

# 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](mailto: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>

# 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 randomization 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](http://www.sls-book.net) for further information.

# Experimental analysis

The experimental approach is very common in science

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

Therefore, it is easy to forget the basics of the **experimental approach**

- 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 the study of algorithms?*

# Purposes of the analysis

The experimental analysis investigates

- 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 sort them
- 3 describe the relation between the indices and parametric values of the instances (size  $n$ , etc. . . )
- 4 suggest improvements to the algorithms

# Benchmark

As not all instances can be tested, a benchmark sample must be defined

A **meaningful 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

- **instances 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

# Comparing heuristic algorithms

A heuristic algorithm is better than another one when it simultaneously

- ① obtains better results
- ② requires a smaller time

Slow algorithms with good results and fast algorithms with bad results cannot be compared in a meaningful way

It can be justified to neglect the computational time when

- considering a single algorithm with no comparison
- comparing algorithms that perform the same operations (e. g., variants obtained modifying a numerical parameter)
- comparing algorithms that mostly perform the same operations with few different ones that take a negligible fraction of the time (e. g., different initializations or perturbations)

# A statistical model of algorithm performance

If we model the execution of algorithm  $A$  as a random experiment

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

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

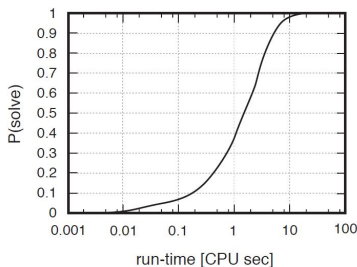


# Analysis of the computational time (*RTD* diagram)

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

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

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

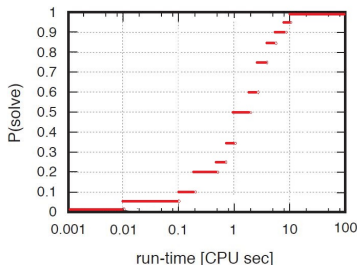


If all influential parameters are identified and fixed, the *RTD* diagram degenerates into a step function (*all instances require the same time*)

# 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  $t \geq \max_{I \in \mathcal{I}} T(I)$** : all are solved within the longest time



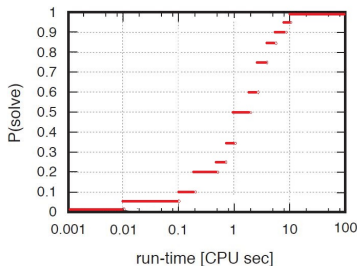
For large benchmark samples, the plot looks continuous, but it is not!

*(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  $T_A(\bar{\mathcal{I}}) = \{T_A(I) : I \in \bar{\mathcal{I}}\}$
- 3 sort  $T_A(\bar{\mathcal{I}})$  by nondecreasing values:  $t_1 \leq \dots \leq t_{|\bar{\mathcal{I}}|}$
- 4 plot the points  $\left(t_j, \frac{j}{|\bar{\mathcal{I}}|}\right)$  for  $j = 1, \dots, |\bar{\mathcal{I}}|$  (for equal  $t_j$ , the highest) and the horizontal segments (close on the left, open on the right)

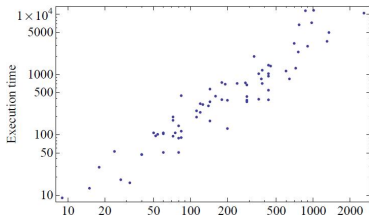


# Analysis of the computational time (*scaling* diagram)

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

- generate a sequence of values of  $n$  and a sample  $\bar{I}_n$  for each value
- apply the algorithm to each  $I \in \bar{I}_n$  for all  $n$

- sketch all points  $(n(I), T(I))$  or the mean points  $\left( n, \frac{\sum_{I \in \bar{I}_n} T(I)}{|\bar{I}_n|} \right)$
- 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*)

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

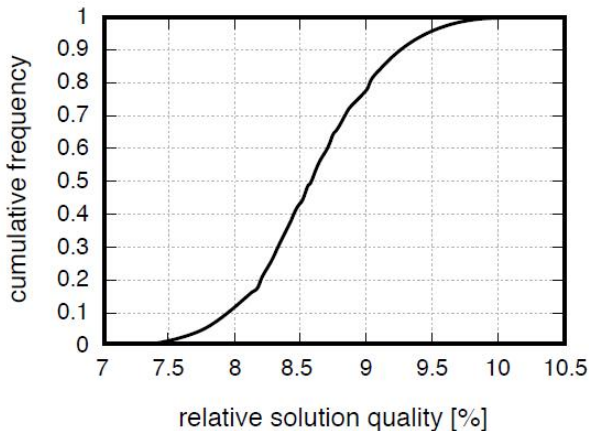
- a **polynomial algorithm** is represented on a **logarithmic scale**  
(*the logarithm is applied to both axes*)

