Metaheuristics: course introduction

Fernando Lobo

Universidade do Algarve

What are metaheuristics?

- A class of algorithms for solving optimization problems.
- They are used when little information is available to help to solve a problem.
- And when a brute-force method is not adequate.

A strange name

- Metaheuristics: not a particularly good name.
- Sounds like heuristics about heuristics, but that's not what they are!
- A heuristic is problem specific.
  - Example: Manhattan distance for a sliding-tile puzzle (used in the A* algorithm, a classic in AI)
- Metaheuristics are often not too specific: they are applicable to a wide range of problems
- Related names: *Stochastic Optimization*, *Stochastic (Local) Search*

Learning outcomes

- You will learn the working principles of these algorithms
- How to implement and apply them to a particular domain
- How to assess the algorithm's performance

Major topics

- Foundations
- Single-State Methods
- Empirical Analysis
- Search Space Structure and Performance
- Population Methods
- Theory of Randomized Search Heuristics

(More detailed information given in Ficha Unidade Curricular available at he UAlg's tutoria)

Prior knowledge and skills

I assume you have (at least a basic) knowledge on:

- Computer Programming
- Algorithms and Data Structures
- Statistics and Probability
- Discrete Mathematics

Grading policy

- Lab assignments (50%), with minimum grade requirement of 9,5 to be admitted to the exam.
- Exam (50%), with minimum grade requirement of 8,5 to be able to pass.
- Final grade is weighted average of Lab assignments and Exam, and must be at least 9.5.

Grading policy: lab assignments

- The lab assignments are done individually or in a group of a maximum of 2 students.
- The grade of the lab assignments of the group is converted into an individual grade at the time of the work validation, which is mandatory.
- The final grade of the assignments is therefore individual, and depends on the performance of each member of the group during the validation.

Our classes: T and TP

- I will present the material and give illustrative examples during the T lectures.
- In the TP lectures you will have hands-on experience on the course materials and work on lab assignments.

Main bibliography

- Stochastic Local Search: Foundations and Applications, by H. Hoos and T. Stützle. Morgan Kaufmann, 2004.
- Essentials of Metaheuristics, 2nd edition (free at `http://www.cs.gmu.edu/~sean/book/metaheuristics/`), by Sean Luke, 2016.
- Introduction to Evolutionary Computing, by A. E. Eiben and J. E. Smith, Springer, 2003.

Some illustrative problems

- Evolving Mona Lisa
- Travelling Salesman Problem
- Magic Square
- Boolean Satisfiability Problem
- Darwin Tunes

Evolving Mona Lisa

- Obtain an image made out of a number of overlapping polygons of various colors and transparencies, that resembles as much as possible to the Mona Lisa painting.

- Let's check a demo at `https://alteredqualia.com/visualization/evolve/`

Evolving Mona Lisa

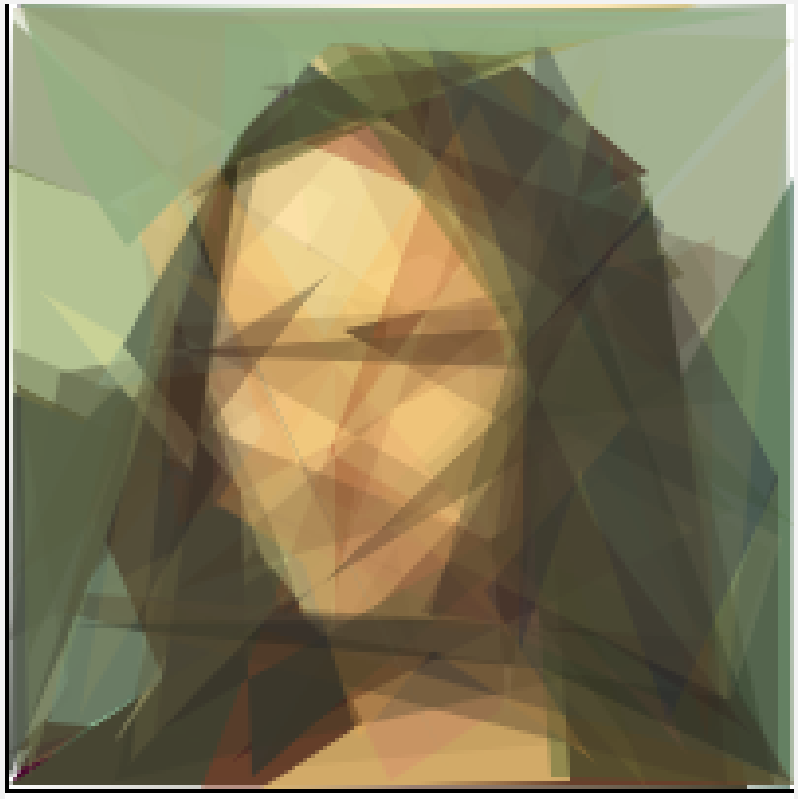
Basic idea:

