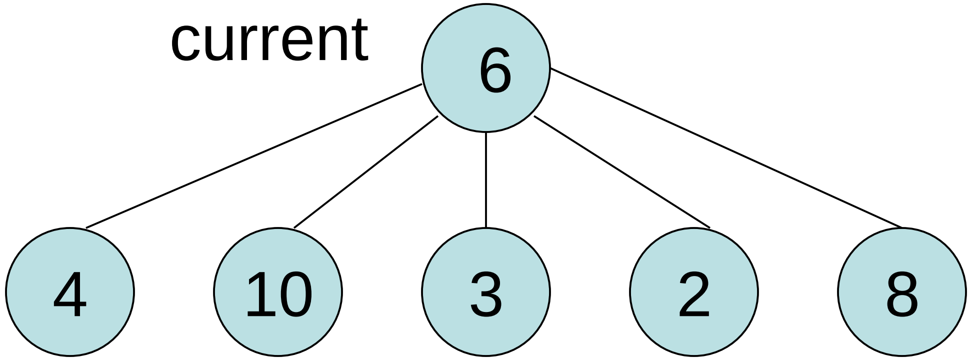
Steepest-Ascent Hill Climbing

```
current ← start node
loop do
    neighbor ← a highest-valued successor of current
    if neighbor.Value <= current.Value then return current.State
    current ← neighbor
end loop
```

At each step, the current node is replaced by the best (highest-valued) neighbor.

This is sometimes called greedy local search.

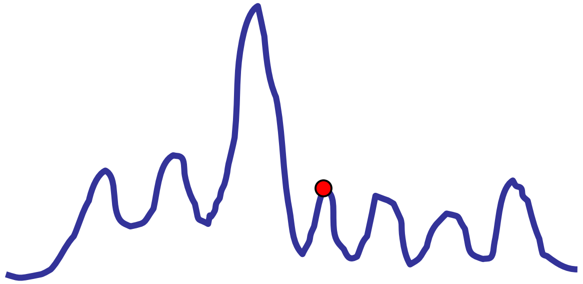
Hill Climbing Search



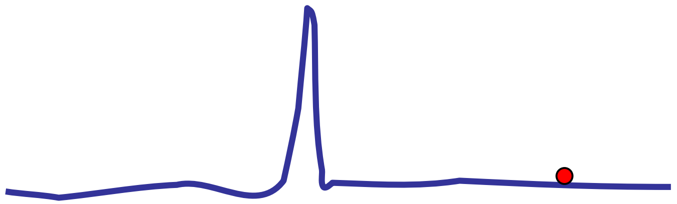
What if current had a value of 12?

Hill Climbing Problems

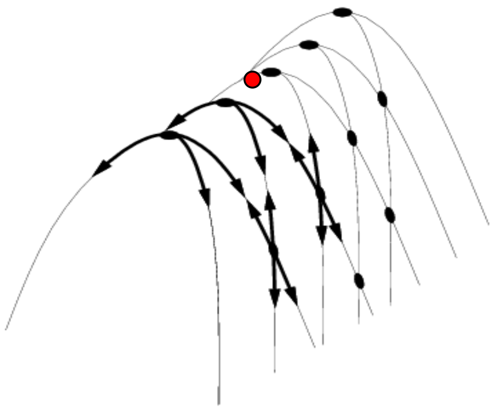
Local maxima



Plateaus

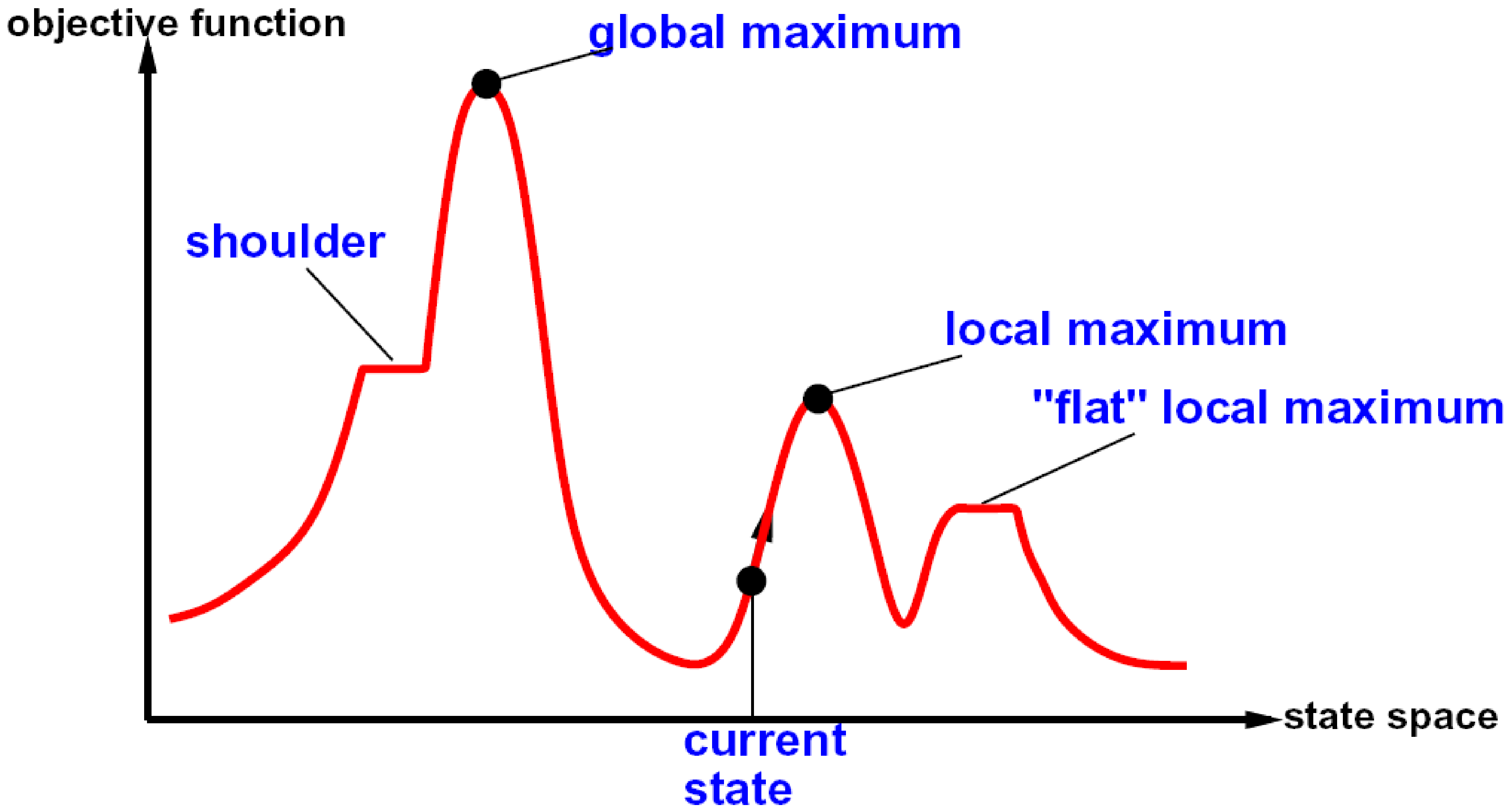


Diagonal ridges



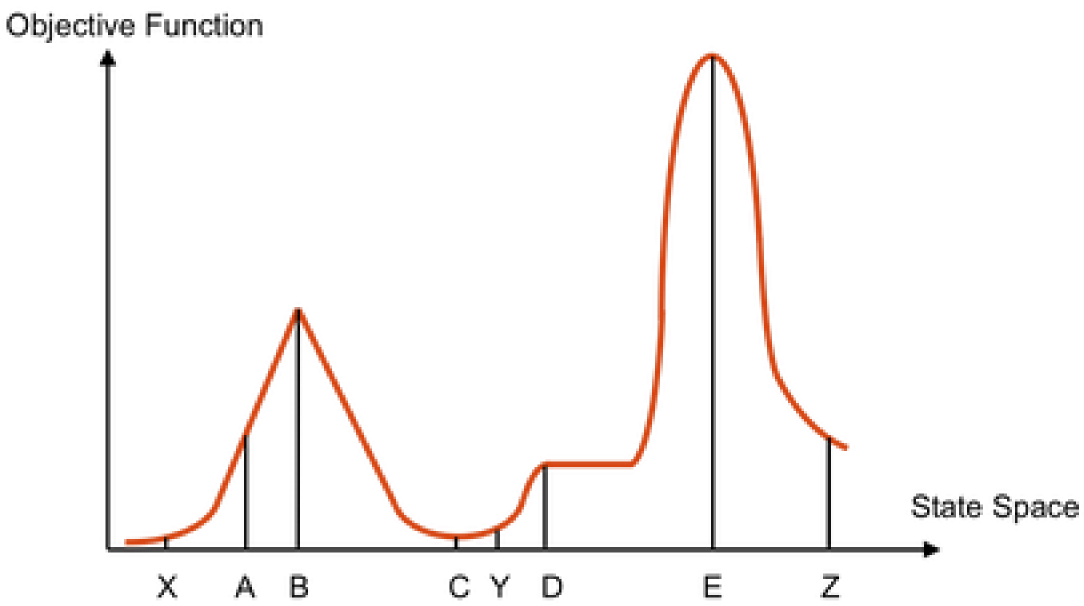
What is it sensitive to?
Does it have any advantages?

