Heuristic Algorithms

Master's Degree in Computer Science/Mathematics

Roberto Cordone

DI - Università degli Studi di Milano

Schedule:	Wednesday 13.30 - 16.30 in classroom Alfa
	Thursday 09.30 - 12.30 in classroom Alfa
Office hours:	on appointment
E-mail:	roberto.cordone@unimi.it
Web page:	https://homes.di.unimi.it/cordone/courses/2026-ae/2026-ae.html
Ariel site:	https://myariel.unimi.it/course/view.php?id=7439

Evaluation of a heuristic algorithm

The performance of a heuristic algorithm can be investigated by

- theoretical analysis (*a priori*): proving a theoretical guarantee on the computational cost or the quality, always or with a given frequency
- experimental analysis (*a posteriori*): measuring the empirical performance of the algorithm on a sample of benchmark instances

The theoretical analysis is complicated by the fact that

- the steps of the algorithm have a complex effect on the solution though usually not on the computational cost
- average case and randomisation require a statistical treatment

The theoretical analysis can be unsatisfactory in practice when its conclusions are based on unrepresentative assumptions

- an infrequent worst case (very hard and very rare instances)
- an unrealistic probability distribution of the instances

This lesson is partly based on slides provided with the book "*Stochastic Local Search*" by H. H. Hoos and T. Stützle, (Morgan Kaufmann, 2004) - see www.sls-book.net for further information.

Experimental analysis

The experimental analysis is a fundamental tool of science, but

- mathematics is an exception, as it is based on the formal approach
- algorithmics is an exception within the exception

The purpose of the analysis is to investigate

- in physics the laws that rule the behaviour of phenomena
- in algorithmics the laws that rule the behaviour of algorithms

The experimental analysis of algorithms aims to

1. obtain compact indices of efficiency and effectiveness of an algorithm
2. compare the indices of different algorithms to rank them
3. describe the relation between the performance indices and parametric values of the instances (size n, etc. . .)
4. suggest improvements to the algorithms

The experimental approach

The basics of the experimental approach are

1. start from observation
2. formulate a model (work hypothesis)
3. repeat the following steps
   a. design computational experiments to validate the model
   b. perform the experiments and collect their results
   c. analyse the results with quantitative methods
   d. revise the model based on the results

   until a satisfactory model is obtained

What is a "model" in algorithmics?

Its two basic aspects are the problem and the algorithm

Simulation models

Empirical studies first build a simulation model of the problem

- assume a probability distribution on $\mathcal{I}_n$ for each $n \in \mathbb{N}$, as in theoretical studies, but possibly based on empirical reasons (e.g., drawn from real-world data),
- build a benchmark of random instances following the distribution (*not all instances can be tested!*)
- apply the algorithm and measure the time spent and quality achieved
- give a statistical description of the measures obtained

We have already discussed several models used in theoretical analysis (*see lesson 3*)

Benchmark

A meaningful benchmark sample must represent different

- sizes, in particular for the analysis of the computational cost
- structural features (for graphs: density, degree, diameter, ...)
- types
  - of application: logistics, telecommunications, production, ...
  - of generation: realistic, artificial, transformations of other problems
  - of probabilistic distribution: uniform, normal, exponential, ...

Looking for an "equiprobable" benchmark sample is meaningless because

- the instance sets are infinite
- infinite sets do not admit equiprobability (*it's a big statistic question*)

On the contrary, we can

- define finite classes of instances that are
  - sufficiently hard to be instructive
  - sufficiently frequent in applications to be of interest
  - quick enough to solve to provide sufficient data for inferences
- extract benchmark samples from these classes

Reproducibility

The scientific method requires reproducible and controllable results

- concerning the instances, one must use
  - publicly available instances
  - new instances made available to the community
- concerning the algorithm, one must specify
  - all implementation details
  - the programming language
  - the compiler
- concerning the environment, one must specify
  - the machine used
  - the operating system
  - the available memory
  - ...

Reproducing results obtained by others is anyway extremely difficult

Behavioural models of algorithm performance

After setting the simulation model of the problem,
empirical studies build a behavioural model of the investigated algorithm

We model the execution of algorithm A as a random experiment

- the whole set of instances $\mathcal{I}$ is the sample space
- the benchmark subset of instances $\bar{\mathcal{I}} \subset \mathcal{I}$ is the sample
- the computational time $T_A(I)$ is a random variable
- the relative difference $\delta_A(I)$ is a random variable

We describe the performance of A with the statistical properties
of the random variables $T_A(I)$ and $\delta_A(I)$