- Start with a fixed number of polygons (say 50), each with a fixed number of vertices (say 6).
- Each polygon has a color represented by RGB values, and a transparency level $0 \leq \alpha \leq 1$.
  - Alpha blending from Computer Graphics: 0 means pixel if fully transparent, 1 means pixel is fully opaque.
- Algorithm starts by creating an image composed of randomly generated polygons.
- It then assesses how close the resulting image is to the Mona Lisa image by doing a pixel by pixel comparison between the two images.

Evolving Mona Lisa

- The algorithm then makes a slight change to the randomly generated image, such as:
  - changing the color on one of its polygons
  - changing the transparency level on one of its polygons
  - changing the x- or y-coordinate of one of its polygons
  - The algorithm then keeps the better of the two images
- . . . and keeps iterating like this over and over
- This is known as a (1+1)-Evolutionary Algorithm.

Evolving Mona Lisa

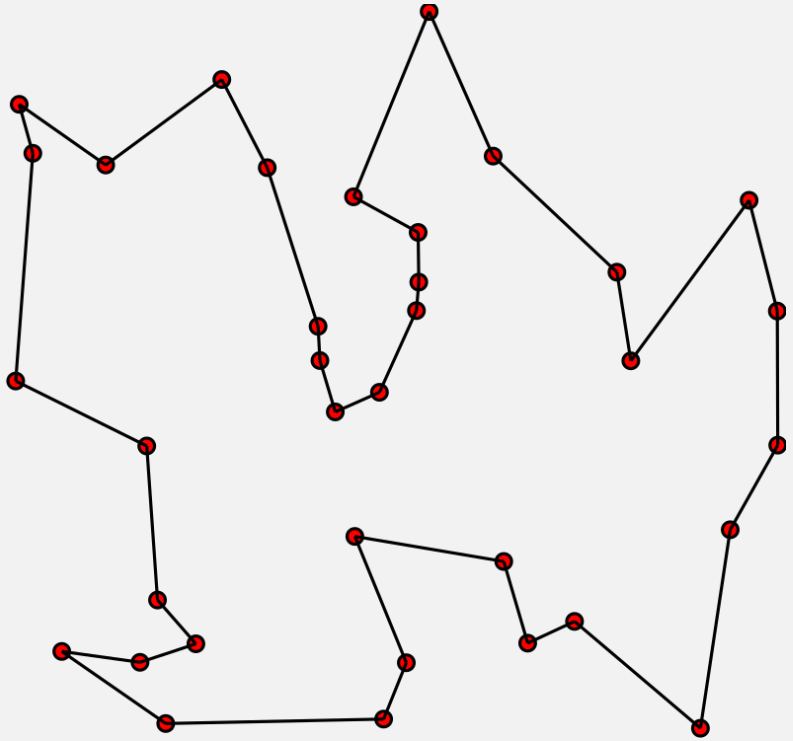


- Left image is the original. The image on the right is composed of 50 polygons, each with 6 vertices, and was obtained after 100 thousand iterations of the algorithm. (About 10 minutes in my old slow laptop.)

Travelling Salesman Problem (TSP)

- A classical combinatorial optimization problem.
- Given a set of n cities and and the distances between each pair of cities, find the shortest route that visits each city exactly once and returns to the starting city.

TSP



TSP

- Completely different problem compared to the Mona Lisa example
- But can use a similar solving strategy
  - start with a random solution, and calculate the distance
  - make a slight change to the solution, and calculate the new distance
  - keep the better of the two solutions
  - iterate the previous steps over and over

TSP

- There are many approaches to solve the TSP
- Some use problem specific heuristics tailored to the problem
- Others use more general methods
- Some online demos available at: `https://tspvis.com/`

