Aim

the principles that underlie the conception of Metaheuristics:

- are somewhat simple
- essential to solve hard optimization problems
- often mimic natural processes

try to give a global understanding of the Metaheuristics
Outline

1. Complexity
2. Simple metaheuristics
3. Metaheuristics that mimics nature
4. More complex Metaheuristics
5. Conclusion
6. Bibliography
Definition (Complexity)

- fundamental notion in computer science
- definition of the difficulty of a problem


Garey and Johnson

how not to be fired if confronted to a complex problem?
How is defined complexity?

**Space complexity**
- defined from the number of input data $n$
- amount of memory used to solve a problem

**Time Complexity**
- defined from the number of input data $n$
- number of basic operations needed to solve a problem
The Big O notation

Definition (the big O notation)

\( O(n) \) means there are \( n \) operations needed to perform a computation
Constant complexity

- $O(1)$
- example: access to element of an array
Logarithmic complexity

- $O(\log_2(n))$
- examples: dichotomic or binary search
Logarithmic complexity

guess a number between 1 and 1000

125
...
1 250 500 1000
Linear complexity

- $O(n)$
- Examples: average of $n$ numbers, search for an element in a list
Linear complexity: search in a list

Search in an unordered list

- Item to search for is:
  - at first position (best case): $O(1)$
  - at last position or not in the list (worst case): $O(n)$
  - average is: $O((1 + n)/2) \sim O(n/2)$
Quadratic complexity

- $O(n^2)$
- example: sort $n$ integer values in an array
Cubic complexity

- $O(n^3)$
- example: naive matrix product
factorial and exponential complexities

- factorial: $O(n!)$
- exponential: $O(2^n)$
- examples:
  - travelling salesman problem
  - phylogenetic reconstruction: $(2n - 3)!$ rooted bifurcating trees
## Complexity and time

One operation is performed in 1 $\mu s$:

<table>
<thead>
<tr>
<th>complexity / $n$</th>
<th>10 $\mu s$</th>
<th>30 $\mu s$</th>
<th>60 $\mu s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>10 $\mu s$</td>
<td>30 $\mu s$</td>
<td>60 $\mu s$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>100 $\mu s$</td>
<td>900 $\mu s$</td>
<td>3600 $\mu s$</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>1000 $\mu s$</td>
<td>2700 $\mu s$</td>
<td>0.21 s</td>
</tr>
<tr>
<td>$O(n^5)$</td>
<td>0.1 s</td>
<td>24.3 s</td>
<td>13 m</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>0.001 s</td>
<td>19.9 m</td>
<td>36500 y</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>3.6 s</td>
<td>8.41e+18 y</td>
<td>2.63e+68 y</td>
</tr>
</tbody>
</table>
Problem with factorial or exponential complexity

A problem that has an exponential or factorial complexity is then generally intractable

Rice and chessboard

\[ 2^{64} = 1.8 \times 10^{19} \]
If one operation can be performed in $10^{-9}$ s then $n$ operations will last:

<table>
<thead>
<tr>
<th>#operations</th>
<th>duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sim 10^{14}$</td>
<td>1 day</td>
</tr>
<tr>
<td>$7 \times 10^{14}$</td>
<td>1 week</td>
</tr>
<tr>
<td>$3 \times 10^{15}$</td>
<td>1 month</td>
</tr>
<tr>
<td>$3.65 \times 10^{16}$</td>
<td>1 year</td>
</tr>
</tbody>
</table>
Intractability

Tractability of a category of problems

- however there is a category of problems with an exponential or factorial complexity that are tractables!
- they don’t require to perform all computations to get a solution
- they are called optimization problems
The Travelling Salesman Problem (TSP)

- **Input data:**
  - A set of cities
  - Distances between each pair of cities
  - An integer bound $B$

- **Question:** is there a Hamiltonian cycle of cost $\leq B$

- **Complexity:** $O(n!)$

---

**In other words**

Given a number of cities and the costs of travelling from any city to any other city, what is the cheapest round-trip route that visits each city once and then returns to the starting city?
The Travelling Salesman Problem

