Heuristic Algorithms

Master's Degree in Computer Science/Mathematics

Roberto Cordone

DI - Università degli Studi di Milano

Schedule:	Thursday 14.30 - 16.30 on MS-Teams
	Friday 14.30 - 16.30 on MS-Teams
Office hours:	on appointment
E-mail:	roberto.cordone@unimi.it
Web page:	https://homes.di.unimi.it/cordone/courses/2020-ae/2020-ae.html
Ariel site:	https://rcordoneha.ariel.ctu.unimi.it

Aims of the course

This course aims to

1. show that heuristic algorithms are not recipes for specific problems: heuristics and problems can be matched freely
   (*of course, with different performance*)
2. discuss the common and general aspects of these algorithms
3. teach how to design a heuristic for a specific problem
4. teach how to evaluate its performance

Heuristics

eurisko = *I find*

It is a word derived from Greek

- inspired by the famous story of Archimedes and the golden crown

but it was

- never used by the ancient Greeks
- coined during the 19th century

Some historical facts

- 4th century CE: Pappus of Alexandria discusses the *analyòmenos* (*treasure of analysis*), that is how to build a mathematical proof
    - how to move from the hypotheses to the thesis of a theorem
    - how to move from the data to the solution of a geometrical problem
- 17th century: Descartes, Leibnitz *et al.* discuss the ars inveniendi (*art of finding*), *i. e.* the attainment of truth through mathematics
- 19th century: Bernard Bolzano discusses in detail the most common strategies to build mathematical proofs (*Erfindungskunst*)
- 19th-20th century: phylosophers, psychologists and economists define heuristics as practical and simple decision rules that do not aim at an optimal result, but at a satisficing one (Simon, 1957)
- 1945: the short essay *How to solve it* by György Pólya comes back to the mathematical meaning of heuristic as an informal process that leads to prove a thesis or to find a solution

So, what about *heuristic algorithms*?

Algorithms and heuristics

Some scientific sectors use the two words as opposites:

- algorithm as a formal, deterministic procedure, consisting of a finite sequence of elementary steps
- heuristic as an informal, creative, open rule

One could even say that

- an algorithm is a correctness proof
- a heuristic is a bunch of common sense arguments

In fact, an algorithm has a correctness proof, a heuristic has none

The phrase heuristic algorithm is an oxymoron, in some respects

Then what does it mean?

Heuristic algorithms

A heuristic algorithm is an algorithm which does not guarantee a correct solution

Then it is useless!

Quite to the contrary, it can be useful, provided that

1. it “costs” much less than a correct algorithm:
   this requires a definition of computational cost of an algorithm
   - time
   - space
2. it “frequently” yields something “close” to the correct solution:
   this requires to define a solution space endowed with
   - a metric to express a “satisfactory distance” from the correct solution
   - a probabilistic distribution to express the “satisfactory frequence” of solutions at a satisfactory distance from the correct solutions

Proofs and algorithms

Mathematical proofs and algorithms are strictly related

- every algorithm has/is a correctness proof
- both are mechanical symbolic transformations from a starting point (hypotheses/data) to an ending point (thesis/solution)
- Turing's undecidability proof mirrors Gödel's incompleteness proof

Heuristics are the construction of both proofs and algorithms

- in case of success, the heuristic is abandoned and the proof preserved
- otherwise, a good heuristic frequently provides a good result, instead of always providing a perfect one

This is the motivation for heuristic algorithms

The focus of this course

The course focuses on heuristic algorithms

- that apply to Combinatorial Optimization problems
- that are solution-based (as opposed to model-based)

So, we limit

1. the kind of problem
2. the kind of algorithm

It is still a pretty wide field

Let us further discuss the two limitations

Problem classification

A problem is a question on a mathematical system

Problems can be classified based on the nature of their solution:

- decision problems: their solution is either *True* or *False*
- search problems: their solution is any feasible subsystem (that is, satisfying certain conditions)
- optimization problems: their solution is the minimum or maximum value of an objective function defined on the feasible subsystems
- counting problems: their solution is the number of feasible subsystems
- enumeration problem: their solution is the collection of all feasible subsystems

