COE206 – Principles of Artificial Intelligence

Mustafa MISIR

Istinye University, Department of Computer Engineering

mustafa.misir@istinye.edu.tr

http://mustafamisir.github.io
http://memoryrlab.github.io

L4: Local Search[1]

[1] https://en.wikipedia.org/wiki/Local_search_(optimization)

Outline

- Optimization Problems
- Hill-Climbing
- Simulated Annealing
- Genetic Algorithms

Outline

- Optimization Problems
- Hill-Climbing
- Simulated Annealing
- Genetic Algorithms

Optimization Problems[2]

Finding the best state according to some objective function, e.g.

- timetable of classes (looks at clashes, awkward hours, unsuitable rooms ...)
- route for a garbage collector truck (visiting all the bins without driving around too much)

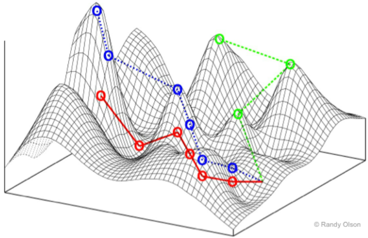
[2] http://www.cs.nott.ac.uk/~psznza/G52PAS/lecture3.pdf

Optimization Problems – Iterative Improvement[3]

Often no clear goal test and path (or its cost) to solution does not matter

In such cases, can use **iterative improvement algorithms**:

- keep a single current state, try to improve it



[3] image source: https://en.wikipedia.org/wiki/Fitness_landscape

Optimization Problems – Solution Space

Assuming the objective function gives a single numerical value, we can plot solutions against this value;

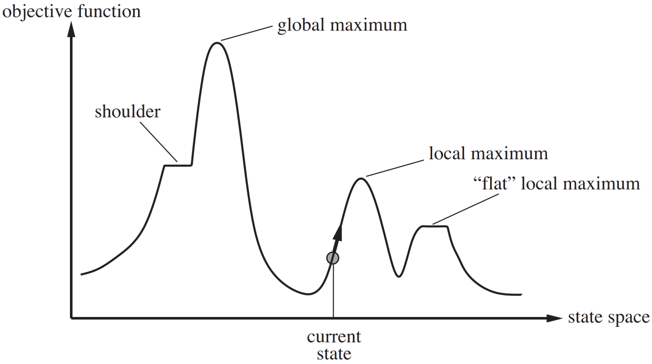
- **local search** explore this *landscape* (location is the solution and elevation is the objective function value)
- assuming the bigger the value of the function the better: we are looking for the global maximum

Complete **local search**: finds a solution if it exists
Optimal **local search**: finds a global maximum

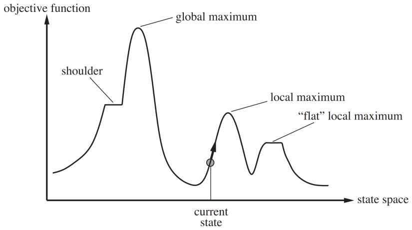
Optimization Problems – Landscape

A one-dimensional state-space landscape in which elevation corresponds to the objective function.

- The aim is to find the global maximum.