Hill Climbing Diagram



- Random-restart hill climbing overcomes local maxima—trivially complete
- Random sideways moves escape from shoulders loop on flat maxima

Hill Climbing Quiz



Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Solving the Problems

- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part of the search space

Simulated Annealing

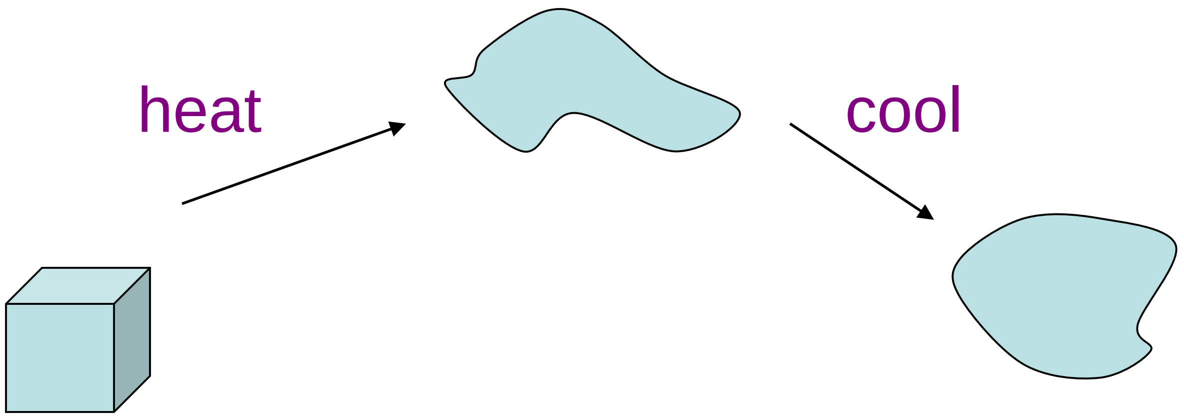
- Variant of hill climbing (so up is good)

- Tries to explore enough of the search space early on, so that the final solution is less sensitive to the start state.

- May make some downhill moves before finding a good way to move uphill.

Simulated Annealing

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is

  p = e^(-ΔE/kT)

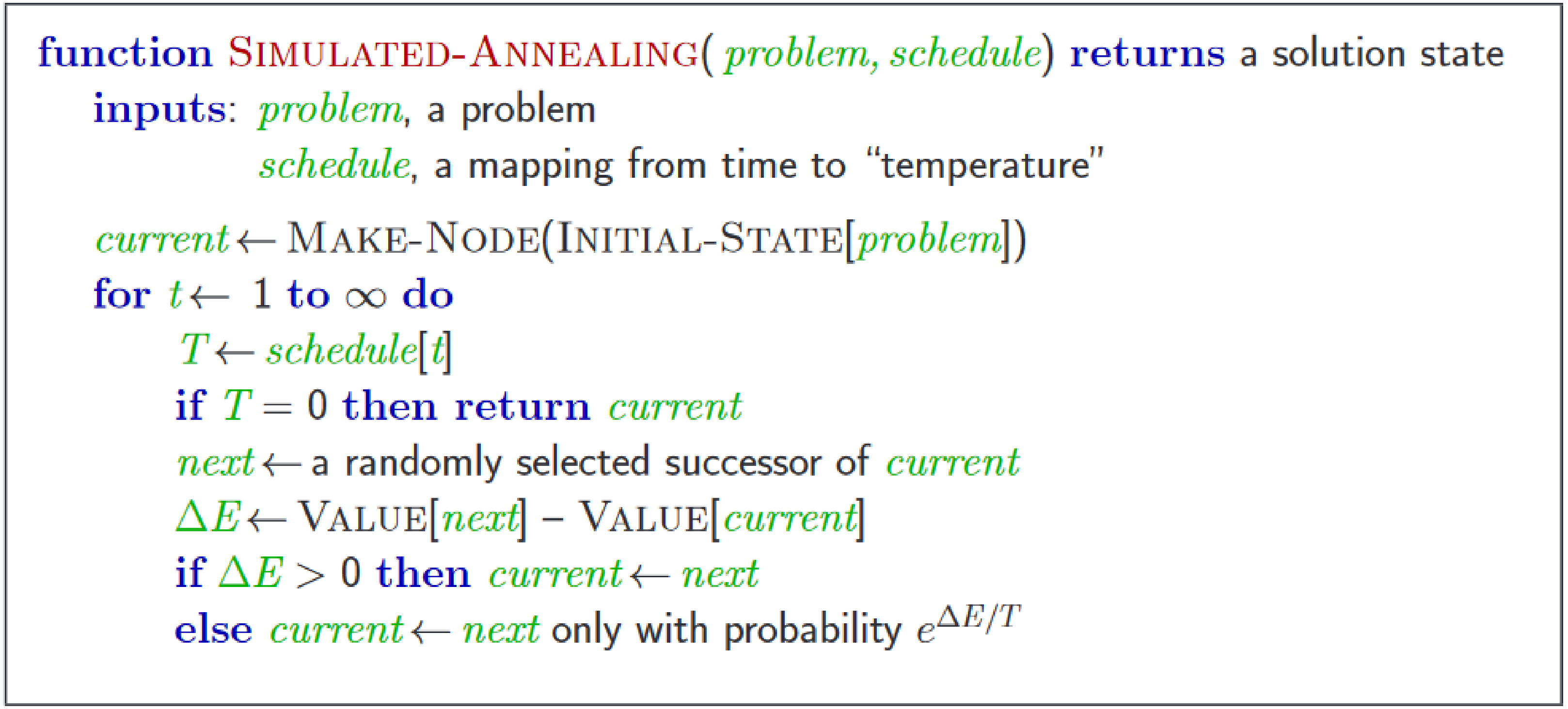
  where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann’s constant.

Simulated Annealing

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - ΔE/kT gets bigger
  - (-ΔE/kT) gets smaller
  - e^(-ΔE/kT) gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

Simulated annealing

Idea: Escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency



```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       $\Delta E$ ← VALUE[next] – VALUE[current]
       if $\Delta E > 0$ then current ← next
       else current ← next only with probability $e^{\Delta E/T}$
```

Note: The *schedule* should **decrease** the temperature T
so that it gradually goes to 0

Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling

- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

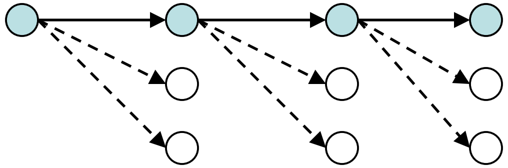
Local Beam Search

- Keeps more previous states in memory
  - Simulated annealing just kept one previous state in memory.
  - This search keeps k states in memory.

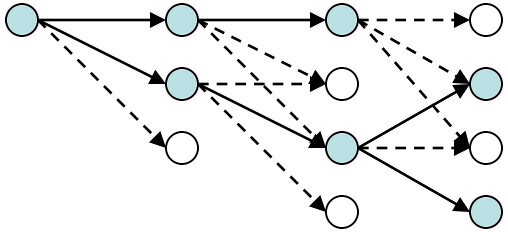
    - randomly generate k initial states
    - if any state is a goal, terminate
    - else, generate all successors and select best k
    - repeat

Beam Search

- Like greedy hill climbing search, but keep K states at all times:



Greedy Search



Beam Search

- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- Complete? Optimal?
- Why do we still need optimal methods?