Simple example of TSP

kilometers between cities

<table>
<thead>
<tr>
<th>cities</th>
<th>Angers</th>
<th>Nantes</th>
<th>Lyon</th>
<th>Marseille</th>
<th>Bordeaux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angers</td>
<td></td>
<td>88</td>
<td>607</td>
<td>907</td>
<td>384</td>
</tr>
<tr>
<td>Nantes</td>
<td>88</td>
<td></td>
<td>695</td>
<td>970</td>
<td>334</td>
</tr>
<tr>
<td>Lyon</td>
<td>607</td>
<td>695</td>
<td></td>
<td>315</td>
<td>568</td>
</tr>
<tr>
<td>Marseille</td>
<td>907</td>
<td>970</td>
<td>315</td>
<td></td>
<td>647</td>
</tr>
<tr>
<td>Bordeaux</td>
<td>384</td>
<td>334</td>
<td>568</td>
<td>647</td>
<td></td>
</tr>
</tbody>
</table>
The Travelling Salesman Problem

- Angers
- Nantes
- Bordeaux
- Marseille

Distance:
- 3524 km
- 1991 km
The Travelling Salesman Problem

What if:
Importance of the TSP in CS and OR

Importance of TSP

- One of the most intensely studied problems in Computer Science (Artificial Intelligence) and Mathematics (Operational Research)
- Applications
  - logistics (obvious)
  - jobs scheduling
  - controlling satellites
  - designing telecommunication networks
  - designing and testing VLSI
  - clustering data arrays [5], Genome sequencing
Importance of the TSP in CS and OR

Importance of TSP

In 1972, Karp[4] proved that TSP is **NP-hard**
- exact methods run in exponential time
- impossible (for the moment) to solve NP-hard problems in polynomial time
- whether this can be done is a famous open question in theoretical computer science.
Definition (Combinatorial optimization)
In applied Mathematics and theoretical Computer Science, **combinatorial optimization** is a topic that consists of finding an optimal object in a finite set of objects.

Definition (Optimization problem)
Find the best solution from all feasible solutions.
Exact or approximate method?

Definition (Exact method)
will provide the exact solution (global optimum) but can take an eternity!

Definition (Approximate method)
- will provide an optimal, sub-optimal or near-optimal solution with less computation time than an exact method
- we are not guaranteed to obtain the optimum!
What are Metaheuristics?

Definition (Metaheuristics)

- comes from the greek words:
  - *meta*: beyond
  - *heuriskein*: to find

- high level strategies (approximate methods) for exploring a huge search space in order to find the best configuration

*high level* means more clever than a simple random search
Diversification and intensification

**Definition (Diversification and intensification)**

cleverness of Metaheuristics will come from two complementary notions (stem from Tabu Search [3]):

- **diversification**: exploration of the search space
- **intensification**: exploitation of the accumulated search experience
Simple metaheuristics
Model

Elements of the model

To solve an optimization problem we need to define three elements:

- a search space
- an objective function
- a neighborhood
Search space

**Definition (Search space)**

a set of configurations (individuals) that can be clearly identified that represent potential solutions of the problem

**Examples**

- a population
- a land
- a forest
Objective function

Definition (Objective function)
- a function that gives an estimate of the **quality** of a configuration
- helps compare configurations
- also known as *quality, cost* or *fitness* function

Examples
- altitude
- size of a tree
**Neighborhood**

**Definition (Neighborhood)**
returns the neighbors: a set of configurations that are close to a given configuration

**Examples**
- points around a given position \((x, y)\)
- relatives or friends
- objects surrounding a given object
Example

find the point that maximizes the function $f(x, y)$ such that:

$$
\begin{align*}
  f(x, y) &= y \times \log\left(\frac{x+1}{2}\right) \times \sin\left(\frac{x}{0.4}\right) \times \cos(x) \\
  x, y &\geq 0 \\
  x, y &\leq 6 \\
  x + y &\leq 9
\end{align*}
$$
**Example**

**Figure:** Landscape of $f(x, y)$

$f_{\text{max}}(5.5, 3.5) = 2.7070823$
Local search methods

Definition (Local search / Hill-Climbing / Descent)

A stochastic process to find a best solution

1. start from an initial solution
2. iteratively move to a better neighbor
3. stop if a maximum number of iterations is reached or go back to point 2
LS and neighbors

Should we make a thorough search and examine all neighbors?