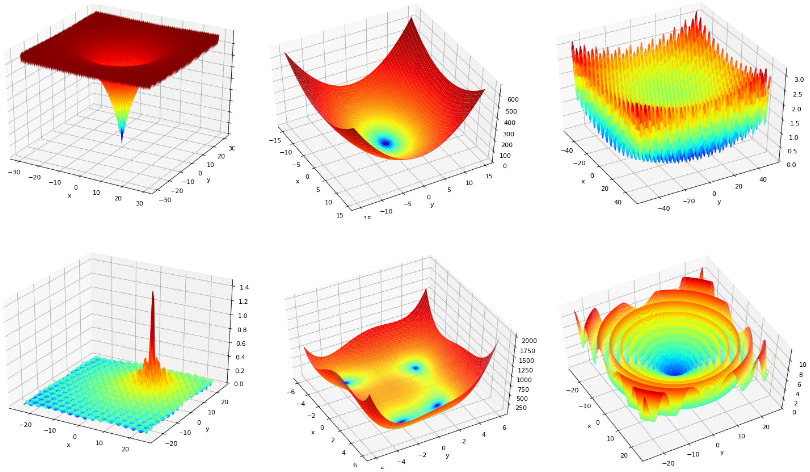


Optimization Problems – Landscape



- **Current state**: a state where an agent is currently at.
- **Global maximum**: the best possible state of state space, with the highest value of objective function.
- **Local maximum**: a state which is better than its neighbors, yet there is one or more better states.
- **Flat local maximum**: a flat space where all the neighbors of a current state have the same value.
- **Shoulder**: a plateau with an uphill edge.

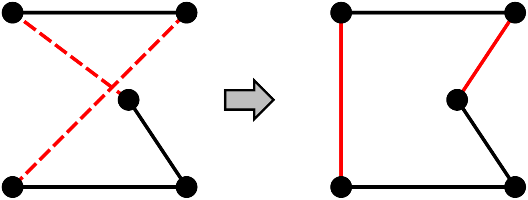
Optimization Problems – Landscape[4]



[4] graphics source: https://deap.readthedocs.io/en/master/api/benchmarks.html

Optimization Problems – Traveling Salesman[5], e.g.

Start with any complete tour, perform pairwise exchanges



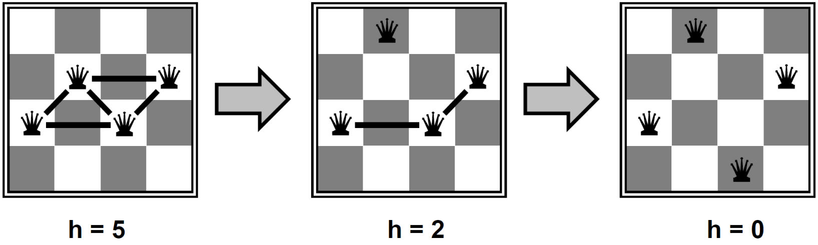
Variants of this approach get within 1% of optimal very quickly with thousands of cities

[5] https://en.wikipedia.org/wiki/Travelling_salesman_problem

Optimization Problems – n-Queens[6], e.g.

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
- Heuristic h: number of attacks



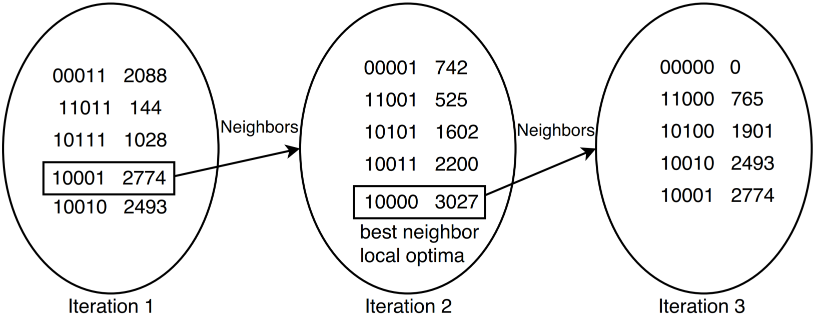
Almost always solves n-queens problems instantaneously for very large n, e.g., $n = 1$ million

[6] https://en.wikipedia.org/wiki/Eight_queens_puzzle

Local Search[7]

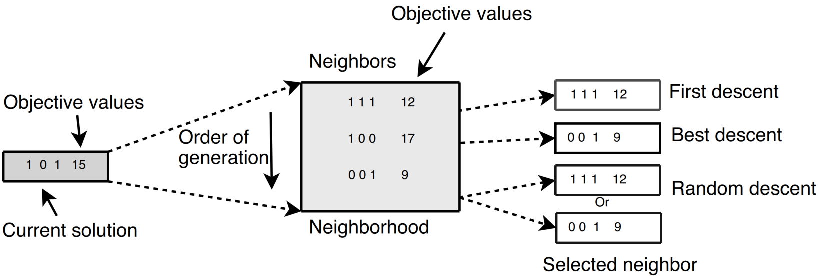
A simple algorithm, starting at a given initial solution.