Next: *Genetic Algorithms, which are motivated by human genetics. How do you search a very large search space in a fitness oriented way?*

Natural Selection

- The origin of species: “Preservation of favourable variations and rejection of unfavourable variations.”
- There are more individuals born than can survive, so there is a continuous struggle for life.
- Individuals with an advantage have a greater chance for survive: so survival of the fittest.

Why evolutionary (biology-inspired) algorithms?

- In reality there are many populations that live in conditions with scarce resources
- The competition for the scarce resources is the reason for selection of those individuals that are fitter than the rest to adapt to the environment
- The selected individuals serve as seeds that generate new individuals by crossbreeding and mutation
- New individuals prove their fitness to adapt to the environment and compete with others to survive
- In time, natural selection results in a population which is fitter to the conditions and the environments

Genetic algorithms

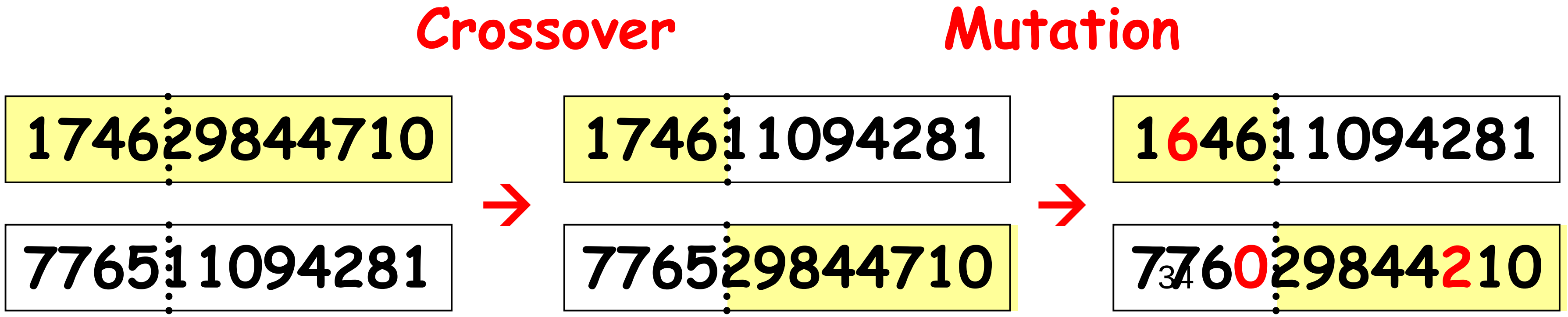
- GA are evolutionary algorithms (inspired from biology)
    - inheritance
    - selection
    - crossover / recombination
    - mutation
- Applied in: computer science, engineering, economy, chemistry, physics, mathematics, ...

Genetic Algorithms

- The *genetic algorithm* is a **probabilistic search algorithm** that iteratively **transforms a set** (called a *population*) of mathematical **objects** (typically fixed-length binary character strings), each **with an associated fitness value,** into **a new population of offspring objects** using the Darwinian principle of natural selection and **using operations that are patterned after naturally occurring genetic operations**, such as crossover (sexual recombination) and mutation
- Technique for finding the exact or approximate solution in optimization problems or search problems using a global search heuristics

Genetic Algorithms

- Start with random population of states
  - Representation serialized (ie. strings of characters or bits)
  - States are ranked with “fitness function”
- Produce new generation
  - Select random pair(s) using probability:
    - probability ~ fitness
  - Randomly choose “crossover point”
    - Offspring mix halves
  - Randomly mutate bits



Crossover

Mutation

174629844710

776511094281

→

174611094281

776529844710

→

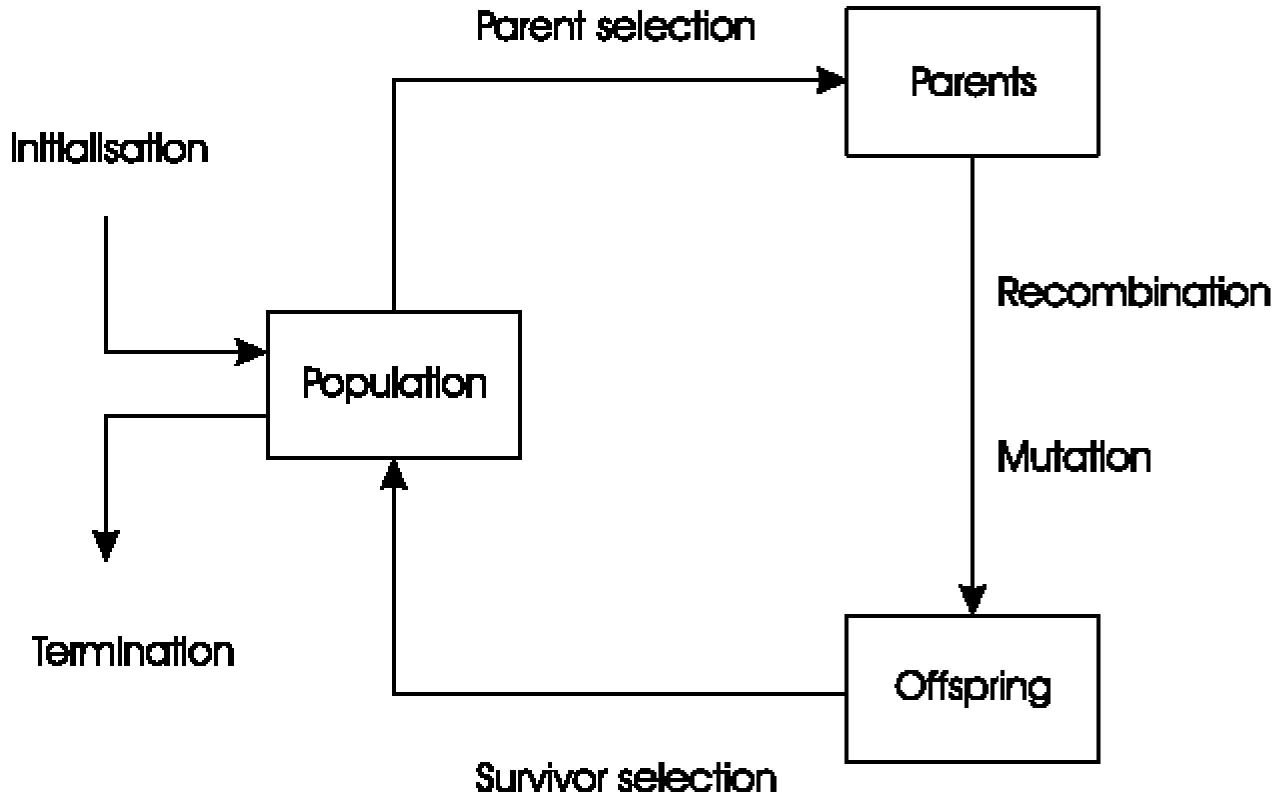
164611094281

776029844210

Genetic Algorithm Pseudo Code

```
function GENETIC-ALGORITHM(population, FITNESS-FN)
    returns an individual
    inputs: population, a set of individuals
            FITNESS-FN, a function that measures the fitness of an individual
    repeat
            parents ← SELECTION(population, FITNESS-FN)
            population ← REPRODUCTION(parents)
    until some individual is fit enough
    return the best individual in population, according to FITNESS-FN
```