- **no**: less computation, *best first search*

- **yes**: more computation,
  - what to do if there are several neighbors of same quality? (Parallelization)
Rough and smooth landscapes

A landscape is:

- **smooth**: not many variations between neighbors
- **rough**: many variations between neighbors
The local optimum problem

Definition (Local optimum)
- a configuration better than all its neighbors
- but not the best \((\neq \text{global optimum})\)

\[ f(x,y) \]

**global optimum**

**local optimum**
The local optimum problem

LS and local optimum

Local search gets stuck in local optima
The local optimum problem

Examples of two different LS

START WITH [2.0 3.0 ], score = 0.48540655
NEW BEST [2.0 3.5 ], score = 0.56630766
NEW BEST [2.0 4.0 ], score = 0.64720875
NEW BEST [2.0 4.5 ], score = 0.7281099
NEW BEST [2.0 5.0 ], score = 0.8090109
NEW BEST [2.0 5.5 ], score = 0.88991207
NEW BEST [2.0 6.0 ], score = 0.9708131
LOCAL OPTIMUM [2.0 6.0 ] 0.9708131

START WITH [5.0 4.0 ], score = -0.08267284
NEW BEST [4.0 3.5 ], score = 1.1404024
NEW BEST [4.0 4.0 ], score = 1.3033171
NEW BEST [4.0 4.5 ], score = 1.4662316
NEW BEST [5.5 3.5 ], score = 2.7070823
LOCAL OPTIMUM [5.5 3.5 ] 2.7070823
How to escape from a local optimum?

- accept neighbors of same quality
- accept neighbors of lower quality (SA)
- start search from new configuration (RR, GRASP)
- modify the current configuration (ILS)
- change the neighborhood (VNS)
- change the objective function
RR and GRASP

- **Random Restart**: several independent searches
- **Greedy Randomized Adaptive Search Procedures** [8]: construct feasible solutions submitted to LS

Start new search from another configuration.
GRASP

- construct a first set of diverse high quality configurations
  - use a **greedy** algorithm to build a final configuration by **randomly** selecting a best neighbor in a candidate list
- submit each one to LS and keep the best

**Definition (greedy algorithm)**
make the locally optimal choice at each step
Iterated Local Search

- perturbation of current configuration followed by LS
- could be called the *flea algorithm* (see next section)
Iterated Local Search
Variable Neighborhood Search

- change the neighborhood during the search [6]
  - once stuck, use a wider neighborhood to escape from local optimum
  - $N_1 \subset N_2 \subset \ldots \subset N_k$
  - generally $k = 2$
  - involve more computations
Tabu search

Definition (Tabu search)

from *Fred Glover 1989*

- keep a history of visited configurations (*tabu list*)
- avoid to visit those configurations for a given time (= number of iterations)
- avoid cycles
Metaheuristics that mimics nature
Simulated Annealing
Simulated Annealing

Definition (Simulated Annealing)

- derived from the Metropolis algorithm or metal annealing processing [10]
- decrease the temperature to reach a stable low-energy configuration
- use number of iterations instead of temperature \( T \)
- accepts changes that improve the objective function
- accepts configuration of lower quality with probability \( p \) that decreases with \( T \)
Simulated Annealing

Properties of Simulated Annealing

- ability to avoid being trapped in local minima
- becomes a hill-climbing at very low temperatures (= important number of iterations)
Evolutionary Algorithms
Evolutionary algorithms

Definition (Evolutionary algorithms - EA)
apply principles of Darwin and Lamark to resolution of problems in computer science:

- evolution is based on competition and selection
- that leads to survival of the fittests,
- transmission of acquired characters to descendants
- mutation of genes
- began in the 50s [14]
Evolutionary algorithms

Approaches

- Evolutionary Programming
- Evolution Strategy
- Genetic Algorithms (GA)
Genetic Algorithms
Genetic algorithm