- At each iteration, the heuristic replaces the current solution by a neighbor that improves the objective function



[7] Metaheuristics: From Design to Implementation by El-Ghazali Talbi - 2009 Wiley: e.g. Local search process using a binary representation of solutions, a **flip move operator**, and the **best neighbor selection strategy**. The **objective function** to **maximize** is $x^3 - 60x^2 + 900x$. The **global optimal solution** is $f(01010) = f(10) = 4000$, while the final **local optima** found is $s = (10000)$, starting from the solution $s0 = (10001)$.

Local Search – Neighbor Selection



- **Best improvement** (steepest descent / ascent): the best neighbor (i.e., neighbor that improves the most the cost function) is selected
- **First improvement**: choosing the first improving neighbor that is better than the current solution.
- **Random selection**: a random selection is applied to those neighbors improving the current solution.

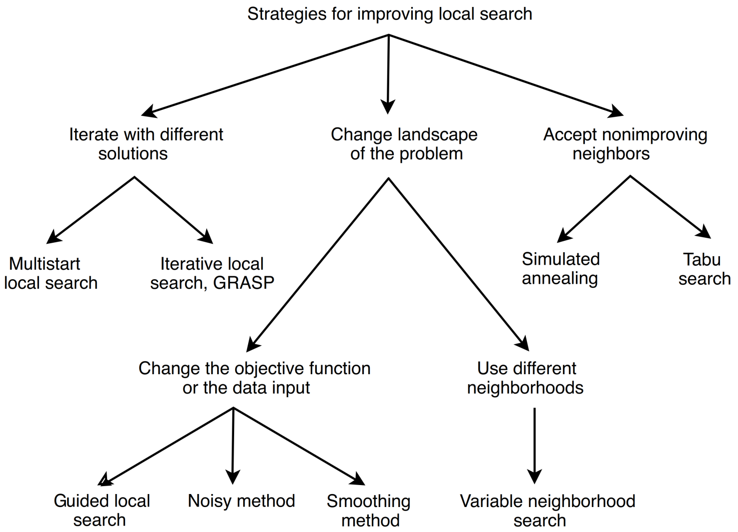
Local Search – Escaping Local Optima

One of the main disadvantages of **local search** is that it converges toward local optima.

Local optima can be avoided via 4 main strategies:

- **Iterating from different initial solutions**: as local search can be sensitive to the initial solution
- **Accepting non-improving neighbors**: degrading the current solution for moving out the basin of attraction of a given local optimum
- **Changing the neighborhood**: performed during the search
- **Changing the objective function or the input data of the problem**: playing with the objective function and the constraints

Local Search – Escaping Local Optima



Outline

- Optimization Problems
- Hill-Climbing
- Simulated Annealing
- Genetic Algorithms

Hill-Climbing[10]

The hill-climbing search[8] algorithm (steepest-ascent[9] version) is simply a loop that continually moves in the direction of increasing value—that is, uphill.

- does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.
- does not look ahead beyond the immediate neighbors of the current state

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of *current*
    if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
    current ← *neighbor*

---

8 sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.

9 vs. steepest-descent: a loop that continually moves in the direction of decreasing value—that is, downhill – https://mathworld.wolfram.com/MethodofSteepestDescent.html

10 https://en.wikipedia.org/wiki/Hill_climbing

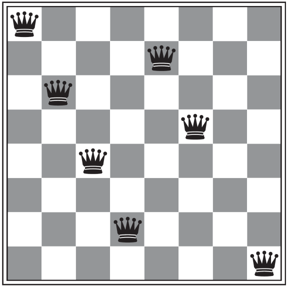
Hill-Climbing – 8-Queens, e.g.

Local search algorithms typically use a complete-state formulation, where each state has 8-queens on the board, one per column.

- The successors of a state are all possible states generated by moving a single queen to another square in the same column (so each state has $8 \times 7 = 56$ successors).

