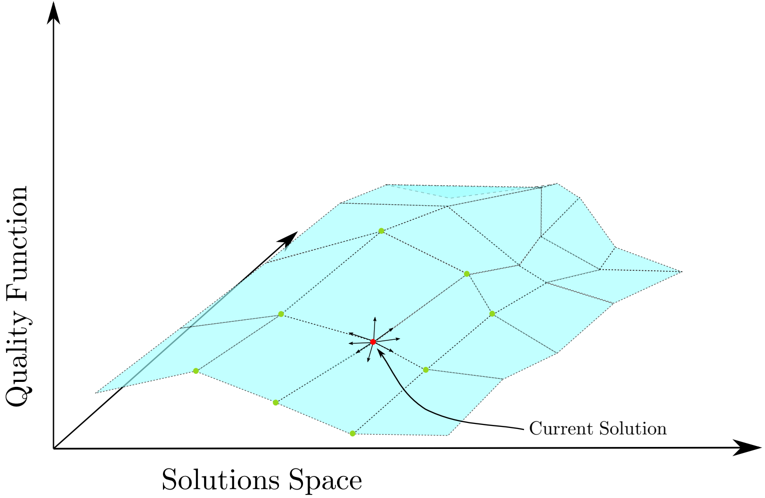
Local Search

- Sometimes we don't need the path that reaches a solution, we search in the space of solutions
- We want to obtain the best attainable solution in an affordable time (optimal is impossible)
- We have a function that evaluates the quality of the solution, this value is not related to a path cost
- Search is performed from an initial solution that we try to improve using actions
- The actions allow to move in the solution neigbourhood

Local search

- The heuristic function:
  - Approximates the quality of a solution (it is not a cost)
  - The goal is to optimize it (maximize or minimize)
  - Combines all the elements of the problem and its constraints (possibly using different weights for different elements)
  - There are no constrains about how the function can be, it only has to represent the quality relations among the solutions
  - It can be positive or negative

Local Search



Local Search

- The size of the space of solutions doesn't allow for an optimal solution search
- It is not possible to perform a systematic search
- The heuristic function is used to prune the space of solutions (solutions that don't need to be explored)
- Usually no history of the search path is stored (minimal space complexity)

Hill climbing

- First-choice Hill climbing
    - First action that improves the current solution is taken
- Steepest-ascent hill climbing, gradient search
    - The best action that improves the current solution is taken

Hill Climbing

Algorithm: Hill Climbing

```
Current ← initial state
End ← false
while not End do
    Successors ← generate_successor(Current)
    Successors ← sort_and_prune_bad_solutions(Successors, Current)
    if not empty?(Successors) then
        Current ← best_successor(Successors)
    else
        End ← true
    end
end
```

- Only are considered successors those solutions with a heuristic function value better than the current solution (pruning of the space of solutions)
- A stack could be used to store the best successors to backtrack, but usually the space requirement are prohibitive
- The algorithm may not find any solution even when there are

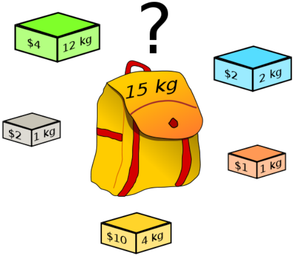
Hill climbing

- The characteristics of the heuristic function and the initial solution determine the success and the time of the search
- The strategy of this algorithm may end the search in a solution that is only apparently the optimal
- Problems
  - Local optima: No neighbor solution has a better value
  - Plateaus: All neighbours have the same value
  - Ridge: A sequence of local optima