General framework of genetic algorithms



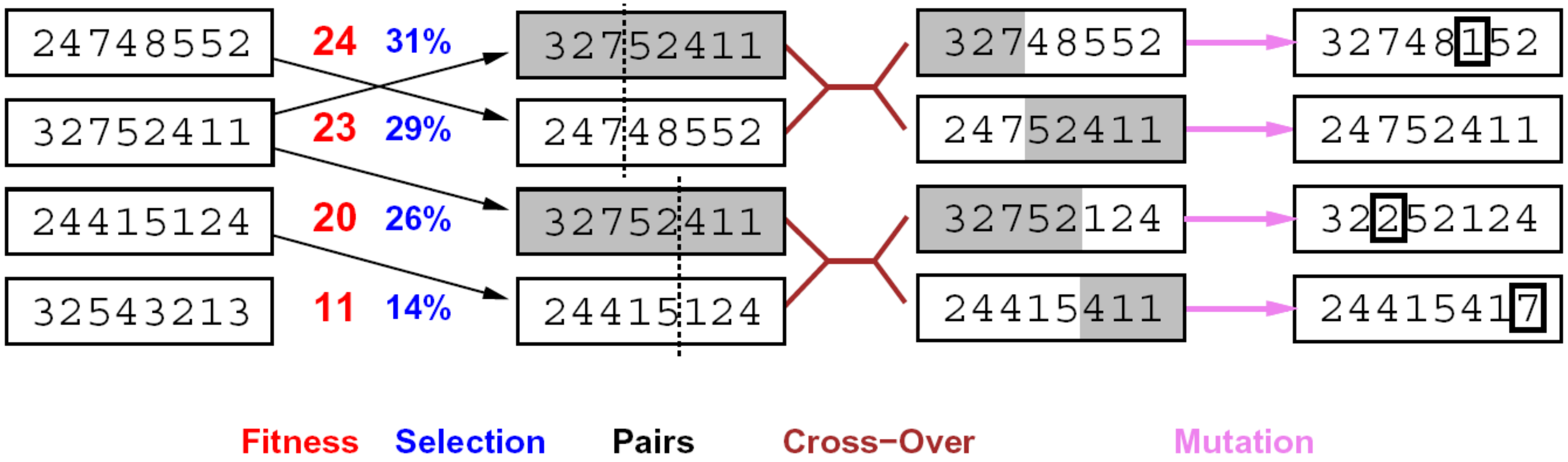
Components of GA

A problem to solve, and ...	
Encoding technique	(*gene, chromosome*)
Initialization procedure	*(creation)*
Evaluation function	*(environment)*
Selection of parents	*(reproduction)*
Genetic operators	(*mutation, recombination)*
Parameter settings	(practice and art)

Population Representation

- To implement a GA one needs to define:
  - A genetic representation of the solution space
    - Bit sequence (0101…1100)
    - real values (43.2 -33.1, ….0.0 89.2)
    - list of rules (R1 R2 R3… R33)
    - Permutations (1 3 5 2 6 4 7 8)
    - program elements (genetic programming)
    - Any data structure (trees,..)
  - A fitness function

Genetic Algorithms



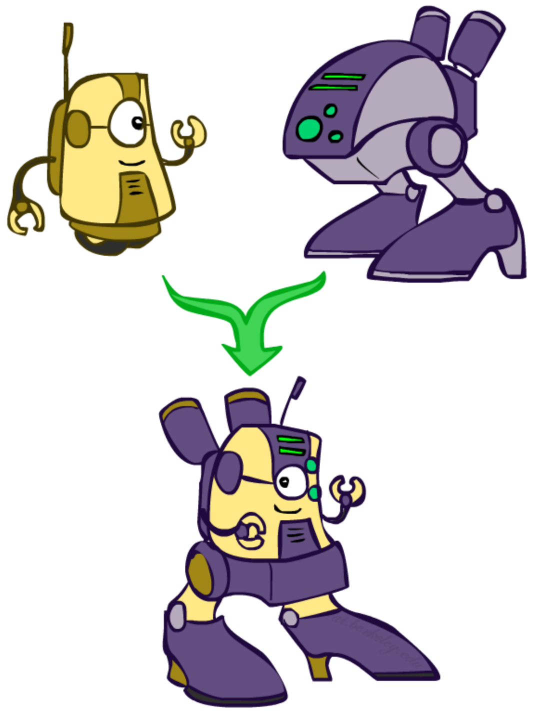
24748552 24 31% 32752411 32748552 32748152

32752411 23 29% 24748552 24752411 24752411

24415124 20 26% 32752411 32752124 32252124

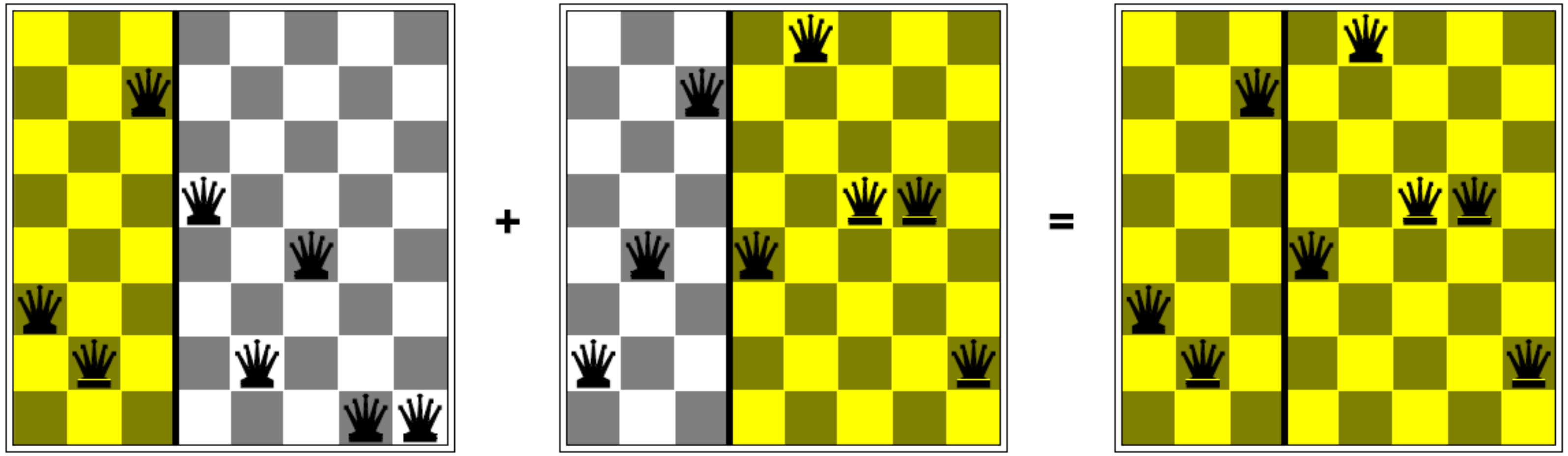
32543213 11 14% 24415124 24415411 24415417

Fitness Selection Pairs Cross−Over Mutation



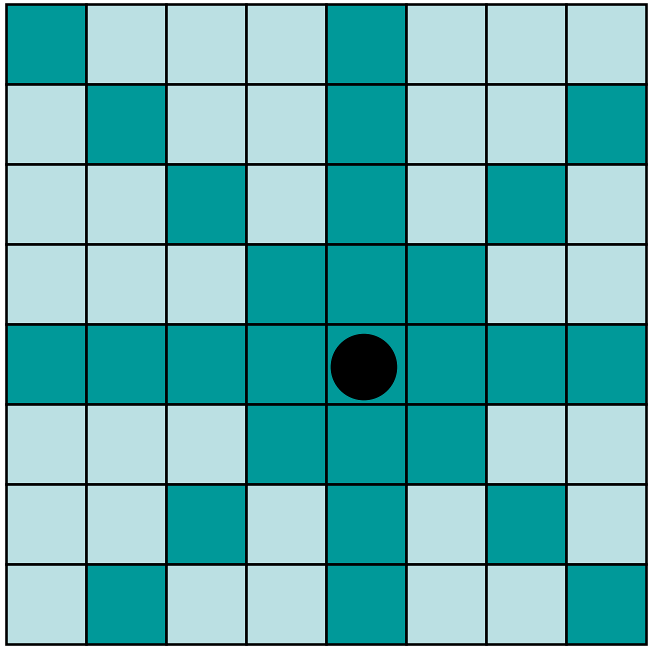
- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Example: N-Queens



- Why does crossover make sense here?
- When wouldn’t it make sense?
- What would mutation be?
- What would a good fitness function be?

