Heuristic Algorithms for Combinatorial Optimisation problems

Ph.D. course in Computer Science

Roberto Cordone

DI - Università degli Studi di Milano

E-mail: roberto.cordone@unimi.it

Web page: https://homes.di.unimi.it/cordone/courses/2025-haco/2025-haco.html

Problems

Informally, a problem is a question on a system of mathematical objects

The same question can often be asked on many similar systems

- an instance $I \in \mathcal{I}$ is each specific system concerned by the question
- a solution $S \in \mathcal{S}$ is an answer corresponding to one of the instances

Example: "*is n a prime number?* " is a problem with infinite instances and two solutions ($\mathcal{I} = \mathbb{N}^+ \setminus \{1\}$ and $\mathcal{S} = \{ \texttt{yes}, \texttt{no} \}$)
instance $I = 7$ corresponds to solution $S_I = \texttt{yes}$
instance $I' = 10$ corresponds to solution $S_{I'} = \texttt{no}$
...

Formally, a problem is the function which relates instances and solutions

$$P : \mathcal{I} \to \mathcal{S}$$

Defining a function does not mean to know how to compute it

Algorithms

An algorithm is a formal procedure, composed by elementary steps, in finite sequence, each determined by an input and by the results of the previous steps

An algorithm for a problem P is an algorithm which, given in input $I \in \mathcal{I}$, returns in output $S_I \in \mathcal{S}$

$$A : \mathcal{I} \to \mathcal{S}$$

An algorithm defines a function plus the way to compute it; it is

- exact if its associated function coincides with the problem
- heuristic otherwise

A heuristic algorithm is useful if it is

1. efficient: it "costs" much less than an exact algorithm
2. effective: it "frequently" provides a solution "close" to the right one

Let us start from the efficiency

Cost of a heuristic algorithm

The "cost" of an (exact or heuristic) algorithm denotes

- not the monetary cost to buy or implement it
- but the computational cost of running it
  - time required to terminate the finite sequence of elementary steps
  - space occupied in memory by the results of the previous steps

The time is much more discussed because

- the space is a renewable resource, the time is not
- using space requires to use at least as much time
- it is technically easier to distribute the use of space than of time

Space and time are partly interchangeable:
it is possible to reduce the use of one by increasing the use of the other

A useful measure of time

The time required to solve a problem depends on several aspects

- the specific instance to solve
- the algorithm used
- the machine running the algorithm
- ...

Our measure of the computational time should be

- unrelated to technology, that is the same for different machines
- concise, that is summarised in a simple symbolic expression
- ordinal, that is sufficient to compare different algorithms

The computational time in seconds for each instance violates all requisites

Worst-case asymptotic time complexity

The worst-case asymptotic complexity of an algorithm (nearly) provides such a measure through the following passages

1. define time as the number T of elementary operations performed (that is a value independent from the specific computer)
2. define the size of an instance as a suitable value n (e.g., the number of elements of the ground set, variables or clauses of the CNF, rows or columns of the matrix, nodes or arcs of the graph)
3. find the worst-case, i. e. the maximum of T on all instances of size n

$$T(n) = \max_{I \in \mathcal{I}_n} T(I) \qquad n \in \mathbb{N}$$

(*now time complexity is only a function* $T : \mathbb{N} \to \mathbb{N}$)

4. approximate $T(n)$ from above and/or below with a simpler function $f(n)$, considering only their asymptotic behaviour (for $n \to +\infty$)
   (*the algorithm should be efficient on instances of large size*)
5. collect the functions in classes with the same approximating function
   (*the approximation relation is an equivalence relation*)

The Θ functional spaces

$$T(n) \in \Theta(f(n))$$

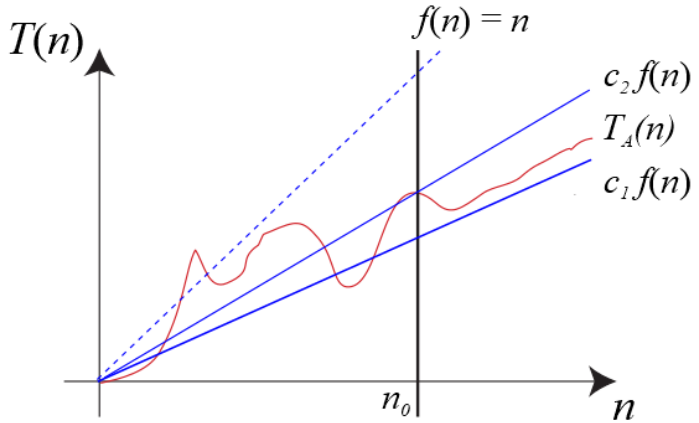
formally means that

$$\exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} : c_1 f(n) \leq T(n) \leq c_2 f(n) \text{ for all } n \geq n_0$$

where c_1, c_2 and n_0 are independent from n

$T(n)$ is *"enclosed"* between $c_1 f(n)$ and $c_2 f(n)$

- for some *"small"* value of c_1
- for some *"large"* value of c_2
- for some *"large"* value of n_0
- for some definition of *"small"* and *"large"*



Asymptotically, $f(n)$ estimates $T(n)$ up to a multiplying factor:

- for large instances, the computational time is at least and at most proportional to the values of function $f(n)$

The O functional spaces

$$T(n) \in O(f(n))$$

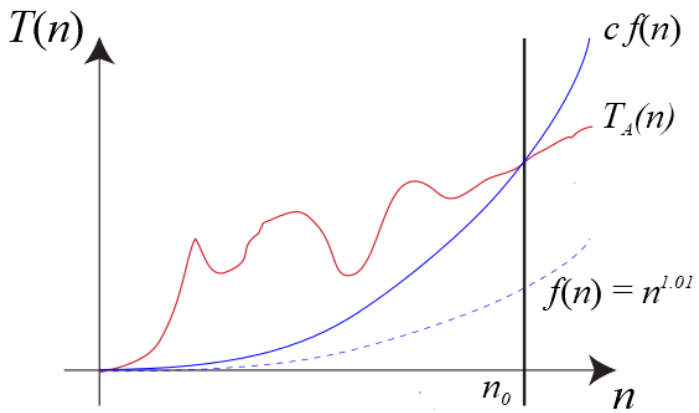
formally means that

$$\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} : T(n) \leq c\, f(n) \text{ for all } n \geq n_0$$

where c, and n_0 are independent from n

$T(n)$ is *"dominated"* by $c\, f(n)$

- for some *"large"* value of c
- for some *"large"* value of n_0
- for some definition of *"small"* and *"large"*