Hill climbing

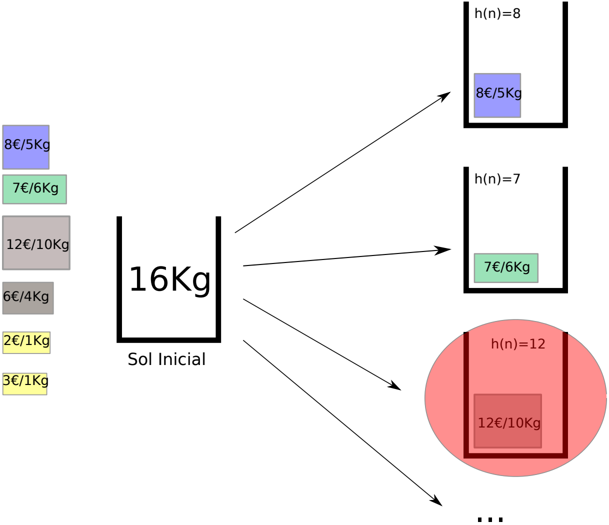
- Possible solutions
  - Backtrack to a previous solution and follow another path (it is only possible if we limit the memory used for backtracking, *Beam Search*)
  - Restart the search from another initial solution looking for a better solution (*Random-restarting Hill-Climbing*)
  - Use two or more actions to explore deeper the neighbourhood after making any decision (expensive in time and space)
  - Parallel Hill-Climbing (for instance: divide the search space in regions and explore the most promising ones, possibly sharing information)

Hill Climbing - Example - Knapsack problem

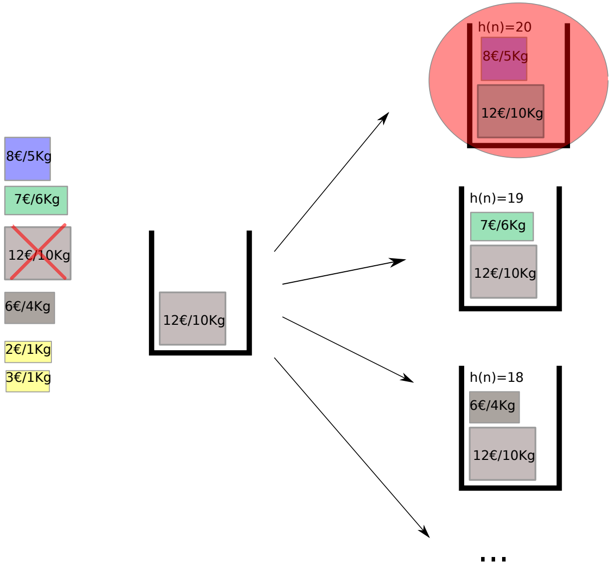


- Solution: Any combination of objects inside the knapsack
- Initial solution: Empty knapsack
- Operators: Put objects in and take objects from the knapsack
- Heuristic Function: $\max \sum_i Value_i$ or $\max \sum_i \frac{Value_i}{Weight_i}$

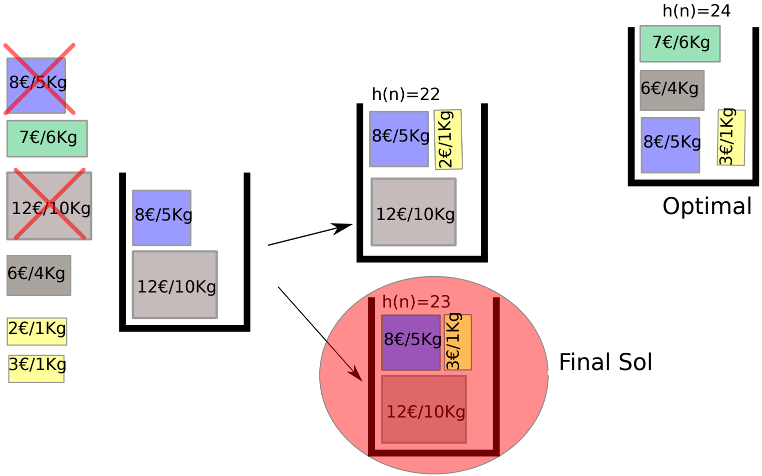
Hill Climbing - Example - Knapsack problem