8 Queens problem (1)



Place 8 queens on a 8x8 board so that they don't attack each other

8 Queens problem (2)

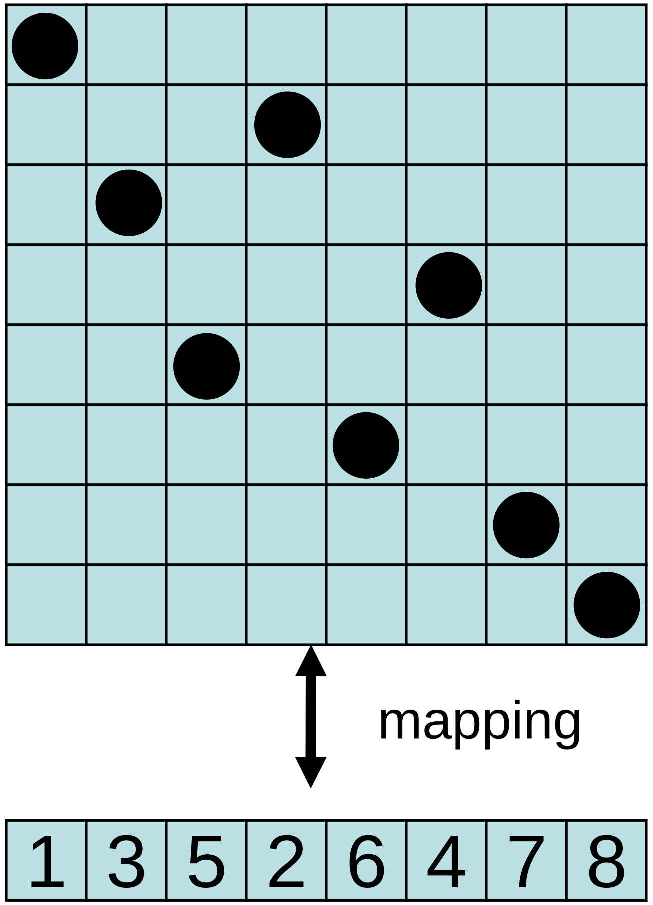
Phenotype:

Table configuration

Genotype:

Permutation of

the numbers 1-8



1	3	5	2	6	4	7	8

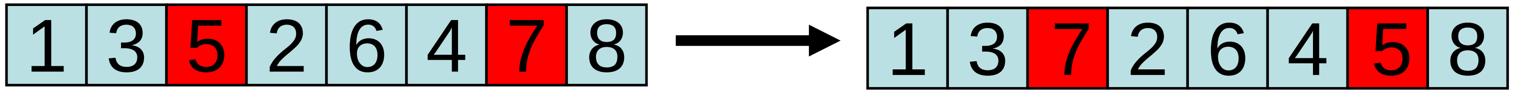
8 Queens problem (3)

- Penalty for a queen:
  - Number of queens that it attacks
- Penalty for the whole configuration:
  - Sum of penalties for all queens
- Goal: To minimize the penalty
- Fitness of the configuration:
  - Minimize the inverse of the penalty

8 Queens problem (4)

Mutations – small changes within a single permutation

E.g. Swapping the values for two randomly chosen positions



1	3	5	2	6	4	7	8

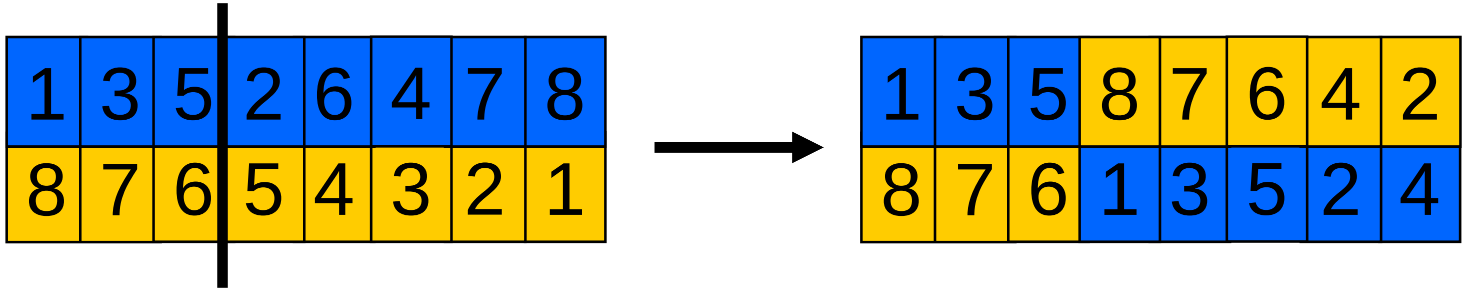
→

1	3	7	2	6	4	5	8

8 Queens problem (5)

Recombination of two permutations in a new permutation

- Random choice of a crossover point
- Copy first parts to offspring
- Create second parts by substituting from the second parent
  - same ordering as in parents
  - starting from the crossover point
  - skipping parts that are already inherited