Brute-force approach

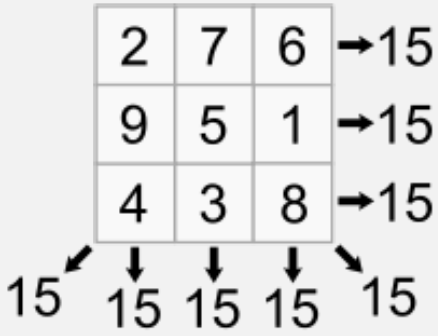
- We could try solving the TSP using an exhaustive search
  - How? testing every possible tour, and choosing the best one
- How much effort would that take?
  - For a set of n cities there are $n!$ orderings of those cities.
  - Some of those orderings correspond to the same tour, but we still have $O(n!)$ distinct tours.
  - Can only solve in an exhaustive manner for small values of n.
  - For larger values we need *approximate* algorithms, or metaheuristics

Metaheuristics

- Do not guarantee an optimal solution
- But tend to give a reasonably good solution, quickly

Magic square

- A magic square of order n is a $n \times n$ array of distinct positive integers $1, 2, \ldots, n^2$, placed in such way that the sum of the numbers in every row, every column, and both diagonals, equals the same value (called the *magic constant* or *magic sum*).



- It can be shown that there is a magic square of order n, for every $n \geq 3$.

- Magic sum $= (1 + 2 + \ldots + n^2)/n = \frac{(1+n^2)n}{2}$

A magic square of order 8 (magic sum $= 260$)

8	58	59	5	4	62	63	1
49	15	14	52	53	11	10	56
41	23	22	44	45	19	18	48
32	34	35	29	28	38	39	25
40	26	27	37	36	30	31	33
17	47	46	20	21	43	42	24
9	55	54	12	13	51	50	16
64	2	3	61	60	6	7	57

Solving a magic square with a simple metaheuristic

- Again, can use a similar solving strategy used for the Mona Lisa or TSP.
  - start with a random solution, and calculate “how close” it is to a magic square
  - make a slight change to the solution, and evaluate it
  - keep the better of the two solutions
  - iterate the previous steps over and over

How to measure “how close”?

- We know the magic sum for a magic square of order n.
- Given a square of numbers, we can add the numbers in each row, column, and diagonal.
- Record the difference between the obtained sum and the magic sum.
- Add up all those differences (or some function of those differences. A common strategy is to sum the square of the differences.)
- This gives us a distance measure to assess how close we are to a magic square
  - For a magic square the sum of the differences is zero.

Boolean satisfiability problem (SAT)

- Another classical problem in computer science
- Consists of determining if a boolean formula can be satisfied (i.e., can be evaluated to True)
- Note: every boolean formula can be converted to the conjunctive normal form (CNF)
  - conjunction of one or more clauses
  - where each clause is a disjunction of literals, and a literal is a variable or its negation

SAT: example

$$
\begin{aligned}
F = & (\neg x_1 \vee x_2) \\
& \wedge (\neg x_2 \vee x_1) \\
& \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\
& \wedge (x_1 \vee x_2) \\
& \wedge (\neg x_4 \vee x_3) \\
& \wedge (\neg x_5 \vee x_3)
\end{aligned}
$$

- Is F satisfiable?
- In other words, is there an assignment to x_1, x_2, x_3, x_4, x_5, that makes F True?
- Yes: $x_1 = \mathit{True}$, $x_2 = \mathit{True}$, $x_3 = \mathit{False}$, $x_4 = \mathit{False}$, $x_5 = \mathit{False}$.

MAXSAT

- MAXSAT is an optimization version of SAT
- Goal of MAXSAT is to maximize the number of satisfied clauses
- A brute-force method would evaluate all possible variable assignments
  - for n variables we have 2^n possible assignments
- Can use a metaheuristic

Darwin Tunes: evolving music

- We can represent music as a sequence of numbers (everything is bits!)
- Start by generating random pieces of music
  - Should sound terrible!
- Let people judge the quality of each piece on a relative scale
  - say 1–5, with 1 the lowest and 5 the highest quality
- An algorithm can then pick the pieces with better evaluations, and make variations of them
- And this process is iterated over and over
- Check this out: `http://darwintunes.org/audio-snapshots.html`