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

# Analysis of the quality of the solution (*SQD*) diagram

The *Solution Quality Distribution* (*SQD*) diagram is the plot of the distribution function of  $\delta_A(I)$  on  $\bar{I}$

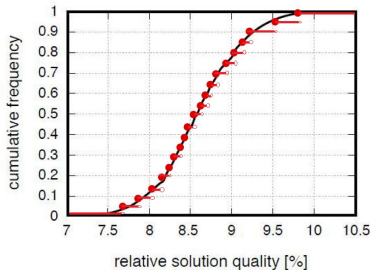
$$F_{\delta_A}(\alpha) = \Pr[\delta_A(I) \leq \alpha] \text{ for each } \alpha \in \mathbb{R}$$



# Solution Quality Distribution (SQD) diagram

For any algorithm, the distribution function of  $\delta_A(I)$

- **monotone nondecreasing**: more instances are solved with worst gaps
- **stepwise and right-continuous**: the graph steps up at each  $\delta(I)$
- **equal to zero for  $\alpha < 0$** : no instance is solved with negative gap
- **equal to 1 for  $\alpha \geq \max_{I \in \tilde{I}} \delta(I)$** : all are solved within the largest gap



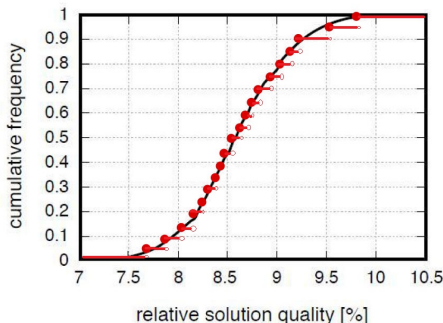
If  $A$  is an

- **exact algorithm**, it is a **stepwise function**, equal to 1 for all  $\alpha \geq 0$
- **$\bar{\alpha}$ -approximated algorithm**, it is a function equal to 1 for all  $\alpha \geq \bar{\alpha}$

# Building the $SQD$ diagram

In order to build the diagram

- 1 run the algorithm on each instance  $I \in \bar{\mathcal{I}}$
- 2 build the set  $\Delta_A(\bar{\mathcal{I}}) = \{\delta_A(I) : I \in \bar{\mathcal{I}}\}$
- 3 sort  $\Delta_A(\bar{\mathcal{I}})$  by nondecreasing values:  $\delta_1 \leq \dots \leq \delta_{|\bar{\mathcal{I}}|}$
- 4 plot the points  $\left(\delta_j, \frac{j}{|\bar{\mathcal{I}}|}\right)$  for  $j = 1, \dots, |\bar{\mathcal{I}}|$  (for equal  $\delta_j$ , the highest) and the horizontal segments (close on the left, open on the right)

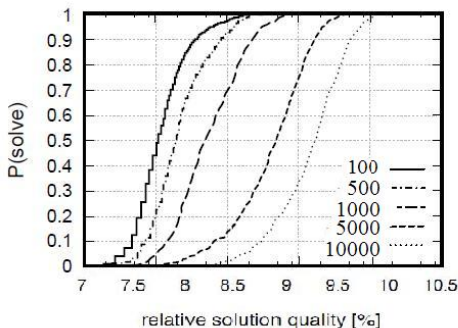




# Parametric SQD diagrams

Given the theoretical and practical problems to build a meaningful sample often the diagram is parameterized with respect to

- a descriptive parameter of the instances (size, density, ...)
- a parameter of the probability distribution assumed for the instances (expected value or variance of the costs, ...)



The conclusions are more limited, but the sample is more significant

General trends can be highlighted (what happens as size increases?)

# Comparison between algorithms with the *SQDs*

How to determine whether an algorithm is better than another?

- **strict dominance**: it obtains better results on all instances

$$\delta_{A_2}(I) \leq \delta_{A_1}(I) \quad \text{for each } I \in \mathcal{I}$$

This usually happens only in trivial cases (e.g.,  $A_2$  “includes”  $A_1$ )

- **probabilistic dominance**: the distribution function has higher values for every value of  $\alpha$

$$F_{\delta_{A_2}}(\alpha) \geq F_{\delta_{A_1}}(\alpha) \quad \text{for all } \alpha \in \mathbb{R}$$

The following plot shows no dominance, but  $A_1$  is less “robust” than  $A_2$ :  $A_1$  has results more dispersed than  $A_2$  (both better and worse)