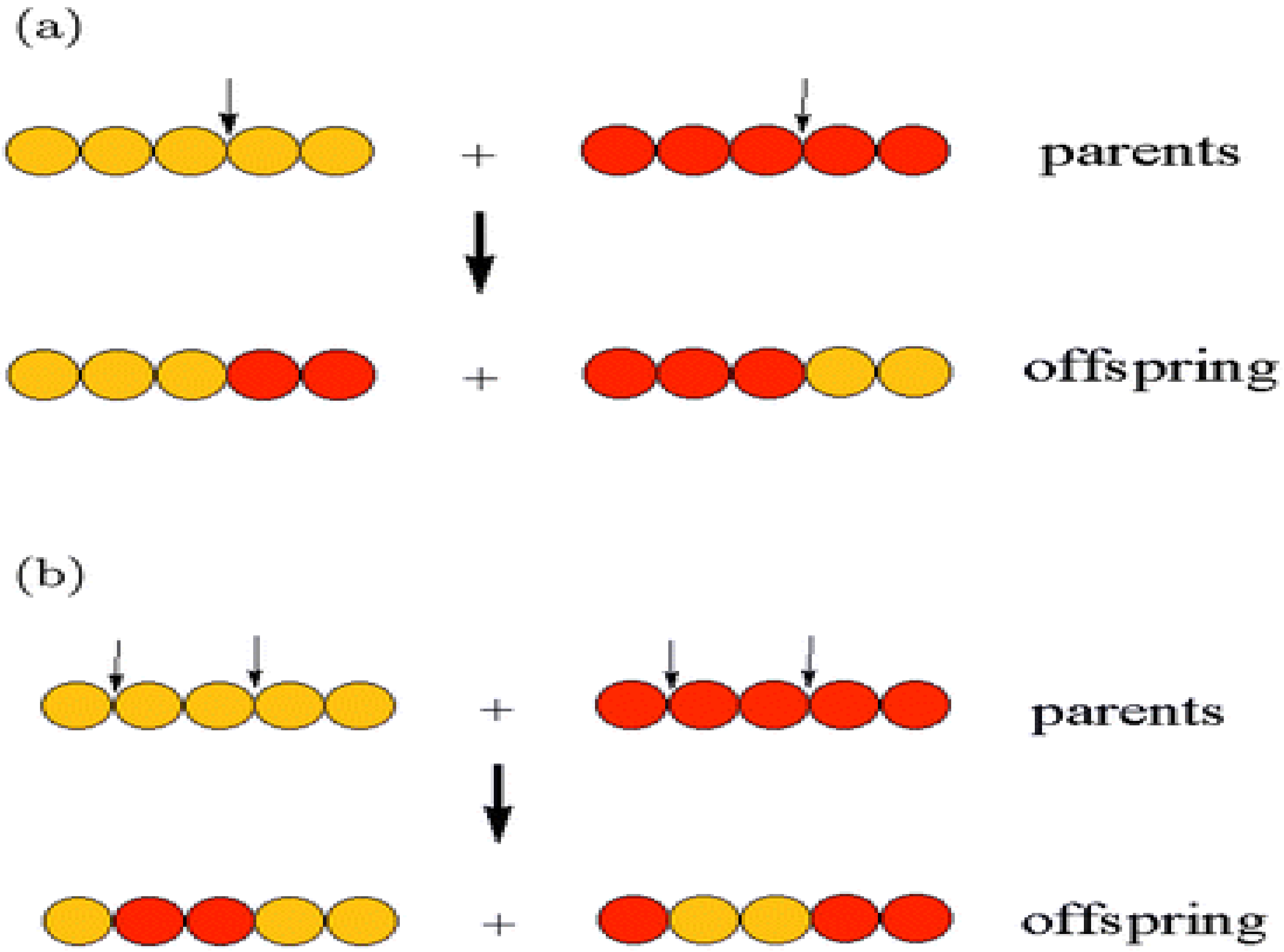


1 3 5 | 2 6 4 7 8
8 7 6 | 5 4 3 2 1

→

1 3 5 8 7 6 4 2
8 7 6 1 3 5 2 4

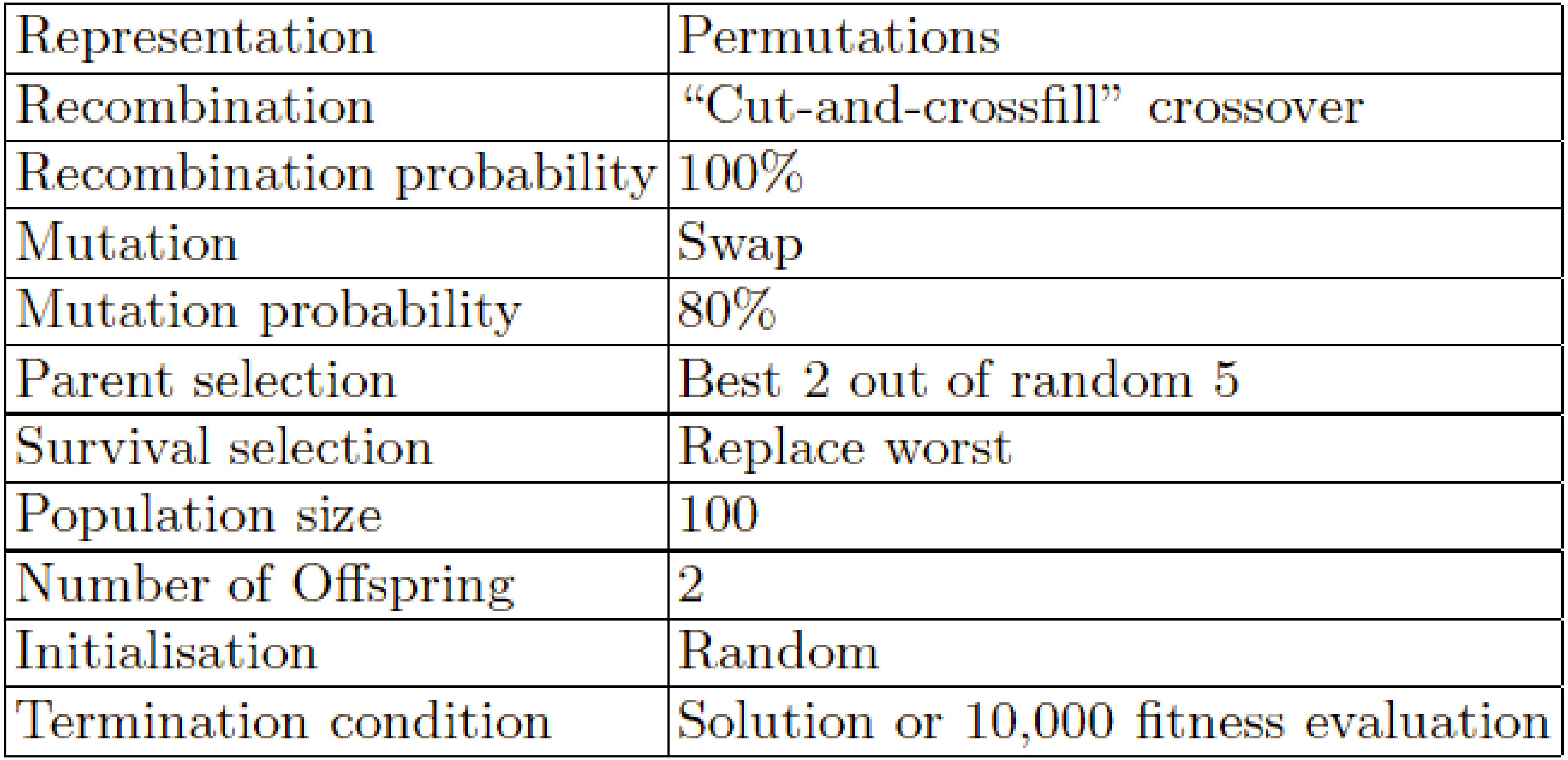
Crossover with one or two points



8 Queens problem (6)

- Parent selection
  - Take 5 parents and select 2 that will take part in the crossover
- Survivor selection
  - *Generational* GA: entire populations replaced with each iteration
  - *Steady-state* GA: a few members replaced each generation
    - Sort the population in decreasing order by the fitness function
    - Enumerate the population list starting from the member with highest fitness
    - Remove the first individual with fitness smaller then the fitness of the new member

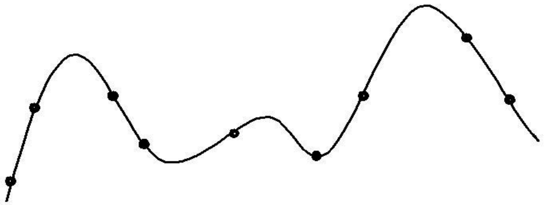
8 Queens problem (7)



Representation	Permutations
Recombination	“Cut-and-crossfill” crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

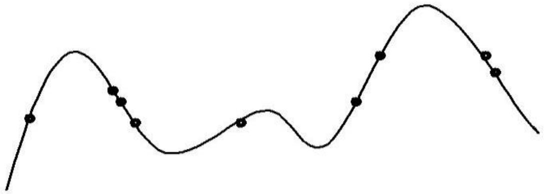
This is only one possible choice of operators and parameters

Evolutionary algorithms behavior



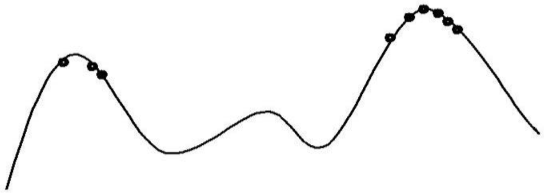
Initial phase:

Quasi-random distribution of the population



Intermediate phase:

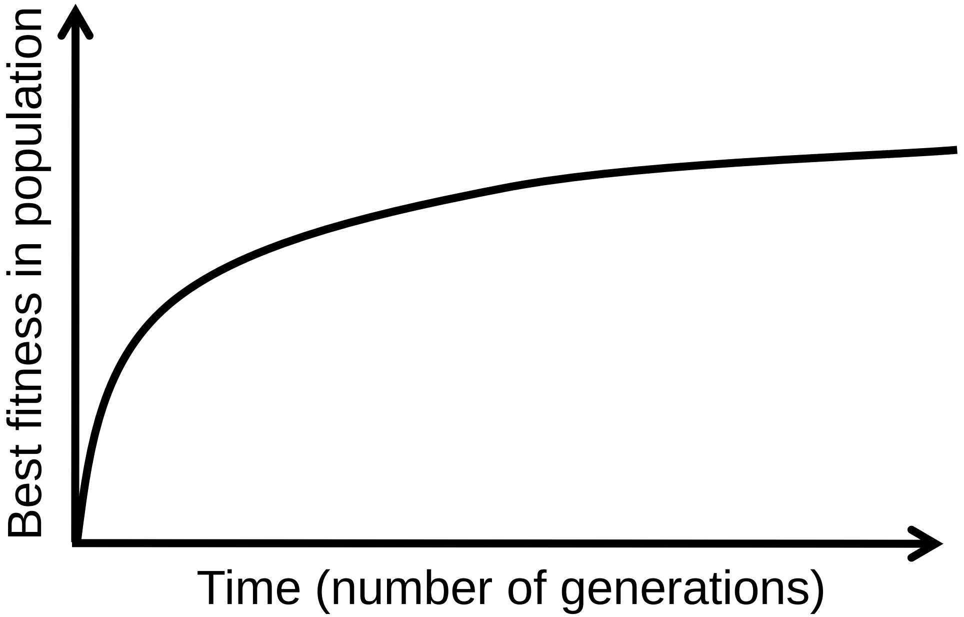
Population concentration around elevations