We focus on the combination of optimization and search, that is, we look for the optimal value and a subsystem assuming that value

Optimization/search problems

An optimization/search problem can be represented as

$$\begin{array}{l} \text{opt } f(x) \\ x \in X \end{array}$$

where

- a solution x describes each subsystem of the problem
- the feasible region X (feasible solution space) is the set of subsystems which satisfy given conditions
- the objective function $f : X \to \mathbb{R}$ quantitatively measures the quality of each subsystem ($\text{opt} \in \{\min, \max\}$)

The problem consists in determining

- optimization: the optimal value f^* of the objective function:

$$f^* = \underset{x \in X}{\text{opt}}\, f(x)$$

- search: at least one optimal solution, that is a subsystem

$$x^* \in X^* = \arg \underset{x \in X}{\text{opt}}\, f(x) = \left\{ x^* \in X : f(x^*) = \underset{x \in X}{\text{opt}}\, f(x) \right\}$$

Why optimization/search problem?

Several application fields require objects or structures characterized by very high or very low values of a suitable evaluation function

- *bioinformatics*: the most effective drugs bond with proteins in configurations of minimal potential energy
- *social networks*: the best target for a campaign are the most influentiable, most influential and most uncorrelated groups of individuals
- *machine learning*: the most effective classification systems generate the simplest classifications and the minimum amount of violations
- *hardware design*: the best logical circuits require the minimum space and yield the minimum delay
- *parameter estimation*: the best physical models are the ones which reproduce the observations with the minimum error
- *finance*: the most effective portfoglio management algorithms reproduce the target time series in the most precise way

Exact optimization is costly and not always required, or even desirable: heuristic algorithms could be preferable

Why optimization/search problem?

Other problems can often be reduced to optimization/search problems

- search problems can be reduced by
  - relaxing the conditions to satisfy, so as to enlarge the feasible region from X to $X' \supset X$ and obtain an easy search problem;
  - introducing a function $d(x)$ to quantify the distance of each $x \in X'$ from X;
  - minimizing $d(x)$ to find x^* such that $d(x^*) = 0 \Leftrightarrow x^* \in X$.
- some decision problems concern the existence of feasible subsystems, and are equivalent to search problems

  (*finding the subsystem proves that it exists*)
- some enumeration problems concern the search for subsystems with "good" values of conflicting objective functions (Pareto frontier) and allow direct adaptations of optimization/search algorithms

Such reductions are often possible and useful, though not always

Combinatorial Optimization (*CO*)

A problem is a *CO* problem when the feasible region X is a finite set, that is, it has a finite number of feasible solutions

This looks like a very restrictive assumption

However, the study of *CO* problems can be useful more in general:

1. infinite discrete problems can have a finite set of interesting solutions
2. some continuous problems can be reduced to *CO* problems (*e. g., Linear Programming, Maximum Flow, Minimum Cost Flow*)
3. continuous problems can be reduced to discrete ones by sampling (*usually not very effective*)
4. ideas conceived for *CO* problems can be extended to other problems (*often quite effective*)

Model-based heuristics

They describe the feasible region X with a “model”

A typical example is a Mathematical Programming formulation

$$\begin{array}{l} \text{opt } f(x) \\ x \in X \end{array} \quad \longrightarrow \quad \begin{array}{ll} \min \ \phi(\xi) & \\ g_i(\xi) \leq 0 & i = 1, \dots, m \end{array}$$

where

- $\xi \in \mathbb{R}^n$, that is, a solution is a vector of n real values
- $X = \{\xi \in \mathbb{R}^n : g_i(\xi) \leq 0, i = 1, \dots, m\}$, that is, the feasible region is the set of vectors which satisfy all the inequalities (constraints)

Model-based heuristics exploit the information derived from the model, that is the analytical properties of functions ϕ and g_i $(i = 1, \dots, m)$

Other models can be based on *SAT*, etc...