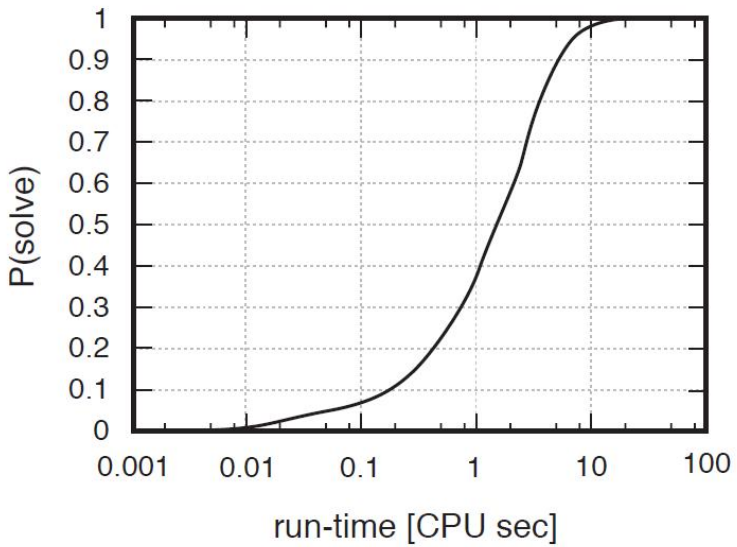
Let us start with the time $T_A(I)$

Analysis of the computational time (*RTD* diagram)

The *Run Time Distribution* (*RTD*) diagram is the plot of the distribution function of $T_A(I)$ on $\bar{\mathcal{I}}$

$$F_{T_A}(t) = Pr[T_A(I) \leq t] \text{ for each } t \in \mathbb{R}$$

Since $T_A(I)$ strongly depends on the size $n(I)$,
meaningful *RTD* diagrams usually refer to benchmarks $\bar{\mathcal{I}}_n$ with fixed n
(and possibly other fixed parameters suggested by the worst-case analysis)

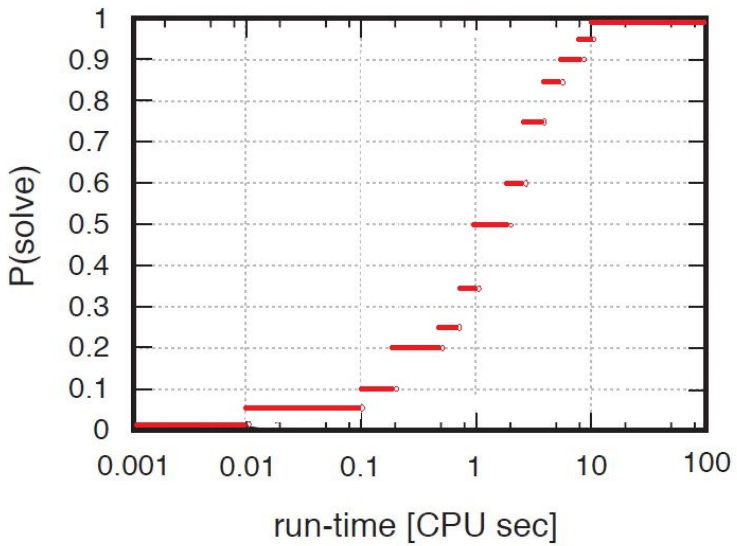


If all influential parameters are identified and fixed, the *RTD* diagram degenerates into a step function (*all instances require the same time*)

The Run Time Distribution (RTD) diagram

The Run Time Distribution (RTD) diagram is

- monotone nondecreasing: more instances are solved in longer times
- stepwise and right-continuous: the graph steps up at each $T(I)$
- equal to zero for $t < 0$: no instance is solved in negative time
- equal to 1 for $t \geq \max_{I \in \bar{\mathcal{I}}} T(I)$: all are solved within the longest time



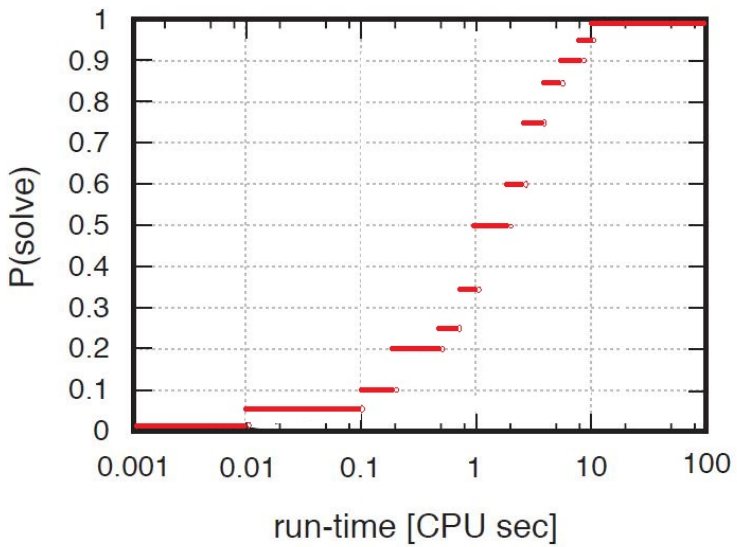
For large benchmark samples, the plot looks continuous, but it is not!