Asymptotically, $f(n)$ overestimates $T(n)$ up to a multiplying factor:

- for large instances, the computational time is at most proportional to the values of function $f(n)$

The Ω functional spaces

$$T(n) \in \Omega(f(n))$$

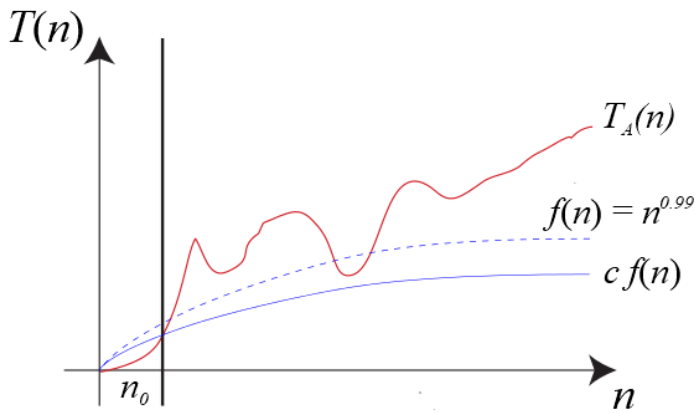
formally means that

$$\exists c > 0, n_0 \in \mathbb{N} : T(n) \geq c\, f(n) \text{ for all } n \geq n_0$$

where c and n_0 are independent from n

$T(n)$ *"dominates"* $c\, f(n)$

- for some *"small"* value of c
- for some *"large"* value of n_0
- for some definition of *"small"* and *"large"*



Asymptotically, $f(n)$ underestimates $T(n)$ up to a multiplying factor:

- for large instances, the computational time is at least proportional to the values of function $f(n)$

The exhaustive algorithm

For Combinatorial Optimisation problems the size of an instance can be measured by the cardinality of the ground set

$$n = |B|$$

The exhaustive algorithm

- considers each subset $x \subseteq B$, that is each $x \in 2^{|B|}$
- tests its feasibility ($x \in X$) in time $\alpha(n)$
- in the positive case, it evaluates the objective $f(x)$ in time $\beta(n)$
- if necessary, it updates the best value found so far

The time complexity of the exhaustive algorithm is

$$T(n) \in \Theta\left(2^n\left(\alpha(n) + \beta(n)\right)\right)$$

that is at least exponential, even if $\alpha(n)$ and $\beta(n)$ are small polynomials (which is the most frequent case)

Most of the time, the exhaustive algorithm is impractical

Beyond the worst-case complexity

The worst-case complexity

- cancels all information on the easier instances
  (*how are they made? how many are they?*)
- gives a rough overestimate of the computational time,
  in some (rare) cases useless
  (*see the simplex algorithm for Linear Programming*)

What if the hard instances are rare in the practical applications?

To compensate, one can investigate

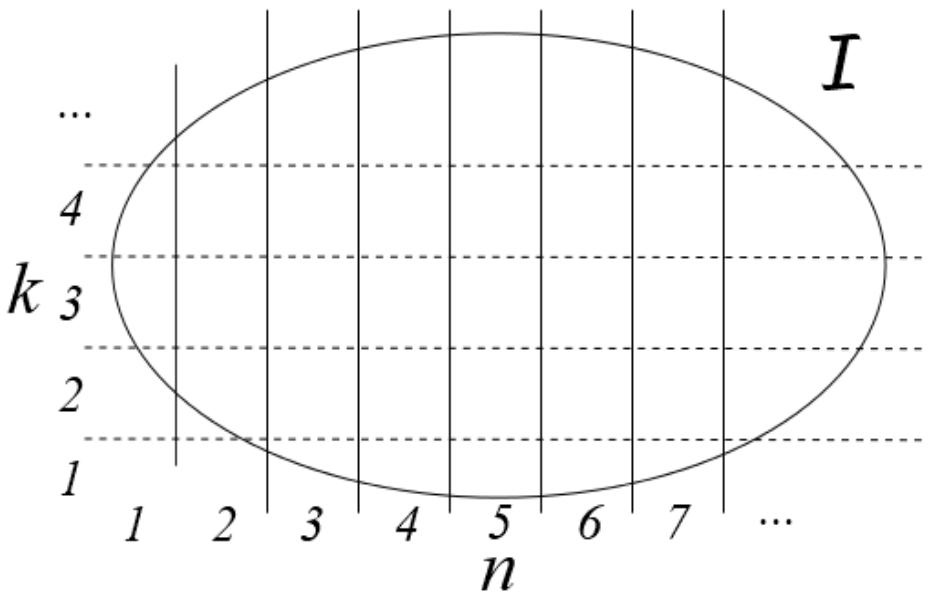
- the parameterised complexity, that is introduce some other relevant parameter k (besides the size n) and express the time as $T(n,k)$
- the average-case complexity, that is assume a probability distribution on $\mathcal{I}$ and express the time as the expected value

$$T(n) = E[T(I) \,|\, I \in \mathcal{I}_n]$$

Parameterised complexity

Some algorithms are exponential in k and polynomial in n, and therefore

- efficient on instances with low k
- inefficient on instances with large k



Nature of the additional parameter

If the additional parameter k is a part of the input, such as

- a numerical constant (e. g., the capacity in the *KP*)
- the maximum number of literals per clause in logic function problems
- the number of nonzero elements in numerical matrix problems
- the maximum degree, the diameter, etc. . . in graph problems

one knows *a priori* whether the algorithm is efficient on a given instance

If the additional parameter k is a part of the solution, such as

- its cardinality (as in the *VCP*)

one will only find out *a posteriori*

(*but an a priori estimate could be available*)

Average-case complexity

Some algorithms are efficient only on nearly all instances
(*see the simplex algorithm for Linear Programming*)

The worst-case characterisation is not very significant

Theoretical studies define a probabilistic model of the problem

- they assume a probability distribution on $\mathcal{I}_n$ for each $n \in \mathbb{N}$
  typically quite simple (e.g., equiprobability, that is full ignorance)
- they prove properties of the expected value of $T(I)$

$$T(n) = E[T(I) \,|\, I \in \mathcal{I}_n]$$

Empirical studies

- build a simulation model of the problem,
  that is a probability distribution on $\mathcal{I}_n$ for each $n \in \mathbb{N}$
  theoretical or empirical (drawn from real-world data)
- build a benchmark of random instances according to the distribution
- apply the algorithm and measure the time required
- give a statistical description of the measures obtained

Computational cost of heuristic algorithms

The time complexity of a heuristic algorithm is usually

- strictly polynomial (with low exponents)
- fairly robust with respect to secondary parameters