We will not use these tools

An alternative definition of *CO*

A problem is a *CO* problem when:

1. the number of feasible solutions is finite
2. the feasible region is $X \subseteq 2^B$ for a given finite ground set B, that is, the feasible solutions are all subsets of the ground set that satisfy suitable conditions

The two definitions are equivalent:

$2 \Rightarrow 1$: if the ground set B is finite, every collection $X \subseteq 2^B$ is finite

$1 \Rightarrow 2$: if the number of feasible solutions is finite, define B as their set and the feasible region X as the collection of all singletons of B
(*a "solution" is a set containing a single solution*)

In general, the sophisticated definition allows a deeper analysis, because

- X is not simply enumerated
- X is defined in a compact and significant way

Solution-based heuristics: a classification for *CO* problems

Solution-based heuristics consider solutions as subsets of the ground set

1. constructive/destructive heuristics:
   - they start from an extremely simple subset (respectively, $\emptyset$ or B)
   - they add/remove elements until they obtain the desired solution
2. exchange heuristics:
   - they start from a subset obtained in any way
   - they exchange elements until they obtain the desired solution
3. recombination heuristics:
   - they start from a population of subsets obtained in any way
   - they recombine different subsets producing a new population

Heuristic designers can creatively combine elements from different classes

Randomization and memory

Two other distinctions concern

- the use of randomization:
  - purely deterministic heuristics
  - randomized heuristics, that are deterministic algorithms whose input includes pseudorandom numbers
- the use of memory:
  - heuristics whose input includes only the problem data
  - heuristics whose input also includes previously generated solutions

These distinctions are independent from the previous classification

Metaheuristics (from the Greek, “beyond heuristics”) is the common name for heuristic algorithms with randomization and/or memory

Risks to beware of

1. reverential or trendy attitude, that is choosing an algorithm based on the social context, instead of the problem
2. magic attitude, that is trusting a method on the basis of an analogy with physical and natural phenomena
3. heuristic integralism, that is using a heuristic for a problem which admits exact algorithms
4. number crunching, that is performing sophisticated and complex computations with unreliable numbers
5. SUV attitude, that is relying on hardware power
6. overcomplication, that is introducing redundant components and parameters, as if that could only improve the result
7. overfitting, that is adapting components and parameters of the algorithm to the specific dataset used in the experimental evaluation

It is fundamental to

- free oneself from prejudices
- evaluate the performance of the algorithm in a scientific way
- distinguish the contribution of each component of the algorithm
- efficiently implement each component of the algorithm

The *Analytics and Optimization* track

This course belongs to the *Analytics and Optimization* track:

- this gives a specific slant to the presentation of the subject
- but the course can be easily attended by students of other tracks

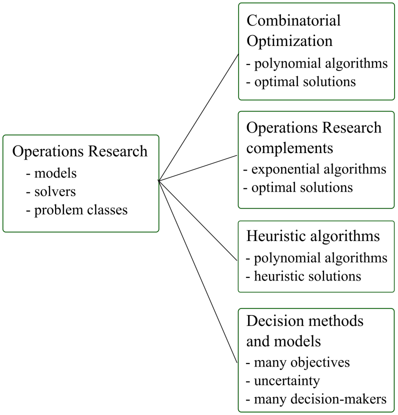
The focus of the track is on practical decisions where

- a large amount of data must be taken into account
- the possible choices are many
- the costs of a wrong choice are high

The correct strategy is first make a model, then compute, finally decide and nowadays this is supported by

- Big Data: huge amounts of precise, structured and cheap data, from which to extract information
- Cloud Computing: pervasive capacity to access and process data
- Business Analytics: a business culture open to the use of models
- new theory: online, stochastic, robust programming, etc. . .

The *Analytics and Optimization* track



But we have already seen that

- heuristic algorithms are useful for any application
- we do not require models

The course is open to any student with these prerequisites

- C programming (laboratory)
- Algorithms and data structures (preferential)