Hill Climbing - Example - Knapsack problem



Hill Climbing - Example - Knapsack problem



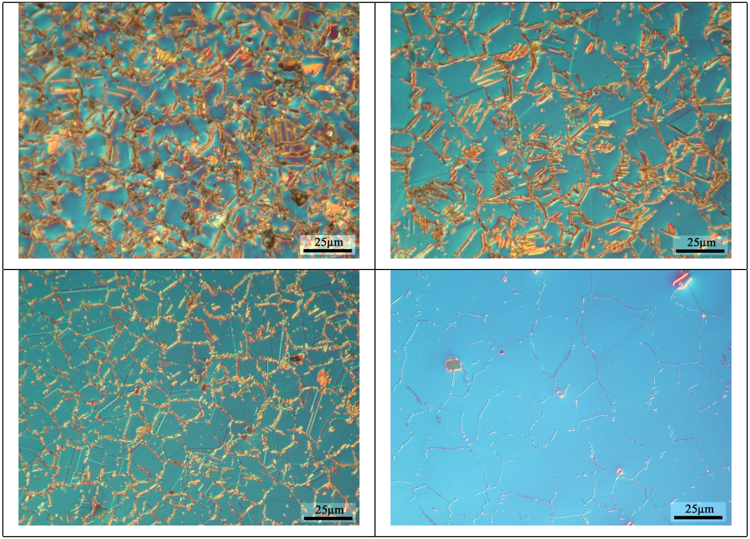
Other local search algorithms

- There are other local search algorithms with different inspirations like physics or biology:
  - **Simulated annealing:** Stochastic Hill-climbing inspired in the controlled cooling of metal alloys and substances dissolution
  - **Genetic Algorithms:** Parallel stochastic Hill-climbing inspired in the mechanism of natural selection
- But also Particle Swarm Optimization, Ant Colony Optimization, Intelligent Water Drop, Gravitational search algorithm, ...

Simulated Annealing

- Stochastic Hill-Climbing (a successor is randomly chosen from the neighbor solutions using a probability distribution, the successor could have worst evaluation than the current solution)
- A random walk of the space of solutions is performed
- Inspired in the physics of controlled annealing (crystallization, metal alloys tempering)
- A metal alloy or dissolution is heated at high temperatures and progressively cooled in a controlled way
- If the cooling process is adequate the minimal state of energy of the system is achieved (global minimum)

Simulated Annealing



Simulated Annealing - Methodology

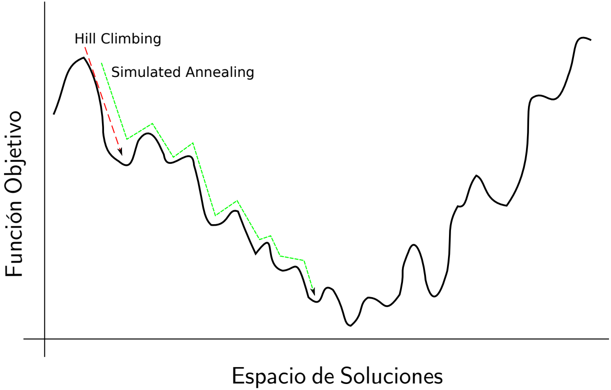
- We have to identify the elements of the problem with the elements of the physics analogy
- **Temperature**, control parameter
- **Energy**, quality of the solution $f(n)$
- **Acceptance function**, allows to decide if to pick a successor solution
  - $\mathcal{F}(\Delta f, T)$, function of the temperature and the difference of quality between the current solution and the candidate solution
  - The lower temperature, the lower the chance to choose a successor with worst evaluation
- **Cooling strategy**, number of iterations to perform, how to lower the temperature and how many successors to explore each temperature step

Simulated annealing - canonical algorithm

Algorithm: Simulated Annealing

An initial temperature is chosen
while *temperature above zero* **do**
    `// Random walk the space of solutions`
    for *the chosen number of iterations* **do**
        NewSol $\leftarrow$ generate_random_successor(CurrentSol)
        $\Delta E \leftarrow f(CurrentSol) - f(NewSol)$
        if $\Delta E > 0$ **then**
            CurrentSol $\leftarrow$ NewSol
        else
            with probability $e^{\Delta E/T}$: CurrentSol $\leftarrow$ NewSol
        end
    end
    Reduce the temperature
end

Simulated Annealing



Simulated Annealing - Applications

- Used for combinatorial optimization problems (optimal configuration of a set of components) and continuous optimization (optimal in a N-dimensional space)
- Adequate for large sized problems in which the global optimal could be surrounded by lots of local optimums
- Adequate for problems where to find a discriminant heuristic is difficult (a random choice is as good as any other choice)
- Applications: TSP, Design of VLSI circuits
- Problems: To determine the value of the parameters of the algorithm requires experimentation (sometimes very extensive)