Therefore, the worst-case estimation is also good on average

Metaheuristics use random steps or memory

- the complexity is well defined for single components of the algorithm
- the overall complexity is not clearly defined
  - in theory, it could extend indefinitely (but the pseudorandom number generator or the memory configurations would yield an infinite loop)
  - in practice, it is defined by a condition imposed by the user

(*more about this later*)

Effectiveness of a heuristic algorithm

A heuristic algorithm is useful if it is

1. efficient: it "costs" much less than an exact algorithm
2. effective: it "frequently" returns a solution "close to" an exact one

Let us now discuss the effectiveness of heuristic algorithms:

- closeness of the solution obtained to an optimal one
- frequency of hitting optimal or nearly optimal solutions

These features can be combined into a

- frequency distribution of solutions more or less close to the optimum

The effectiveness of a heuristic algorithm can be investigated with a

- theoretical analysis (*a priori*), proving that the algorithm finds always or with a given frequency solutions with a given guarantee of quality
- experimental analysis (*a posteriori*), measuring the performance of the algorithm on sampled benchmark instances to show that a guarantee of quality is respected in practice

Indices of effectiveness

The effectiveness of a heuristic optimisation algorithm A is measured by the difference between the heuristic value $f_A(I)$ and the optimum $f^*(I)$

- absolute difference:

$$\tilde{\delta}_A(I) = |f_A(I) - f^*(I)| \geq 0$$

  rarely used, and only when the objective is a pure number

- relative difference:

$$\delta_A(I) = \frac{|f_A(I) - f^*(I)|}{f^*(I)} \geq 0$$

  frequent in experimental analysis (*usually as a percent ratio*)

- approximation ratio:

$$\rho_A(I) = \max\left[\frac{f_A(I)}{f^*(I)}, \frac{f^*(I)}{f_A(I)}\right] \geq 1$$

  frequent in theoretical analysis: the first form is used for minimisation problems, the second one for maximisation problems

Theoretical analysis (in the worst case)

To obtain a compact measure, independent from I, find the worst case
(*as for efficiency, that is complexity*)

The difference between $f_A(I)$ and $f^*(I)$ is in general unlimited,
but for some algorithms it is limited:

- absolute approximation:

$$\exists \tilde{\alpha}_A \in \mathbb{N} : \tilde{\delta}_A(I) \leq \tilde{\alpha}_A \text{ for each } I \in \mathcal{I}$$

  A (rare) example is Vizing's algorithm for *Edge Coloring* ($\tilde{\alpha}_A = 1$)

- relative approximation:

$$\exists \alpha_A \in \mathbb{R}^+ : \rho_A(I) \leq \alpha_A \text{ for each } I \in \mathcal{I}$$

Factor α_A ($\tilde{\alpha}_A$) is the relative (absolute) approximation guarantee

For other algorithms, the guarantee depends on the instance size

$$\rho_A(I) \leq \alpha_A(n) \text{ for each } I \in \mathcal{I}_n, n \in \mathbb{N}$$

Effectiveness can be independent from size (contrary to efficiency)

How to achieve an approximation guarantee?

For a minimisation problem, the aim is to prove that

$$\exists \alpha_A \in \mathbb{R} : f_A(I) \leq \alpha_A f^*(I) \text{ for each } I \in \mathcal{I}$$

1. find a way to build an underestimate $LB(I)$

$$LB(I) \leq f^*(I) \qquad I \in \mathcal{I}$$

2. find a way to build an overestimate $UB(I)$, related to $LB(I)$ by a coefficient α_A

$$UB(I) = \alpha_A \, LB(I) \qquad I \in \mathcal{I}$$

3. find an algorithm A whose solution is not worse than $UB(I)$

$$f_A(I) \leq UB(I) \qquad I \in \mathcal{I}$$

Then $f_A(I) \leq UB(I) = \alpha_A LB(I) \leq \alpha_A f^*(I)$, for each $I \in \mathcal{I}$

$$f_A(I) \leq \alpha_A f^*(I) \text{ for each } I \in \mathcal{I}$$

The *TSP* under the triangle inequality

Consider the *TSP* with the additional (rather common) assumptions that

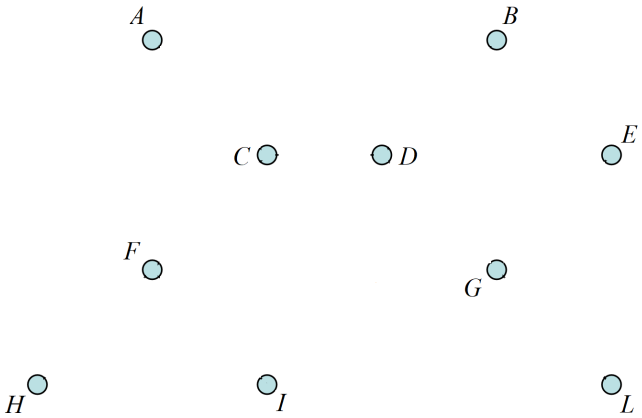
- graph $G = (N, A)$ is complete
- cost c is nonnegative, symmetric and satisfies the triangle inequality

$$c_{ij} = c_{ji} \geq 0 \quad \forall i, j \in N \qquad \text{and} \qquad c_{ij} + c_{jk} \geq c_{ik} \quad \forall i, j, k \in N$$

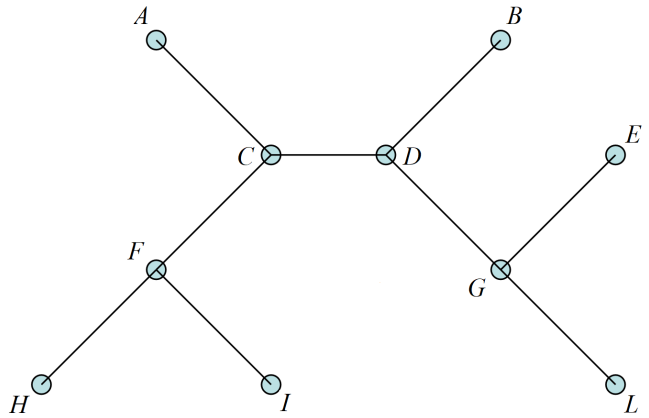
Double-tree algorithm

1. Consider the complete undirected graph corresponding to G
2. Build a minimum cost spanning tree $T^* = (N, X^*)$
3. Make a pre-order visit of T^* and build two lists of arcs:
   a. x' lists the arcs used both by the visit and the backtracking:
      this is a circuit visiting each node, possibly several times
   b. x lists the arcs linking the nodes in pre-order ending with the first:
      this is a circuit visiting each node exactly once