The solution space size[11] is $\binom{n = 8 \times 8}{k = 8} = 4,426,165,368$

- yet, has only 92 feasible solutions

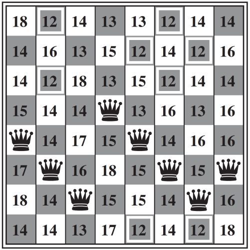


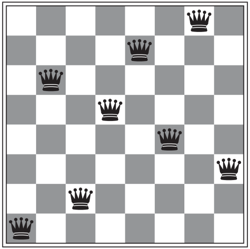
[11] combination – a selection of items from a collection, such that (unlike permutations) the order of selection does not matter $\rightsquigarrow$ $n!/k!(n-k)!$: https://en.wikipedia.org/wiki/Combination

Hill-Climbing – 8-Queens, e.g.

The heuristic cost function h is the number of pairs of queens that are attacking each other.

- The global minimum of this function is zero, which occurs only at perfect solutions.
- (figure on the left) shows a state with $h = 17$. The figure also shows the values of all its successors (obtained by moving a queen within its column), with the best successors having $h = 12$.
- Takes 5 steps to reach the state (figure on the right), which has $h = 1$.

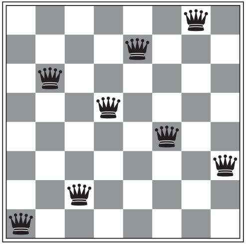




Hill-Climbing – 8-Queens, e.g.

not complete and not optimal

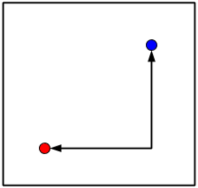
- starting from randomly generated 8-queen state, gets stuck 86% of the times
- gets stuck at local optima (below, $h = 1$ - check col. 4 and 7, white diagonal - and every change will create a worse state)

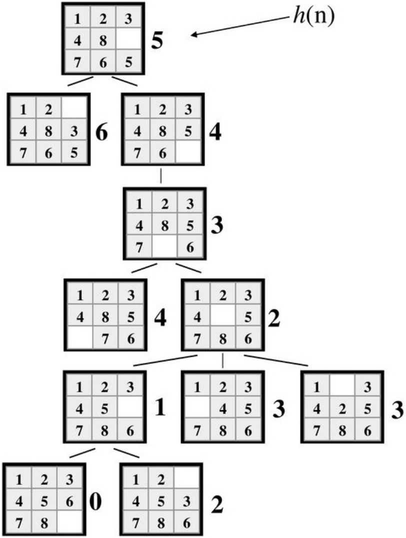


Hill-Climbing – 8-Puzzle[13], e.g.

A feasible solution (steps partially shown)

Using Manhattan distance[12] as the heuristic function – the sum of the horizontal and vertical distances.





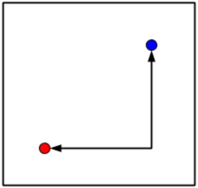
[12] https://xlinux.nist.gov/dads/HTML/manhattanDistance.html

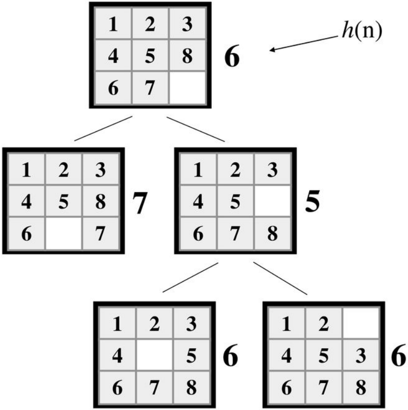
[13] https://slideplayer.com/slide/14373368/

Hill-Climbing – 8-Puzzle, e.g.

Search got stuck (steps partially shown)

Using Manhattan distance as the heuristic function – the sum of the horizontal and vertical distances.





Outline

- Optimization Problems
- Hill-Climbing
- Simulated Annealing
- Genetic Algorithms

Simulated Annealing[15]

Annealing is a process in metallurgy where metals are slowly cooled to make them reach a state of low energy where they are very strong.