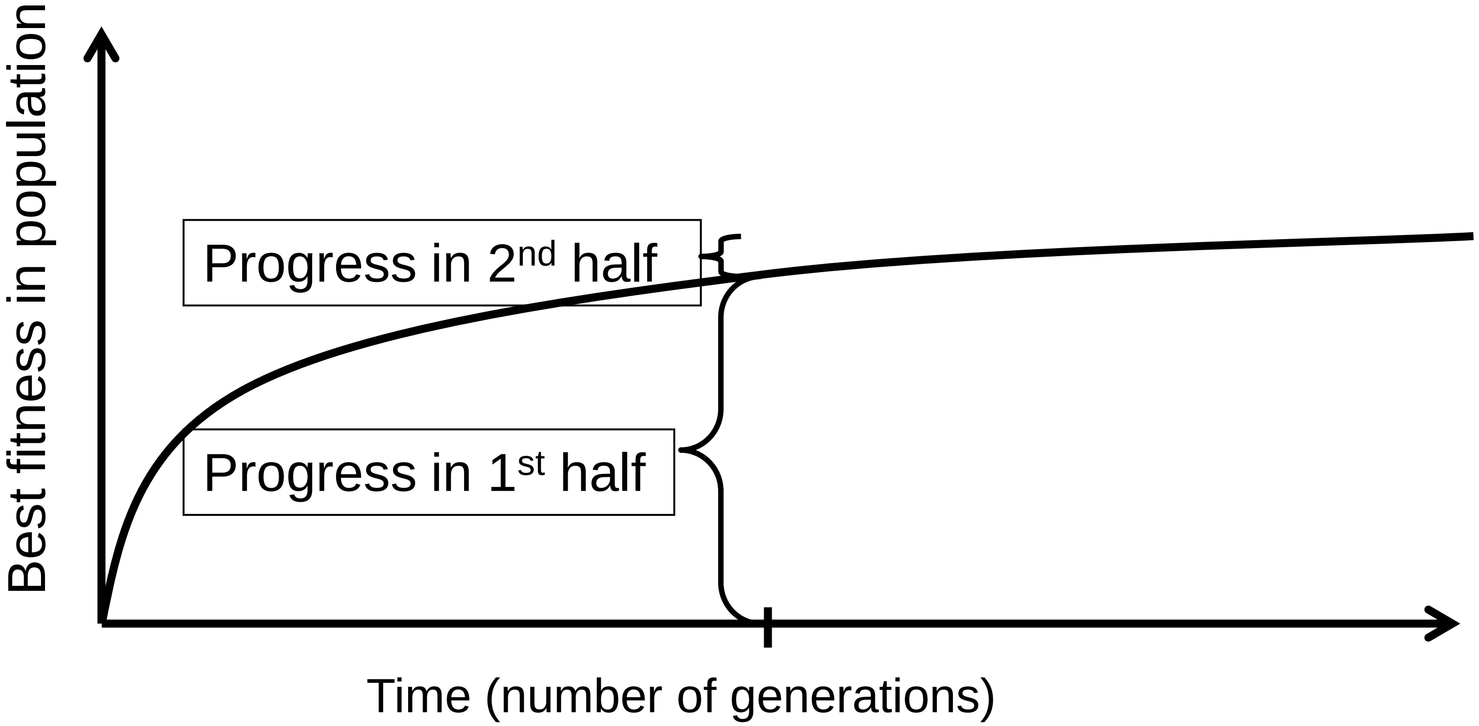
Late phase:

Population concentration around the peaks

Progression of the Fitness

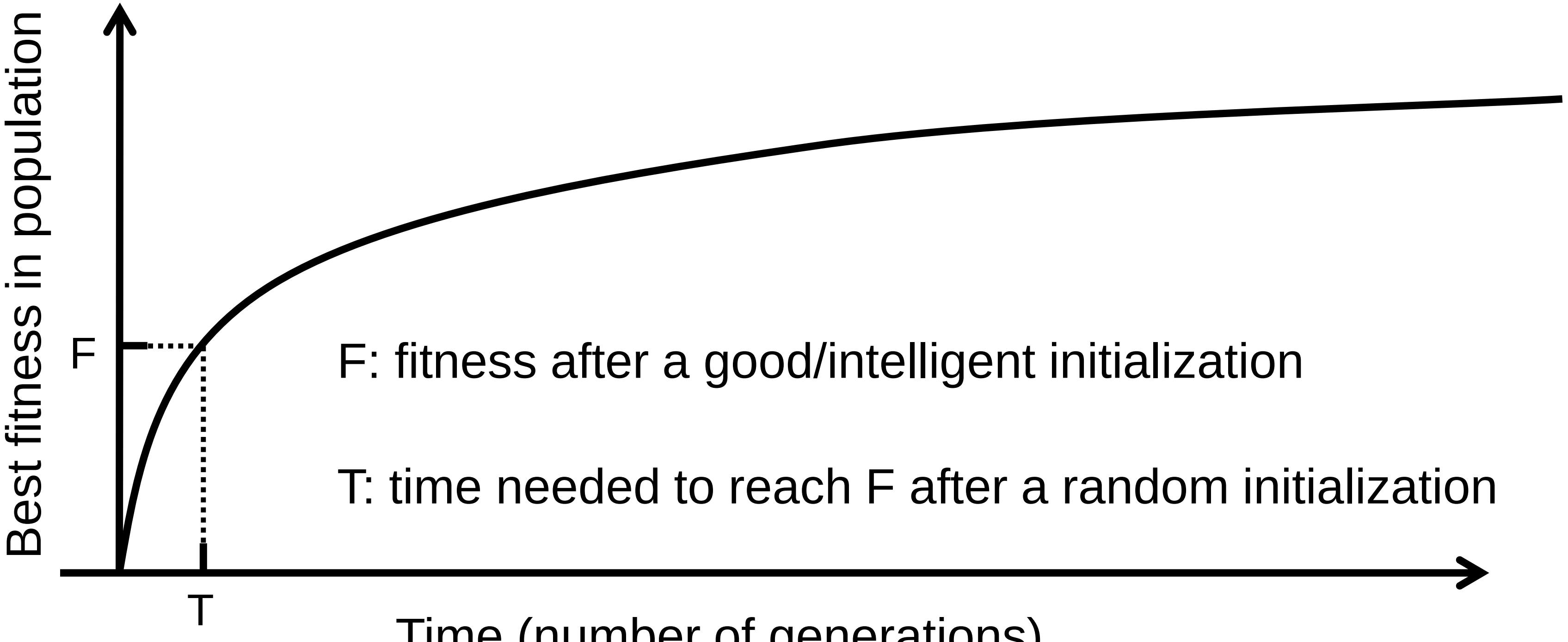


How Long Should a Run Be?



- Answer depends on:
  - ○ how much improvement is expected in the last moments of the progress/fitting
  - ○ sometimes it’s better to have shorter runs

How Important Is A Good Initialization?



F: fitness after a good/intelligent initialization

T: time needed to reach F after a random initialization

- Answer depends on:
  - whether there are good/intelligent initial solutions
  - special attention (hybridization)

Probabilistic Selection Based on Fitness

- Better individuals are preferred
- The best one is not always chosen
- The worst is not always discarded
- No guarantee for anything
- A merge of greedy and adventurous searches
- Similar to simulated annealing

Variations of Genetic Algorithms

- Initialization
- Selection (fitness-based)
  - Select the best
  - Random selection
    - roulette wheel selection
    - tournament selection.
- Reproduction
  - Crossover (single, multiple)
  - Mutation
  - Elitism
- Termination
  - A satisfactory solution is found
  - Predetermined number of generations
  - A plateau of the fitness function is reached