(*as in the previous page*)

Building the *RTD* diagram

In order to build the diagram

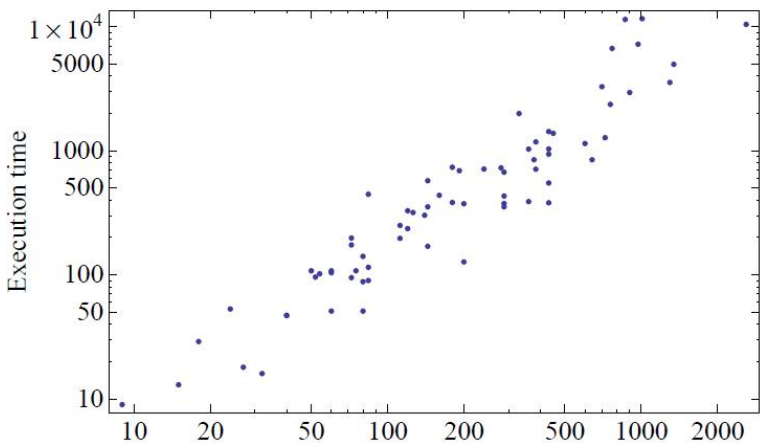
1. run the algorithm on each instance $I \in \bar{\mathcal{I}}$
2. build the set $T_A\left(\bar{\mathcal{I}}\right) = \left\{T_A\left(I\right) : I \in \bar{\mathcal{I}}\right\}$
3. sort $T_A\left(\bar{\mathcal{I}}\right)$ by nondecreasing values: $t_1 \leq \ldots \leq t_{|\bar{\mathcal{I}}|}$
4. plot points $\left(t_j, \frac{j}{|\bar{\mathcal{I}}|}\right)$ for $j = 1, \ldots, |\bar{\mathcal{I}}|$ (for equal t_j, the highest j) and the horizontal segments (close on the left, open on the right)



Analysis of the computational time (*scaling* diagram)

The scaling diagram describes the dependence of $T(I)$ on the size $n(I)$

- generate a sequence of values of n and a sample $\bar{\mathcal{I}}_n$ for each value
- apply the algorithm to each $I \in \bar{\mathcal{I}}_n$ for all n
- sketch all points $(n(I), T(I))$ or the mean points $\left(n, \dfrac{\sum_{I \in \bar{\mathcal{I}}_n} T(I)}{|\bar{\mathcal{I}}_n|} \right)$
- assume an interpolating function (as discussed later)
- estimate the numerical parameters of the interpolating function



This analysis provides an empirical average-case complexity

- with well-determined multiplying factors (*instead of* c_1 *and* c_2)
- not larger than the worst-case one (*it includes also easy instances*)

Interpolation of the *scaling* diagram

The correct family of interpolating functions can be suggested

- by a theoretical analysis
- by graphical manipulations

Linear interpolation is usually the right tool

The scaling diagram turns into a straight line when

- an exponential algorithm is represented on a semilogarithmic scale (*the logarithm is applied only to the time axis*)

$$\log_2 T(n) = \alpha n + \beta \Leftrightarrow T(n) = 2^{\beta} (2^{\alpha})^n$$

- a polynomial algorithm is represented on a logarithmic scale (*the logarithm is applied to both axes*)

$$\log_2 T(n) = \alpha \log_2 n + \beta \Leftrightarrow T(n) = 2^{\beta} n^{\alpha}$$

Phase transitions

The performance of an algorithm with respect to

- the computational time (for exact and heuristic algorithms)
- the quality of the solution (for heuristic algorithms)

can depend strongly on other parameters apart from size

Different values of the these parameters correspond to different regions of the instance set; for example, for graphs:

- $m = 0$ and $p = 0$ correspond to empty graphs
- $m = \frac{n(n-1)}{2}$ and $p = 1$ correspond to complete graphs
- intermediate values correspond to graphs of intermediate density (deterministically for m, probabilistically for p)

Often, the performance variation takes place abruptly in small regions of the parameter space, as in phase transitions of physical systems