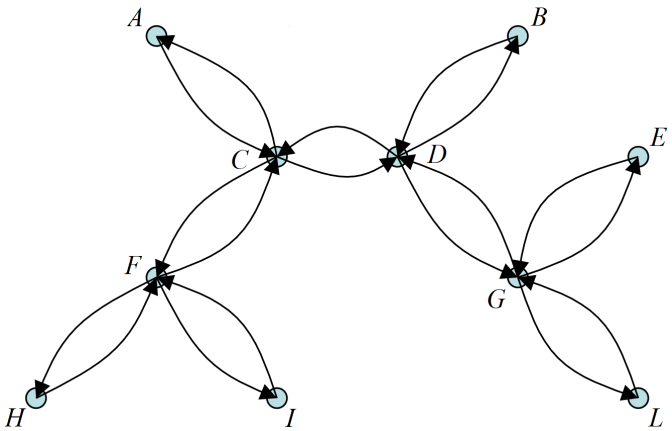
Example



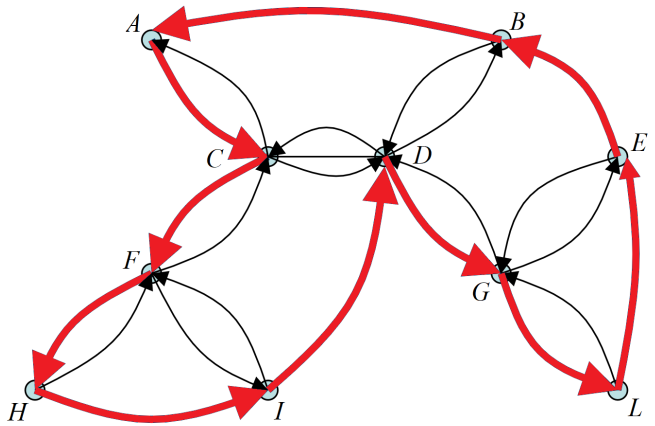
1) Complete graph G (arcs omitted)



2. Minimum spanning tree T^*



3.a) $x' = (A, C, F, H, F, I, F, C, D, G, L, G, E, G, D, B, D, C, A)$



3.b) $x = (A, C, F, H, I, D, G, L, E, B, A)$

Proof

The double-tree algorithm is 2-approximated

1. the cost of the minimum spanning tree is an underestimate $LB(I)$
   - deleting an arc from a Hamiltonian circuit yields a Hamiltonian path that is cheaper
   - a Hamiltonian path is a spanning tree (usually not of minimum cost)
2. the cost of circuit x' is
   - an overestimate $UB(I)$ (*it is a nonminimum Hamiltonian circuit*)
   - equal to $2LB(I)$ (*two arcs correspond to each edge*)
3. the cost of circuit x is $f_A(I) \leq UB(I)$
   (*a single direct arc replacing a sequence decreases the cost*)

This implies that $f_A(I) \leq 2f^*(I)$ for each $I \in \mathcal{I}$, that is $\alpha_A = 2$

Notice: x' is used in the approximation proof, but needs not be computed

Beyond the worst case

As usual, the worst-case approach is rough:
some algorithms often have a good performance, though sometimes bad

The alternative approaches are similar to the ones used for complexity

- parametrisation: prove an approximation guarantee that depends on other parameters of the instances besides the size n
- average-case: assume a probability distribution on the instances and evaluate the expected value of the approximation factor
  (*the algorithm could have a bad performance only on rare instances*)

but there is at least another approach

- randomisation: the operations of the algorithm depend not only on the instance, but also on pseudorandom numbers, so that the solution becomes a random variable which can be investigated
  (*the time complexity could also be random, but usually is not*)

Randomised approximation algorithms

For a randomised algorithm A, $f_A(I, \omega)$ and $\rho_A(I, \omega)$ are random variables depending on the pseudorandom number seed ω

A randomised approximation algorithm has an approximation ratio whose expected value is limited by a constant

$$E[\rho_A(I, \omega)] \leq \alpha_A \text{ for each } I \in \mathcal{I}$$

Max-SAT problem: given a CNF, find a truth assignment to the logical variables that satisfy a maximum weight subset of formulae

Purely random algorithm:

Assign to each variable x_j $(j = 1, \ldots, n)$

- value *False* with probability $1/2$
- value *True* with probability $1/2$

What is the expected value of the solution?

Randomised approximation for the *MAX-SAT*

The objective $f(x) = f_A(I, \omega)$ is the total weight of the satisfied clauses and its expected value is

$$E[f_A(I,\omega)] = E\left[\sum_{i\in\mathcal{C}} \delta_i(x(\omega))w_i\right] = \sum_{i\in\mathcal{C}} (w_i \cdot Pr[\delta_i(x(\omega)) = 1])$$

where $\delta_i(x(\omega)) = 1$ if solution $x(\omega)$ satisfies clause i, 0 otherwise

Let k_i be the number of literals of formula $i \in \mathcal{C}$ and $k_{\min} = \min_{i\in\mathcal{C}} k_i$

$$Pr[\delta_i(x(\omega)) = 1] = 1 - \left(\frac{1}{2}\right)^{k_i} \geq 1 - \left(\frac{1}{2}\right)^{k_{\min}} \quad \text{for each } i \in \mathcal{C}$$

$$\Rightarrow E[f_A(I,\omega)] \geq \sum_{i\in\mathcal{C}} w_i \cdot \left[1 - \left(\frac{1}{2}\right)^{k_{\min}}\right] = \left[1 - \left(\frac{1}{2}\right)^{k_{\min}}\right] \sum_{i\in\mathcal{C}} w_i$$

and since $\sum_{i\in\mathcal{C}} w_i \geq f^*(I)$ for each $I \in \mathcal{I}$ one obtains

$$\frac{E[f_A(I,\omega)]}{f^*(I)} \geq \left[1 - \left(\frac{1}{2}\right)^{k_{\min}}\right] \geq \frac{1}{2}$$

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 material 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 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 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

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

- 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

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 initialisations or perturbations)