http://cs.felk.cvut.cz/~xobitko/ga/

Benefits of GAs

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed

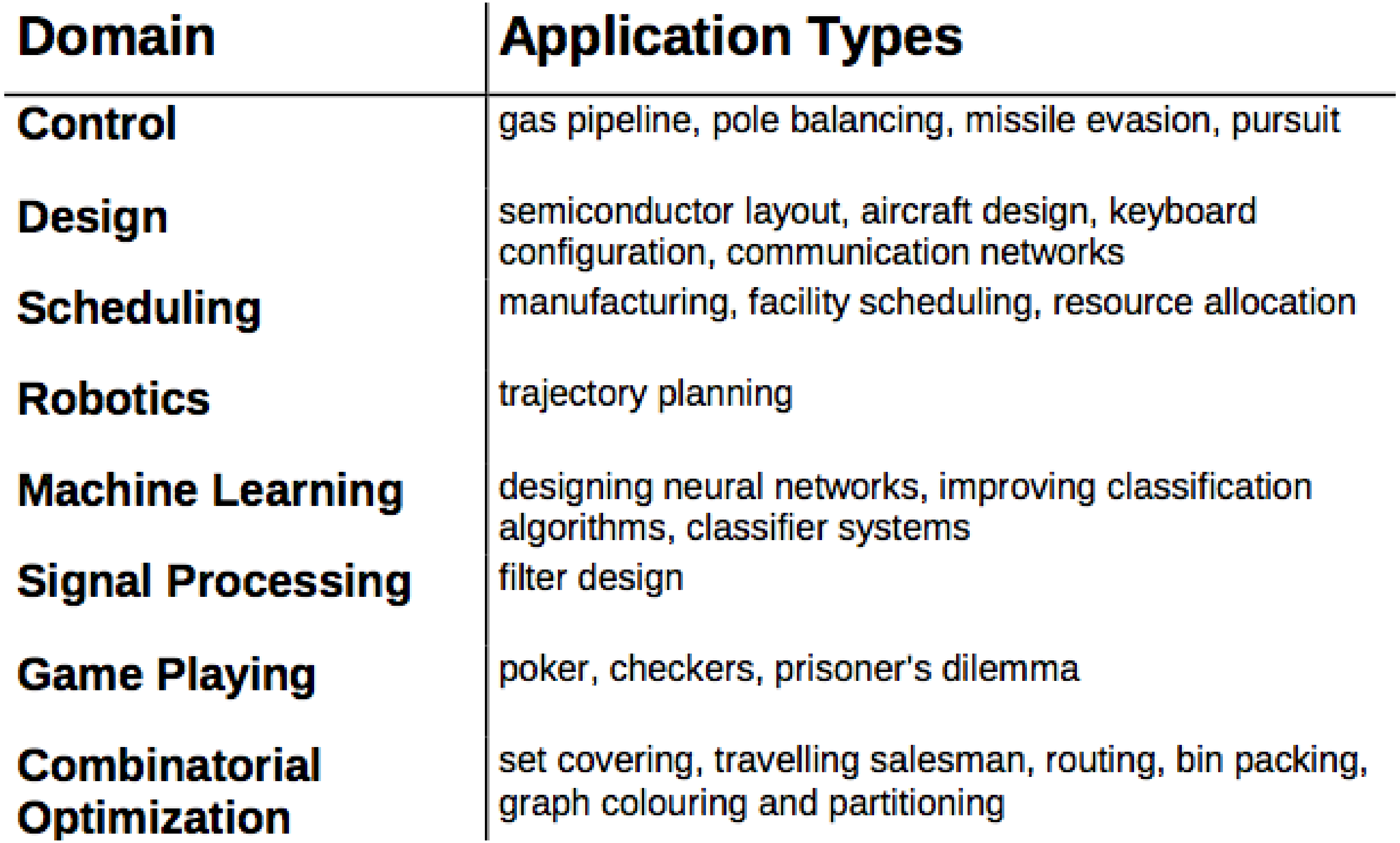
Benefits of GAs (cont.)

- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use

Benefits of GAs (cont.)

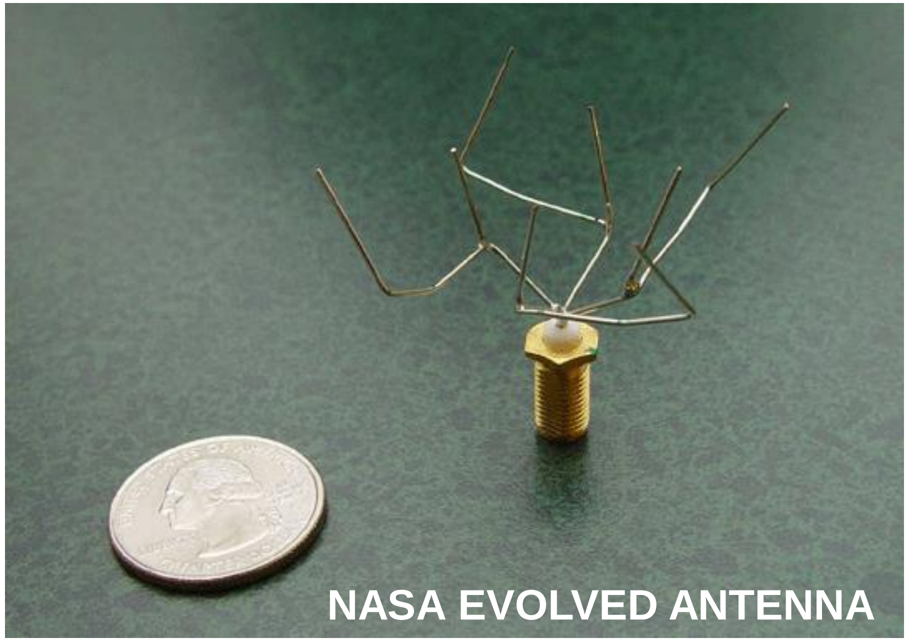
- Alternate solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements

Some GA Applications

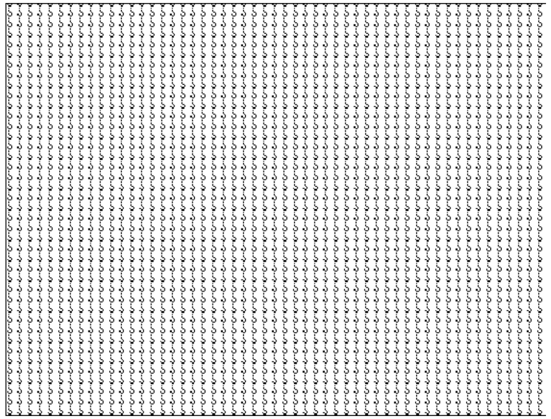


Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

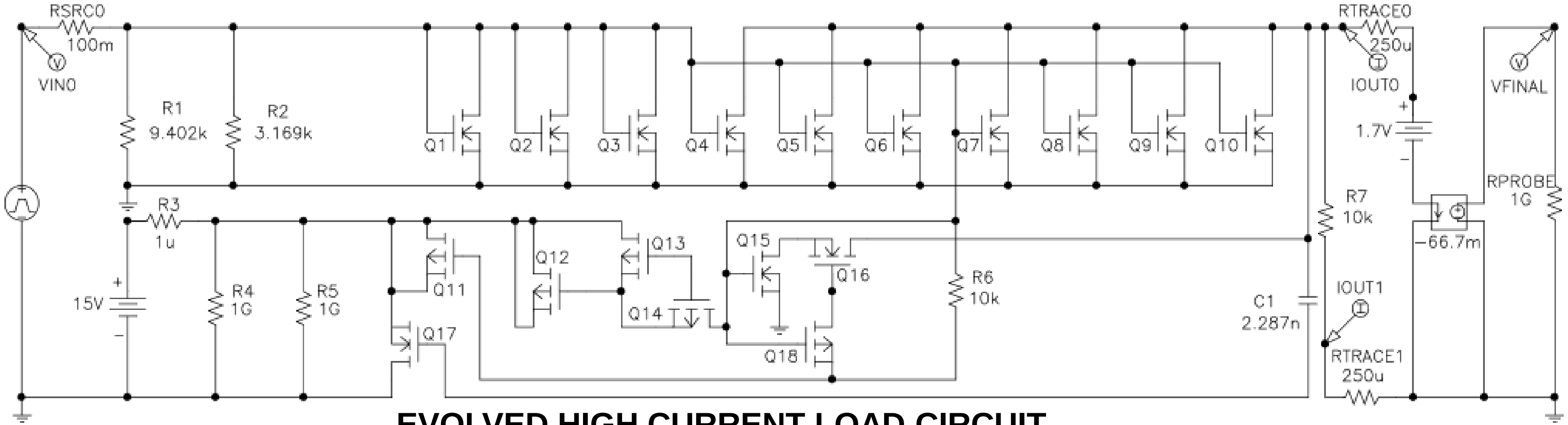
3 POST-2000 PATENTED INVENTIONS



NASA EVOLVED ANTENNA



REGISTER-CONTROLLED CAPACITOR CIRCUIT



EVOLVED HIGH CURRENT LOAD CIRCUIT

Computer creativity?

- Two portrait programs are mated together showing merged strategies of the offspring. Since the genes of each portrait can be saved, it is possible to re-combine (marry) and re-evolve any of the art works in new variants
- www.darwinsgaze.com