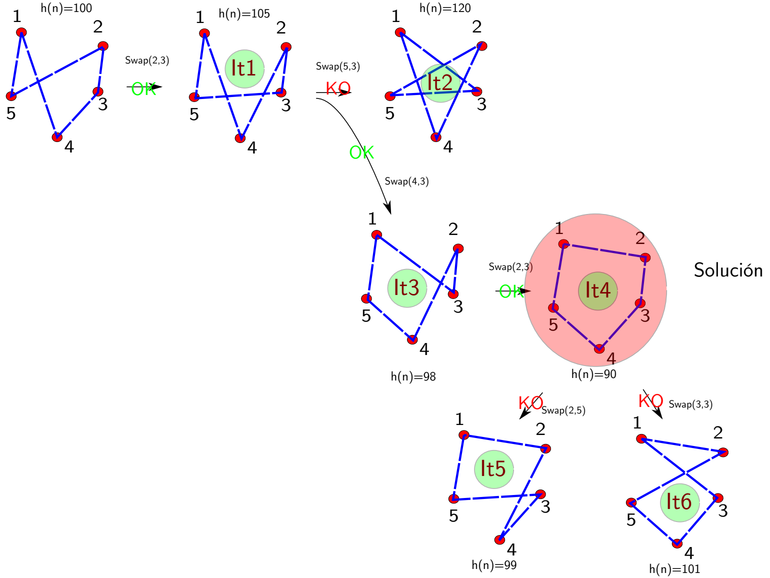
Simulated annealing - Example - TSP

- Traveler salesman problem (TSP): Search space $N!$
- Possible actions to change a solution: Inversions, translation, interchange
- An energy function (Sum of the distance among the cities, following the order in the solution)

$$E = \sum_{i=1}^{n} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}$$

- We should define an initial temperature (experimentation)
- We should determine the number of iterations for each temperature step and how is the temperature decreased

Simulated annealing - Example - TSP



Genetic Algorithms

- Inspired in the mechanisms of natural selection
  - All living things adapt to environment because of the characteristics inherited from their parents
  - The probability of survival and reproduction is related to the quality of these characteristics (fitness of the individual to the environment)
  - The combination of good individuals could result in better adapted individuals
- We can translate this analogy to local search
  - The solutions are individuals
  - The fitness function indicates the quality of the solution
  - Combining good solutions we could obtain better solutions

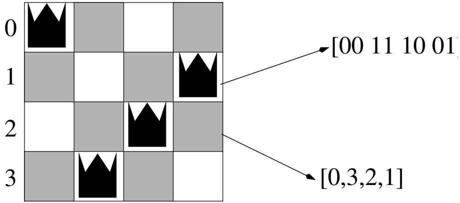
Genetic algorithms (II)

To solve a problem using GA we need:

- To code the characteristics of the solutions (for example as a binary string)
- A function that measures the quality of a solution (fitness function)
- Operators that combine solutions to obtain new solutions (crossover operations)
- The number of individuals in the initial population
- An strategy about how to match the individuals

Genetic algorithms - Coding

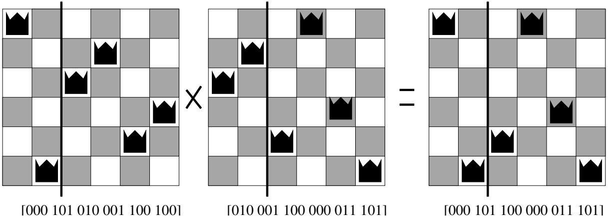
- Usually the coding of the individuals is a binary string (it is not always the best)



- The coding defines the size of the search space and the crossover operators that are needed

Genetic algorithms - Operators

- The combination of individuals is done using crossover operators
- The basic operator is the one-point crossover
  - A cutting point in the coding is chosen randomly
  - The information of the two individuals is interchanged using this point



Genetic algorithms - Operators (II)

- There are other possibilities:
  - two-points crossover
  - random bit interchanging
  - specific operators depending on the coding
- Mutation operators:
  - Following the analogy to genetics and reproduction, sometimes a part of the gene changes randomly
  - The basic mutation operator is to change with a probability a randomly chosen bit in the coding