Definition (Genetic algorithm)
- introduced by Holland in 1975 [12]
- based on the natural selection process
- population-based algorithm
- configuration regarded as a chromosome (vector representation)
Metaheuristics for bioinformatics

Metaheuristics that mimics nature

Genetic algorithm

Natural selection process

1. start from an initial **population** of individuals
2. iteratively generate neighbors by combining individuals
3. apply mutation and selection to keep **better** individuals
4. remove old individuals or individuals of a lower quality
Genetic algorithm

Generation of a new individual

- **crossover**: combine two or more individuals to get one or a set of children
- **mutation**: perform mutation on children
- **selection**: keep children that are better than their parents, remove old or inappropriate individuals
Genetic algorithm

Example of crossover operator

Parent 1

\[ x_1 \quad \ldots \quad x_n \]

Parent 2

\[ y_1 \quad \ldots \quad y_n \]

Children

\[ x_1 \quad x_i \quad y_{i+1} \quad y_n \]

\[ y_1 \quad y_i \quad x_{i+1} \quad x_n \]
**Example of crossover operator**

from parents (2, 5) and (3, 1) we can obtain the following children:

- exchange values: (2, 1), (3, 5)
- combine values: (2.5, 3)
Genetic algorithm

Fitness function

Landscape

P1

P2

P3

children
Genetic algorithm

Problems

- **crossover**: must be tailored to the problem to solve
- **population**:  
  - try to keep diversity  
  - avoid consanguinity
Memetic Algorithms
Memetic or hybrid algorithms

Definition (Memetic or hybrid algorithms)
combination of an **genetic algorithm** and a **local search** algorithm

- **genetic algorithm**: diversification (explore the search space)
- **local search**: intensification (exploitation)
Swarm Intelligence
Swarm Intelligence

Definition (Swarm Intelligence - SI)

- G. Beni and J. Wang, 1989 for cellular robotic systems
- based on nature observation
- collective behaviour of decentralized, self-organized systems
- computer science : agents
Swarm Intelligence

Examples of SI algorithms
- Animal communication (Ants, Bees, ...)
- River Formation Dynamics (RFD)
- Gravitational Search Algorithm (GSA)
- Intelligent Water Drops (IWD)
- Particle Swarm Optimization (PSO)
Ant Colony Optimization
Ant Colony Optimization

Definition (Ant Colony Optimization - ACO)

- proposed by *Dorigo et al, 1991* [9]
- behavior of ants seeking a path between their colony and a source of food
- use of *pheromones*: paths with more pheromones are more likely to be taken
Ant Colony Optimization

**Figure:** source Wikipedia
Bees algorithm

Definition (Bees algorithm - BA)

- proposed in *Pham et al, 2005* [11]
- mimics the food foraging behaviour of swarms of honey bees
- combines diversification and intensification
- kind of memetic algorithm
Bees algorithm

Bees behaviour

- **diversification**: create population of bees
- **intensification**: affect part of bees to *interesting* sites
- **diversification**: let the other part examine different areas
Bees

initial positions of bees
after a few iterations
intensification and diversification
Other nature-based algorithms

- **Firefly Algorithm**: attractiveness is proportional to the light intensity seen by adjacent fireflies
- **Cuckoo Search**: based on the brood parasitism of some cuckoo species
- **Monkey Search**: inspired by the behavior of a monkey climbing trees looking for food
- **Harmony Search**: inspired by the improvisation process of musicians
- ...
More complex Metaheuristics
More complex Metaheuristics

*Glover and Laguna 1997* [7]:

- Scatter Search
- Path Relinking
Scatter Search

Elements of the method

- **Diversification Generation Method**: method to generate a configuration
- **Improvement Method**: improve a solution (ex: LS, TS, ...)
- **Reference Set Update Method**: build and maintain a set of the $b$ best configurations
- **Subset Generation Method**: create sets of references
- **Solution Combination Method**: transform a set into one or more solutions (crossover)
Scatter Search