- **Simulated annealing** is an analogous method for optimization.
- A version of **stochastic hill climbing**[14] where some downhill moves are allowed.
- The random movement corresponds to high temperature; at low temperature, there is little randomness
- The temperature is reduced slowly, starting from a random search at high temperature eventually becoming pure greedy descent as it approaches zero temperature.

---

[14] chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move – https://en.wikipedia.org/wiki/Stochastic_hill_climbing

[15] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P., 1983. Optimization by Simulated Annealing. Science, 220(4598), pp.671-680: https://science.sciencemag.org/content/220/4598/671 — https://en.wikipedia.org/wiki/Simulated_annealing – https://www.cs.ubc.ca/~poole/aibook/html/ArtInt_89.html – e.g. simulated annealing optimization process: https://en.wikipedia.org/wiki/Simulated_annealing#/media/File:Hill_Climbing_with_Simulated_Annealing.gif

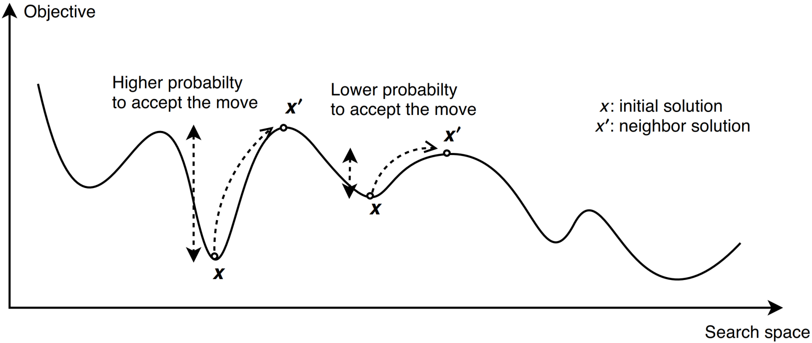
Simulated Annealing

Physical System	Optimization Problem
System state	Solution
Molecular positions	Decision variables
Energy	Objective function
Ground state	Global optimal solution
Metastable state	Local optimum
Rapid quenching	Local search
Temperature	Control parameter T
Careful annealing	Simulated annealing

Simulated Annealing

Uses a control parameter, called temperature, to determine the probability of accepting nonimproving solutions.

For a minimization problem:



Simulated Annealing

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

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

Simulated Annealing, e.g.

Let us maximize a continuous function:

$$f(x) = x^3 - 60x^2 + 900x + 100$$

- A solution x is represented as a string of 5 bits.
- The neighborhood consists in flipping randomly a bit.
- The global maximum of this function is 01010 ($x = 10$, $f(x) = 4100$).

For an initial solution of 10011 ($f(19) = 2399$)

Simulated Annealing, e.g. Scenario 1

1. $p = e^{(-112/500)} = 0.80$
2. $p = e^{(-247/405)} = 0.54$
3. $p = e^{(-16/295.2)} = 0.95$
4. ...

T = 500 and Initial Solution (10011)

T	Move	Solution	f	Δf	Move?	New Neighbor Solution
500	1	00011	2287	112	Yes	00011
450	3	00111	3803	<0	Yes	00111
405	5	00110	3556	247	Yes	00110
364.5	2	01110	3684	<0	Yes	01110
328	4	01100	3998	<0	Yes	01100
295.2	3	01000	3972	16	Yes	01000
265.7	4	01010	**4100**	<0	Yes	01010
239.1	5	01011	4071	29	Yes	01011
215.2	1	11011	343	3728	No	01011

Simulated Annealing, e.g. Scenario 2

The initial temperature is not high enough and the algorithm gets stuck by local optima.