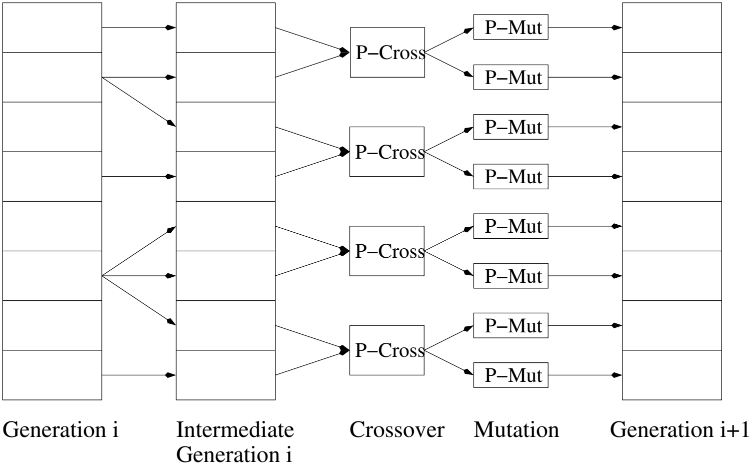
Genetic algorithms - Matching

- Each step in the search a new generation of individuals is obtained maintaining the size of the population constant (N)
- To obtain the next generation the individuals to combine (intermediate generation) are chosen following a criteria, for example:
  - Each individual is chosen with a probability proportional to its fitness
  - N random tournaments are performed among pairs of individuals, the individual that wins is chosen
  - A linear ranking among the individuals is defined using the fitness function
- Always some individuals will appear more that once and some will not appear at all

Genetic algorithms - canonical algorithm

- The basics steps of a GA are:
  1. N individuals are chosen from the current generation to form the intermediate generation (using a specific criteria)
  2. Individuals are paired and for each pair:
     - With probability (P_crossover) the crossover operator is applied and two new individuals are obtained
     - With probability (P_mutation) the new individuals are mutated
  3. These individuals conform the new generation
  4. Iterate until the population converges or a specific number of iterations is performed
- The crossover probability has a crucial influence in the diversity of the next generation
- The probability of mutation is always very low to avoid random search

Genetic algorithms - Canonical algorithm



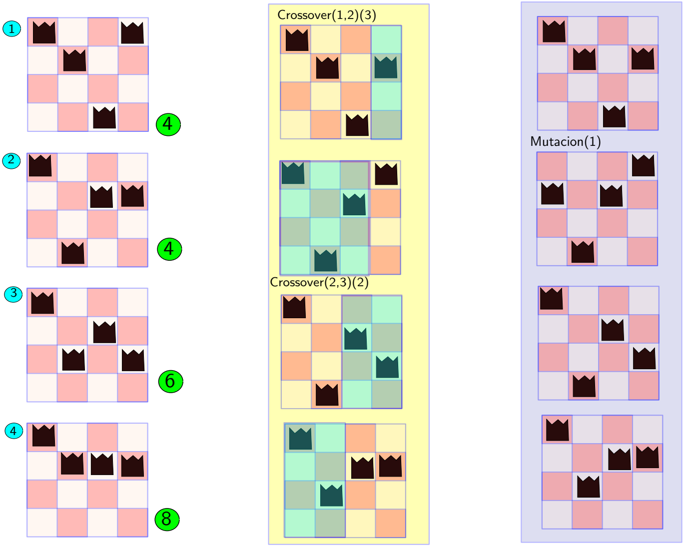
Genetic algorithms - Applications

- Are used virtually in any kind of problem
- They allow to solve problems that not have a known good heuristic
- Usually will perform worst than using hill climbing with a good heuristic
- Applications: Innumerable
- Problems: Coding the solutions, find the good parameters of the algorithm (size of the population, number of iterations, probability of crossover and mutation)
- In some problems GA perform poorly

Genetic algorithms - Example

- **N-queens problems**
- A solution can be coded as a binary string
- Individual= Concat(i=1...N; Binary(column(queen$_i$)))
- Fitness function= number of pairs of queens that attack each other
- Crossover operator= one-point crossover
- Selection of the intermediate population: Proportional to the fitness function value
- Probability of crossover → ¡experiment!
- Probability of mutation → ¡experiment!
- Size of the initial population: ? (size of the search space n^n)

Genetic algorithms - Example - N queens



Genetic algorithms - Example - N queens