Knowing it allows to predict the behaviour of an algorithm on an instance

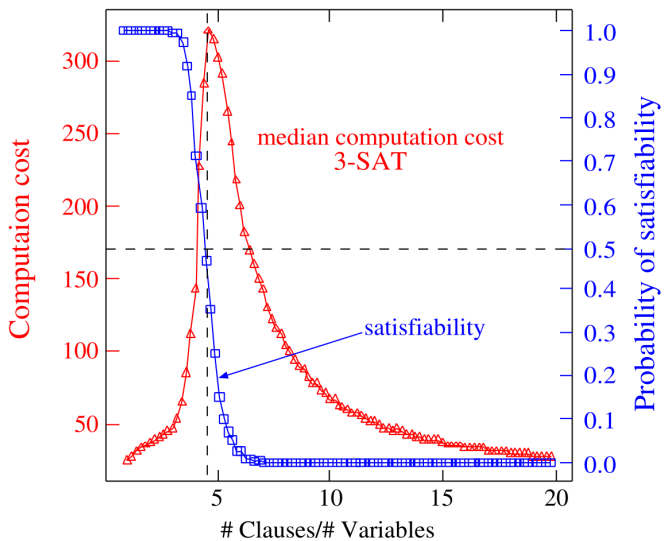
Phase transitions for 3-*SAT* and *Max*-3-*SAT*

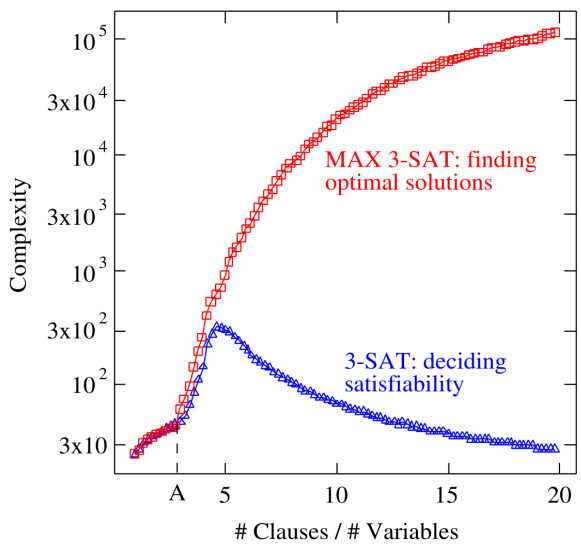
Given a *CNF* on n variables, with logic clauses containing 3 literals

- 3-*SAT*: is there a truth assignment satisfying all clauses?
- *Max*-3-*SAT*: what is the maximum number of satisfiable clauses?

As the clauses/variables ratio, $\alpha = m/n$ increases

- satisfiable instances decrease from nearly all (many variables for few clauses) to nearly none (few variables for many clauses)
- the computing time first sharply increases, then decreases for *SAT*, increases further for *Max-SAT* (*using a well-known exact algorithm*)



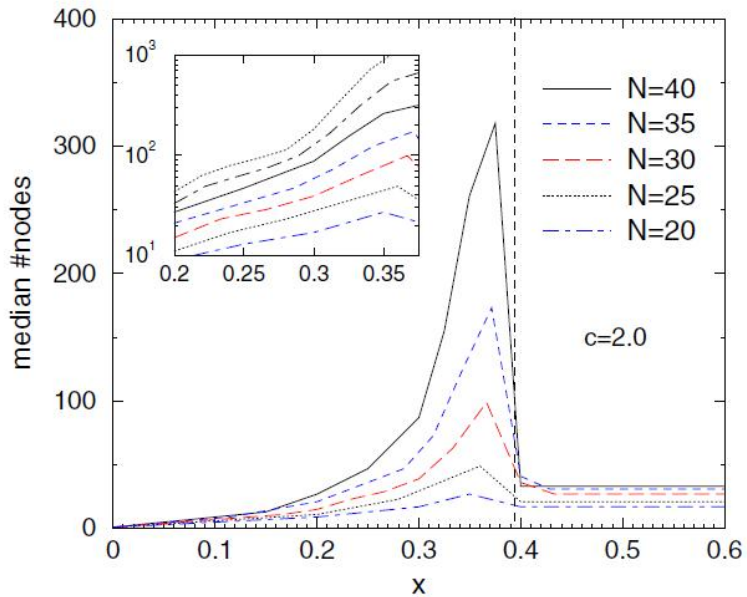


As $n \to +\infty$, the transition concentrates around $\alpha_c \approx 4.26$

Phase transitions for the *VCP*

The *VCP* exhibits a similar phase transition as $\frac{|x|}{|V|}$ increases

- the computational time first explodes, then drops
  (*using a well-known exact algorithm*)
- as $n \to +\infty$ the transition concentrates around a critical value



When $|x|/|V|$ is small, some vertices are clearly necessary: problem solved
when $|x|/|V|$ is large, many vertices are clearly necessary: problem solved