A statistical model of algorithm performance

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

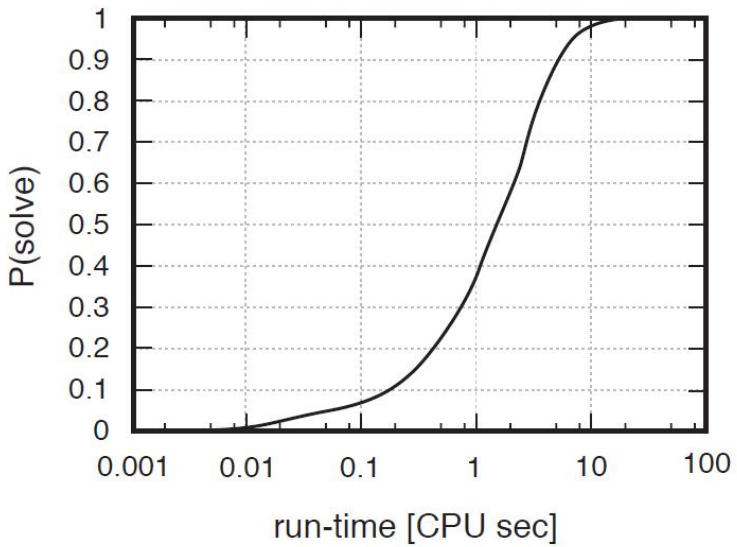
We describe the performance of A with the statistical properties of the random variables $T_A(I)$ and $\delta_A(I)$

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)

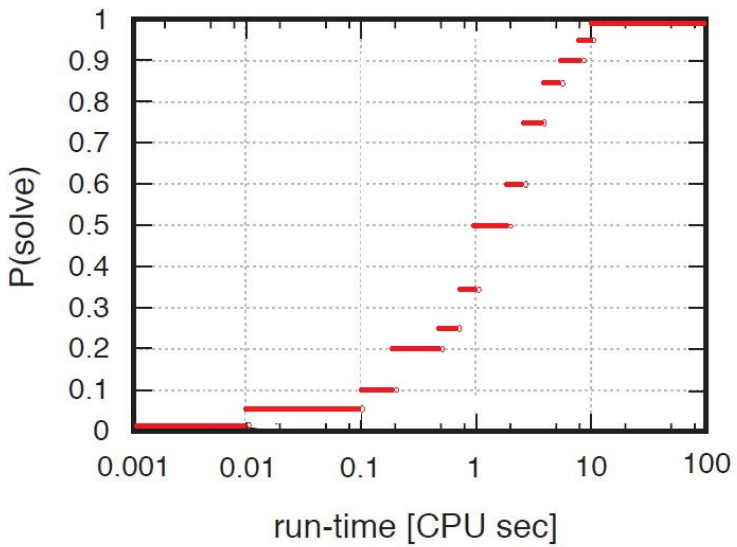


If all influential parameters are identified and fixed, the *RTD* diagram degenerates into a step function (*all instances require the same time*)

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



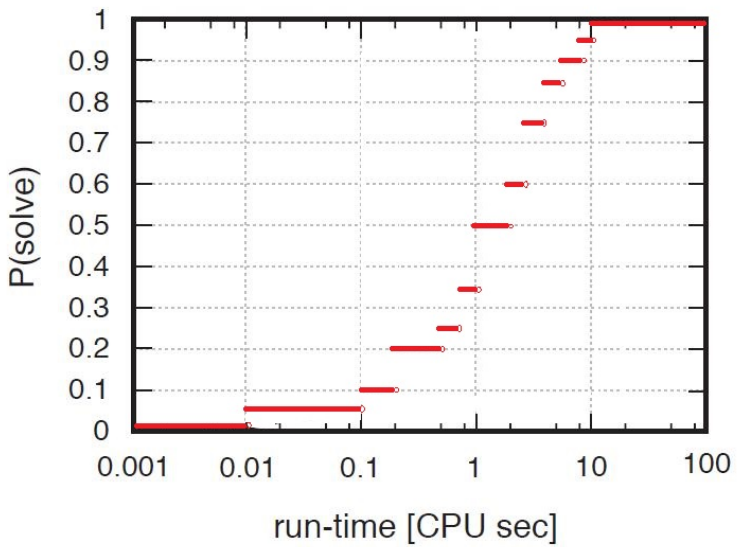
For large benchmark samples, the plot looks continuous, but it is not!

(*as in the previous page*)

Building the *RTD* diagram

In order to build the diagram

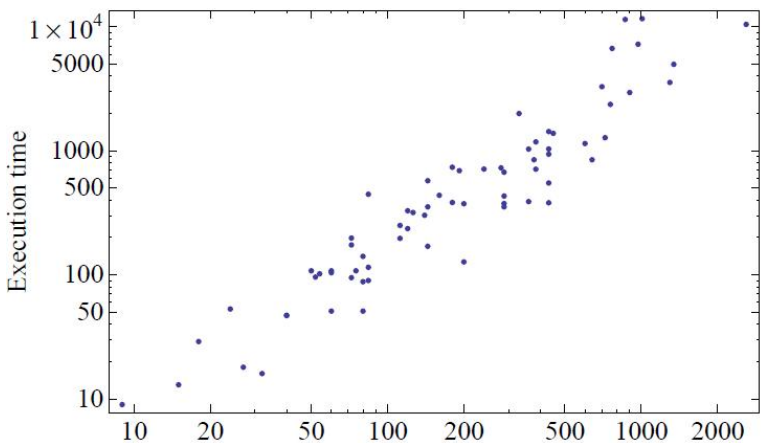
1. run the algorithm on each instance $I \in \bar{\mathcal{I}}$
2. build the set $T_A\left(\bar{\mathcal{I}}\right) = \left\{T_A\left(I\right) : I \in \bar{\mathcal{I}}\right\}$
3. sort $T_A\left(\bar{\mathcal{I}}\right)$ by nondecreasing values: $t_1 \leq \ldots \leq t_{|\bar{\mathcal{I}}|}$
4. plot 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*)

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

Estimates of $\delta_A(I)$

The computation of $\delta_A(I)$ requires to know the optimum $f^*(I)$

$$\delta_A(I) = \frac{|f_A(I) - f^*(I)|}{f^*(I)}$$

What if the optimum is unknown?