T = 100 and Initial Solution (10011). When Temperature is not High Enough, Algorithm Gets Stuck

T	Move	Solution	f	Δf	Move?	New Neighbor Solution
100	1	00011	2287	112	No	10011
90	3	10111	1227	1172	No	10011
81	5	10010	2692	<0	Yes	10010
72.9	2	11010	516	2176	No	10010
65.6	4	10000	**3236**	<0	Yes	10000
59	3	10100	2100	1136	Yes	10000

Simulated Annealing

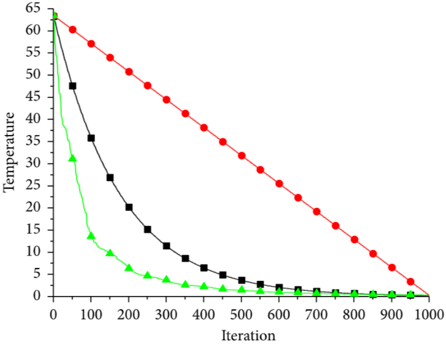
In addition to its common design issues such as the definition of the neighborhood and the generation of the initial solution, the main design issues are:

- Acceptance probability function: enables nonimproving (worsening or equal) neighbors to be selected.
- Cooling schedule: defines the temperature at each step of the algorithm.

Simulated Annealing – Cooling Schedules[16]

Different cooling schedules can be incorporated.

- Besides, adaptive schedules and reheating are also possible...



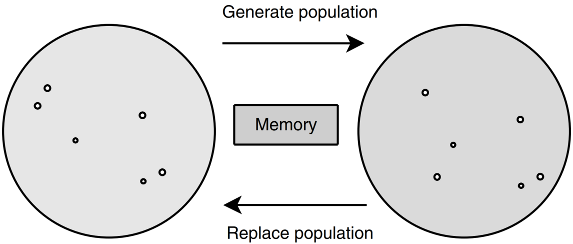
16 image source: https://www.hindawi.com/journals/cin/2016/1712630/

Outline

- Optimization Problems
- Hill-Climbing
- Simulated Annealing
- Genetic Algorithms

Genetic Algorithms[18]

A type of Evolutionary Algorithms (EAs)[17], maintaining a population of solutions instead of a single one.



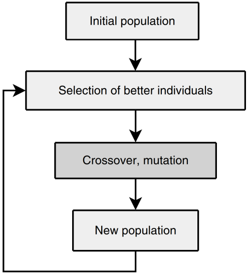
[17] https://en.wikipedia.org/wiki/Evolutionary_algorithm

[18] https://en.wikipedia.org/wiki/Genetic_algorithm

Genetic Algorithms

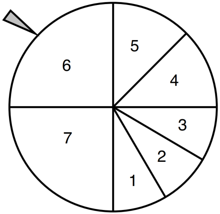
Referring to the term **genetic**, population is a solution subset of the whole solution space where each solution is represented by a chromosome composed of genes.

1. **Selection**: determine parents to be used for children (offsprings) reproduction
2. **Genetic Operators**
   - **Crossover**: mixing and matching parts of two (or more) parents to form children
   - **Mutation**: manipulating an individual (chromosome)
3. **Replacement**: The new offsprings compete with old individuals for their place in the next generation (survival of the fittest).



Genetic Algorithms – Selection

Individuals:	1	2	3	4	5	6	7
Fitness:	1	1	1	1.5	1.5	3	3



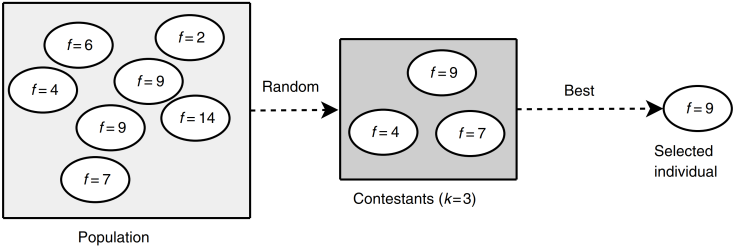
Roulette wheel selection: assign to each individual a selection probability that is proportional to its relative fitness.