Algorithm - Generation of a population

build a population $P$

1. use **Diversification Generation Method** to generate $x$
2. apply **Improvement Method** to obtain $x^*$ and add to $P$
3. repeat until $|P| = P_{size}$
4. use **Reference Set Update Method** to build a $RefSet$ a set of $b$ best configurations
5. order configuration in $RefSet$ such that $x_1$ is the best and $x_b$ is the worst
Algorithm - Improvement of the population

1. use **Subset Generation Method** to generate *NewSubsets* from *RefSet*
2. for each $s \in *NewSubsets*$:
3. apply **Solution Combination Method** to obtain configuration $x$
4. apply **Improvement Method** to obtain $x^*$ and add to $P$
5. add $x^*$ to *RefSet* and remove $x_b$ if $x^*$ is better than $x_b$
6. repeat step 1 if new configuration were added to *RefSet*
Metaheuristics for bioinformatics

More complex Metaheuristics

Scatter Search

Algorithm

1. an improved Genetic Algorithm
2. create *weighted centers of selected sub-regions*
3. close to bees algorithm
4. $|P| \geq 10 \times |\text{RefSet}|$
Path Relinking

Elements of the method

- integrate intensification and diversification in Tabu Search
- close to Scatter Search
- generate new configuration by exploring trajectories that connect high-quality configurations:

\[ x' = x(1), x(2), \ldots, x(r) = x'' \]
Path Relinking

Algorithm

same as Scatter Search:

1. NewSubsets consists of all the pairs of solutions in RefSet
2. for each pair \((x', x'')\):
3. apply Relinking Method to produce sequence path to \(x' \rightarrow x''\)
4. apply Improvement Method to part of the sequence
5. do steps 3 and 4 for \(x'' \rightarrow x'\)
6. add new solutions if better to RefSet
Hyper-heuristics
Hyper-heuristics (from wikipedia)

- Origin of the idea (although not the term) can be traced back to the early 1960s
- The term **hyper-heuristics** was first coined in 1997 by Jörg Denzinger, Matthias Fuchs and Marc Fuchs
- To describe a protocol that chooses and combines several AI methods
- In 2000, Cowling and Soubeiga used it to describe the idea of *heuristics to choose heuristics*
Hyper-heuristics

- Metaheuristics search within a search space of problem solutions
- hyper-heuristics search within a search space of heuristics
  - multiple heuristics from which one can choose for solving a problem: **strength and weakness** (*No free lunch*)
  - automatically devise algorithms by combining the strength and compensating for the weakness of known heuristics
Hyper-heuristics

Two main categories

- **heuristics to choose heuristics**: discover a good sequence of applications of heuristics
- **heuristics to generate heuristics**: evolve new heuristics by making use of the components of known heuristics
- possibly: use of feedback during the learning process
No free lunch theorem \sim postulate

Definition (No free lunch [2])
In computing, there are circumstances in which the outputs of all procedures solving a particular type of problems are statistically identical.

In other words
In general, there is no method always better than the others on all types of problems, it depends on the problem instance.
Reminder and perspectives

Characteristics of Metaheuristics

- Metaheuristics are **not problem-specific**
- the basic concepts of metaheuristics permits an **abstract level description**

Perspectives

Metaheuristics are well-suited to resolution of bioinformatics problems:

- **seldom need the optimal solution**: they require fast and near-optimal solutions
- **data of bioinformatics inherently involve errors**: Metaheuristics *can cope with errors* due to their non-deterministics process
## Complexities of some Metaheuristics

<table>
<thead>
<tr>
<th>Metaheuristic</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill-Climbing, Tabu Search, ILS, SA</td>
<td>$n$</td>
</tr>
<tr>
<td>Random Restart</td>
<td>$r \times n$</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>$k$</td>
</tr>
<tr>
<td>hybrid / memetic methods</td>
<td>$k \times n$</td>
</tr>
<tr>
<td>ACO, Bees</td>
<td>$p \times n$</td>
</tr>
<tr>
<td>Scatter Search</td>
<td>$PSize \times n + n_L \times \frac{RefSize^2}{2} \times n$</td>
</tr>
</tbody>
</table>

$n$ number of iterations, $r$ number of restarts, $k$ number of crossovers, $p$ size of population, $n_L$ number of loops
Questions and answers
Bibliography I