Replace it with an underestimate $LB(I)$ and/or an overestimate $UB(I)$

$$LB(I) \leq f^*(I) \leq UB(I) \Rightarrow \frac{1}{LB(I)} \geq \frac{1}{f^*(I)} \geq \frac{1}{UB(I)} \Rightarrow$$

$$\Rightarrow \frac{f_A(I)}{LB(I)} - 1 \geq \frac{f_A(I)}{f^*(I)} - 1 \geq \frac{f_A(I)}{UB(I)} - 1$$

$$\frac{f_A(I)}{f^*(I)} - 1 = \begin{cases} \delta_A(I) \text{ (minimisation)} & \Rightarrow \frac{f_A(I) - UB(I)}{UB(I)} \leq \delta_A(I) \leq \frac{f_A(I) - LB(I)}{LB(I)} \\ -\delta_A(I) \text{ (maximisation)} & \Rightarrow \frac{UB(I) - f_A(I)}{UB(I)} \leq \delta_A(I) \leq \frac{LB(I) - f_A(I)}{LB(I)} \end{cases}$$

and therefore

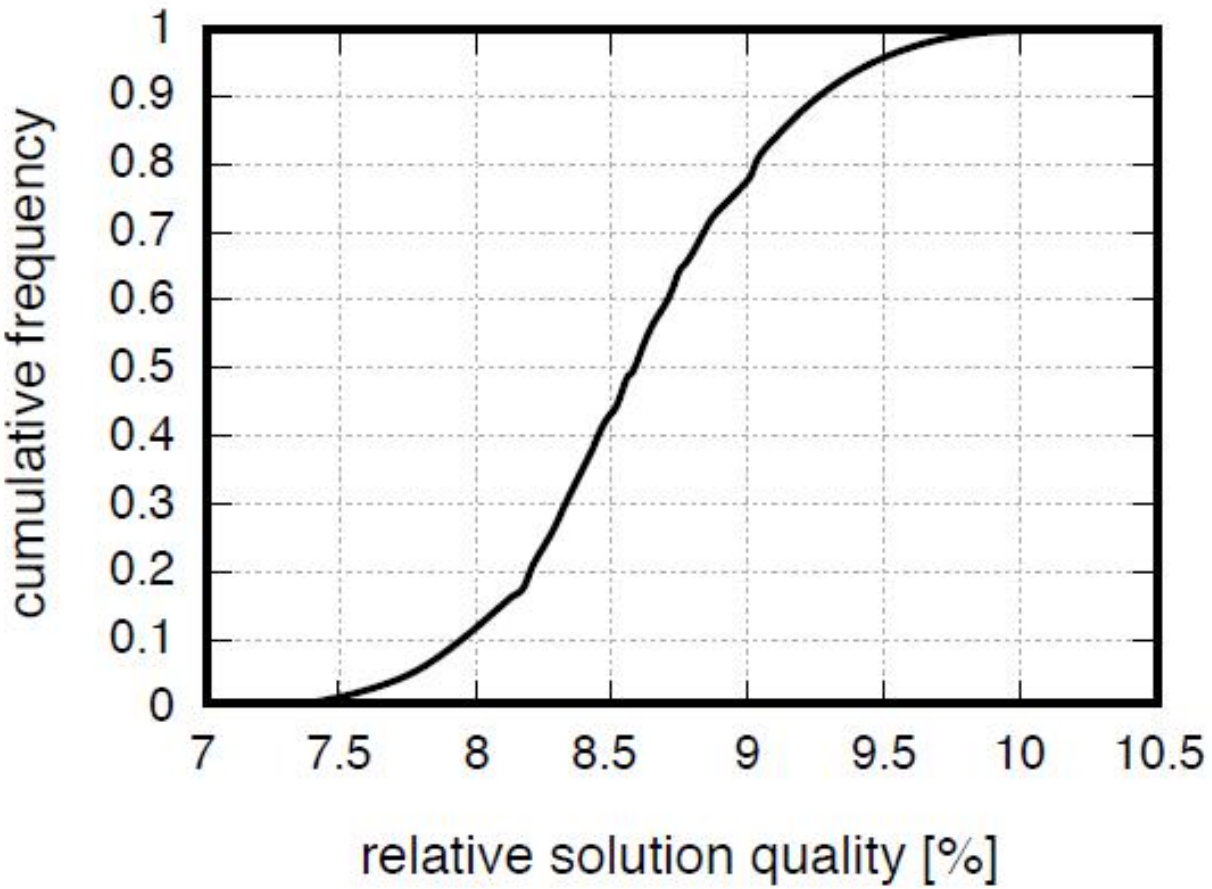
$$\frac{|f_A(I) - UB(I)|}{UB(I)} \leq \delta_A(I) \leq \frac{|f_A(I) - LB(I)|}{LB(I)}$$

This range turns all diagrams on δ_A into region estimates

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 $\bar{\mathcal{I}}$

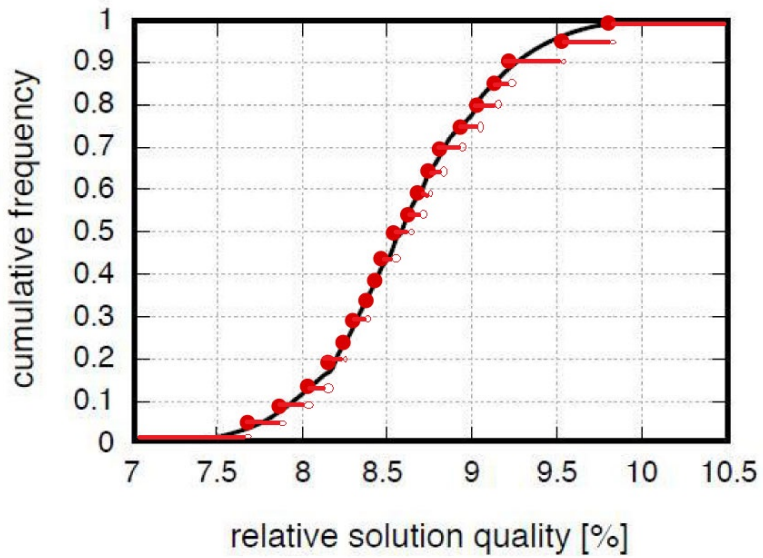
$$F_{\delta_A}(\alpha) = Pr[\delta_A(I) \leq \alpha] \text{ for each } \alpha \in \mathbb{R}$$



Solution Quality Distribution (*SQD*) diagram

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

- monotone nondecreasing: more instances are solved with worse 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 $\alpha \geq \max_{I \in \bar{\mathcal{I}}} \delta(I)$: all are solved within the largest gap