Search and Optimization

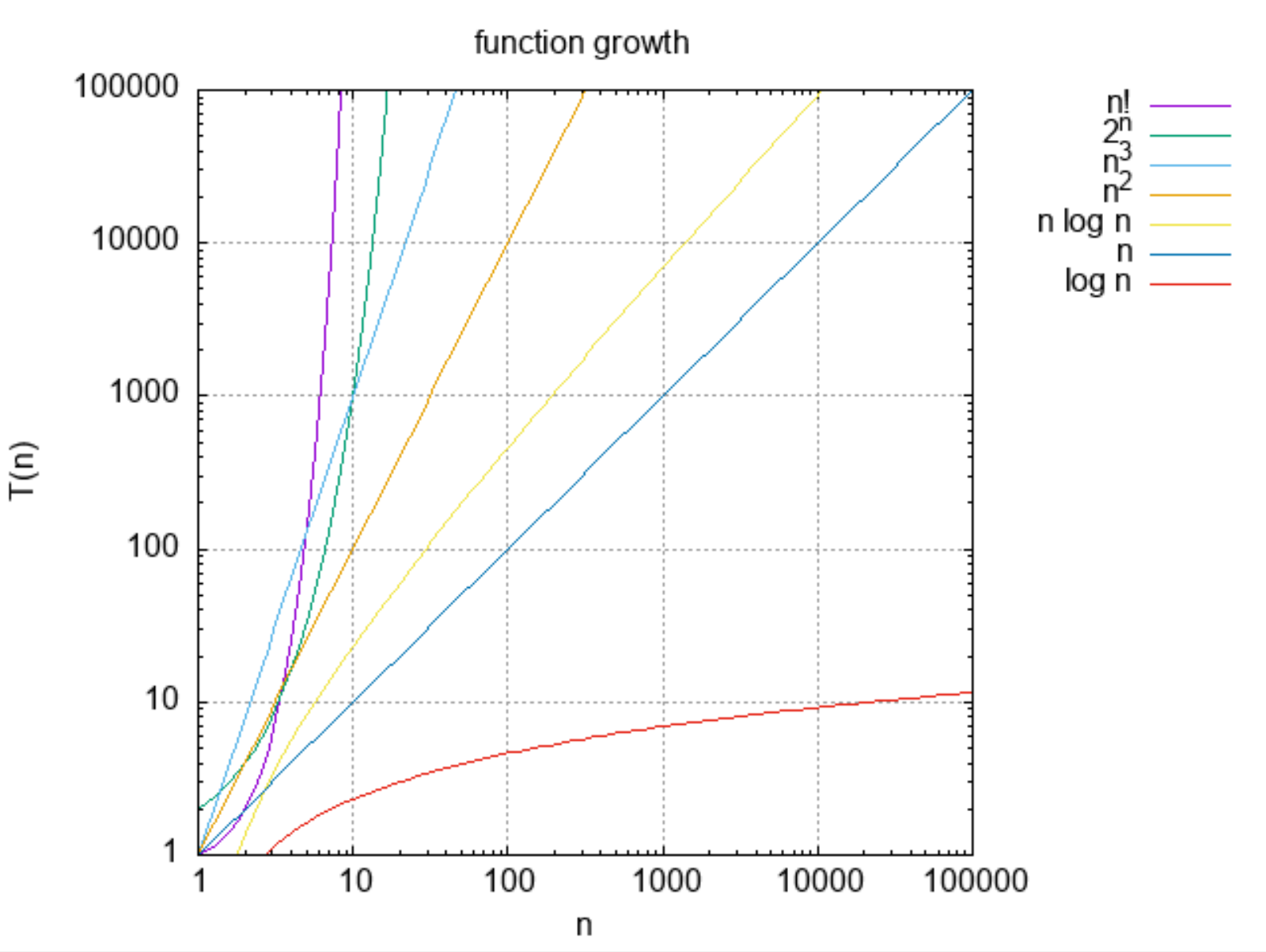
- We sometimes refer to these problems as *search problems*
- Other times as *optimization problems*
- They are really two sides of the same coin

Time Complexity

- Gives the amount of time taken by and algorithm as a function of the input size
- Described by the O, Θ, and Ω notation
- Constant terms are ignored
  - It doesn't mean they are not important . . .
- An algorithm runs in polynomial time if its time complexity if bounded by a polynomial function of the input size
- An algorithm runs in exponential time if its time complexity cannot be bounded by a polynomial function of the input size

Time Complexity

Time complexity	Problem
$\Theta(1)$	Hashing
$\Theta(\lg n)$	Binary search
$\Theta(n)$	Linear search
$\Theta(n \lg n)$	MergeSort, QuickSort
$\Theta(n^2)$	InsertionSort
$\Theta(n^3)$	Traditional matrix multiplication
$\Theta(a^n)$, $a > 1$	SAT
$\Theta(n!)$	TSP


function growth
100000
10000
1000
100
10
1
T(n)
1
10
100
1000
10000
100000
n
n!
2^n
n^3
n^2
n log n
n
log n

Metaheuristics and problem complexity

- Problems that can be solved in polynomial time are said to be tractable. The others are intractable.
- Metaheuristics are appropriate for intractable problems