- Let f_i be the fitness of the individual p_i in the population P
- The selection probability of p_i is $\frac{f_i}{\sum_{j=1}^{n} f_i}$

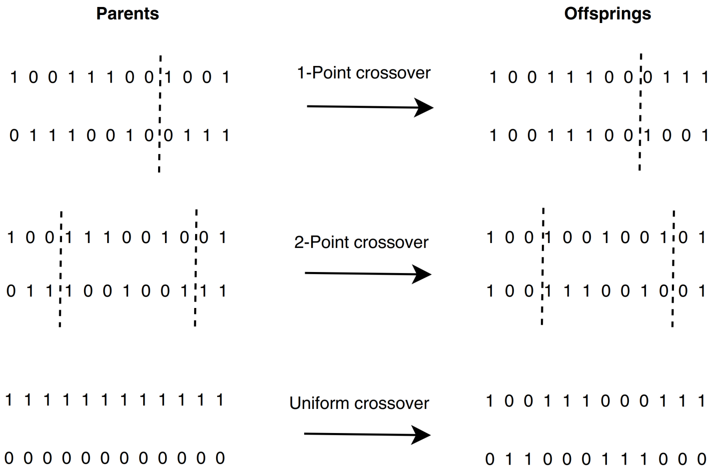
Genetic Algorithms – Selection

Tournament selection: choose the best individual among k randomly selected ones, *w.r.t* their qualities (fitness values)

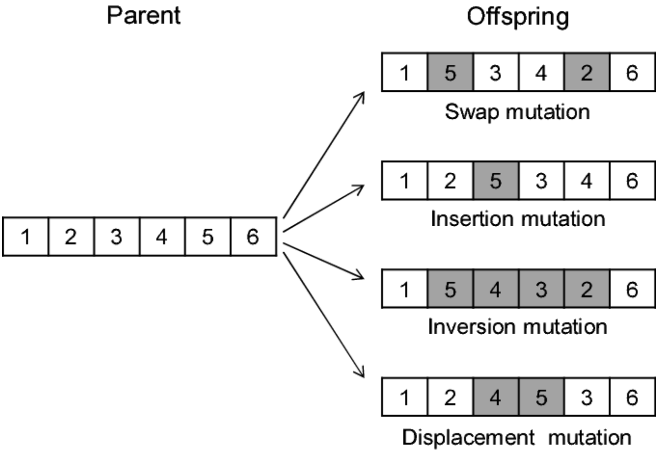
- repeat the process to choose the required number of individuals for crossover



Genetic Algorithms – Crossover



Genetic Algorithms – Mutation[19]



19 https://www.upress.uni-kassel.de/katalog/abstract_en.php?978-3-86219-551-0 – swap / exchange mutation: swap two randomly selected genes; insertion mutation: one randomly selected gene is relocated; inversion mutation: reverse the order of the genes between two random points; displacement mutation: choose two points randomly and relocate the genes as a group in-between - generalized version of the insertion mutation

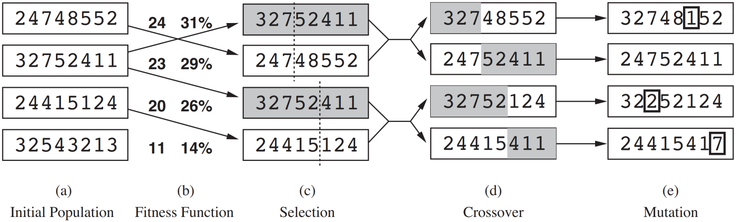
Genetic Algorithms[20]

```
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
     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) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

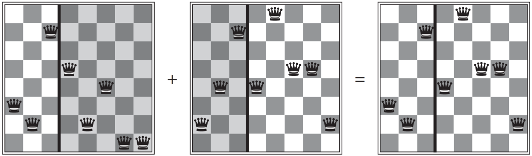
[20] in case of reproducing a single offspring with each crossover

Genetic Algorithms – 8-Queens[21], e.g.



The 8-queens states corresponding to the first two parents in (c) and the first offspring in (d).

- The shaded columns are lost in the crossover step and the unshaded columns are retained.



21 fitness values are converted into parent selection probabilities in percentages, (b)

Genetic Algorithms – 8-Queens, e.g.

Solution representation is critical

- Crossover needs to return a well-formed solution
- What if binary representation is used: each queen position requires 3 digits