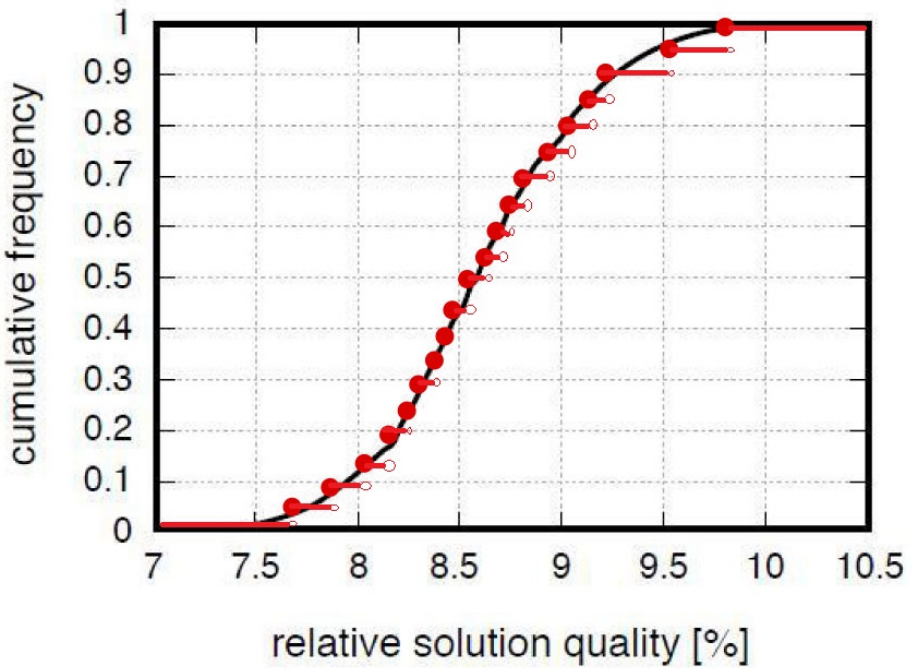
If A is an

- 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 large α

Building the *SQD* diagram

In order to build the diagram

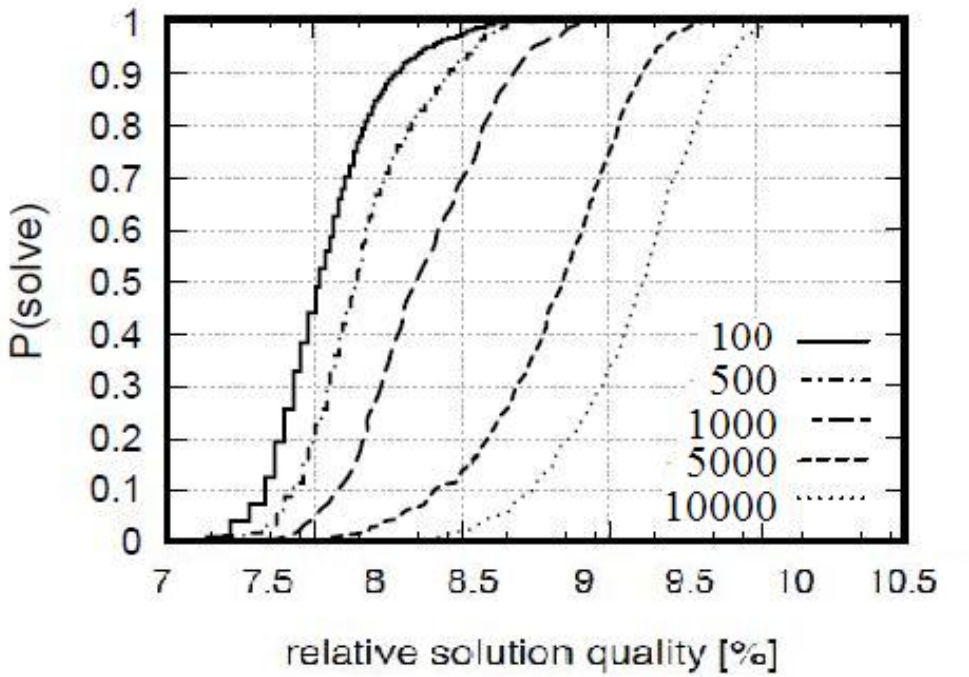
1. run the algorithm on each instance $I \in \bar{\mathcal{I}}$
2. build the set $\Delta_A\left(\bar{\mathcal{I}}\right) = \left\{\delta_A\left(I\right) : I \in \bar{\mathcal{I}}\right\}$
3. sort $\Delta_A\left(\bar{\mathcal{I}}\right)$ by nondecreasing values: $\delta_1 \leq \ldots \leq \delta_{|\bar{\mathcal{I}}|}$
4. plot points $\left(\delta_j, \frac{j}{|\bar{\mathcal{I}}|}\right)$ for $j = 1, \ldots, |\bar{\mathcal{I}}|$ (for equal δ_j, the highest j) 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 parameterised 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 *SQD*s

How to determine whether an algorithm is better than another?

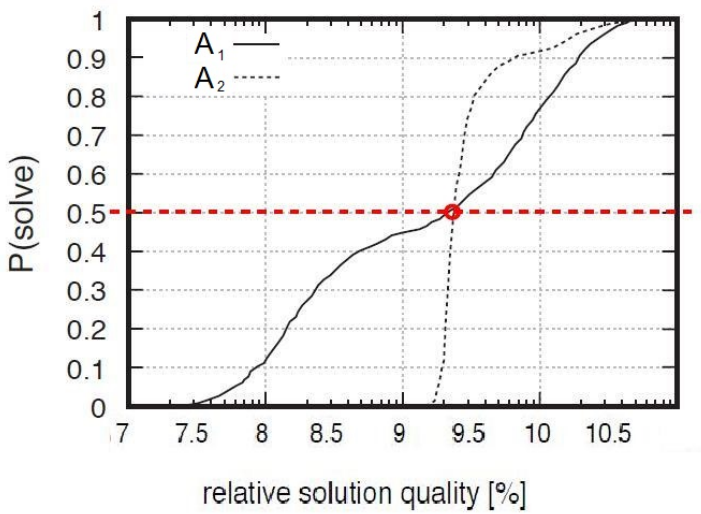
- strict dominance: it obtains better results on all instances

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

  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 α

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

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)



Compact statistical descriptions

The distribution function F_{δ_A} can be replaced or accompanied by more compact characterisations of the effectiveness of an algorithm

This typically involves classical statistical indices of

- position, such as the sample mean

$$\bar{\delta}_A = \frac{\sum_{I \in \bar{\mathcal{I}}} \delta_A(I)}{|\bar{\mathcal{I}}|}$$

- dispersion, such as the sample variance

$$\bar{\sigma}_A^2 = \frac{\sum_{I \in \bar{\mathcal{I}}} \left(\delta_A(I) - \bar{\delta}_A\right)^2}{|\bar{\mathcal{I}}|}$$

These indices "suffer" from the influence of outliers

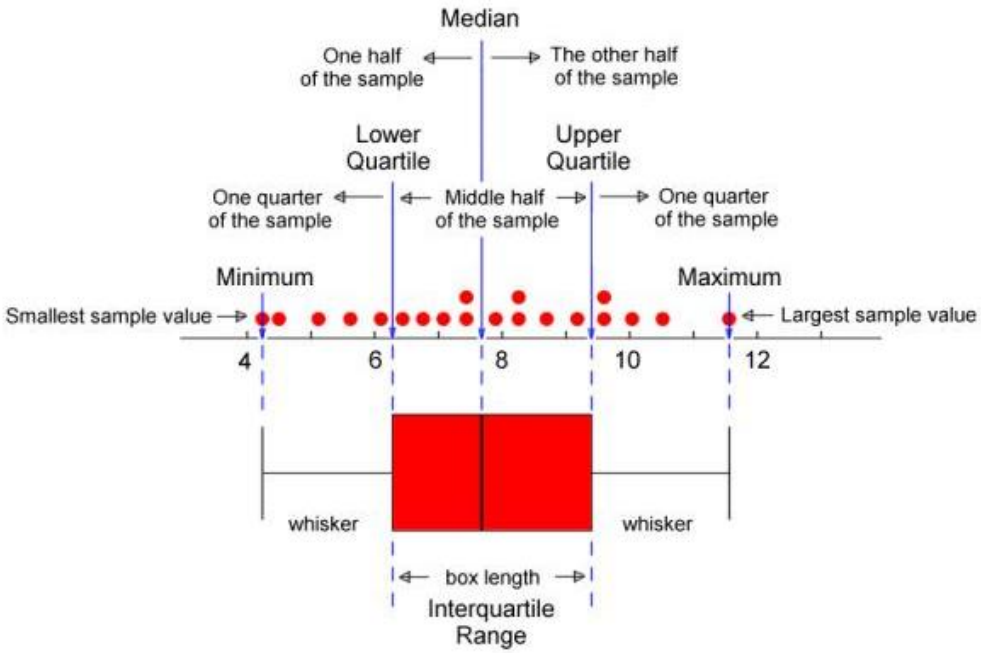
Other statistical indices are "stabler" and more detailed

- the sample median
- suitable sample quantiles

Boxplots

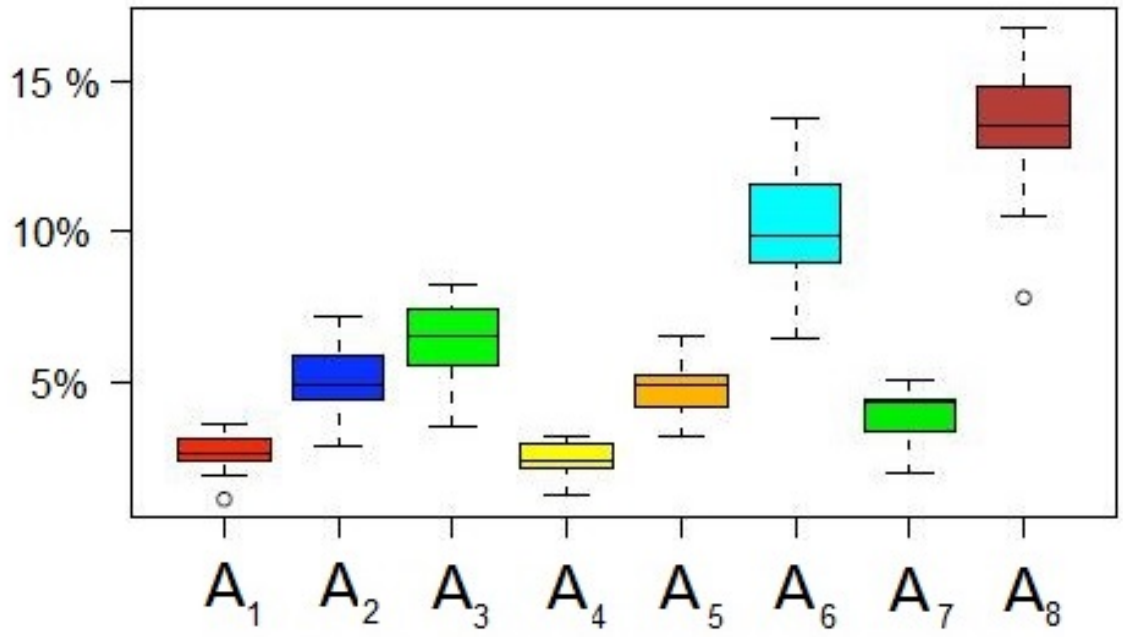
A graphic representation is the *boxplot* (or *box and whiskers diagram*)

- sample median ($q_{0.5}$)
- lower and upper sample quartiles ($q_{0.25}$ and $q_{0.75}$)
- the extreme sample values (often excluding the "outliers")



Comparison between algorithms with *boxplot* diagrams

A more compact comparison can be performed with *boxplot* diagrams



Necessary conditions

$$\text{Strict dominance} \Rightarrow \text{Probabilistic dominance} \Rightarrow q_i \leq q'_i \ (i = 1, \ldots, 5)$$

Strict dominance holds only if probabilistic dominance holds

Probabilistic dominance holds only if each of the five quartiles is not above the corresponding one of the other algorithm (e. g., $A_2 - A_3$)

Comparison between algorithms with *boxplot* diagrams

Sufficient conditions

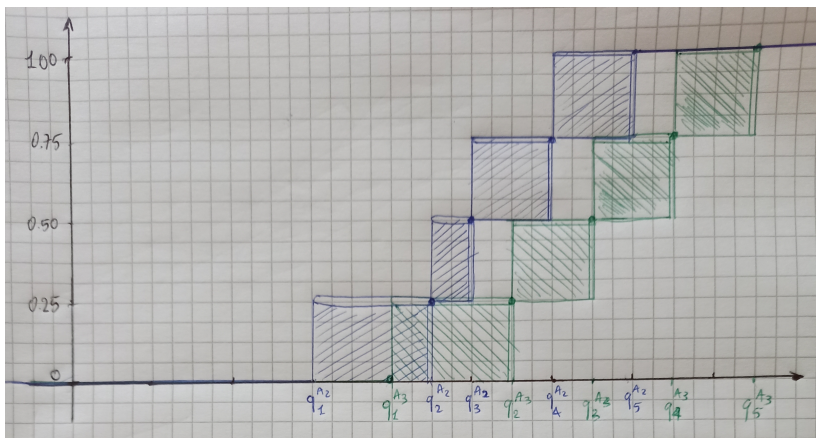
$$q_5 \leq q'_1 \Rightarrow \text{Strict dominance}$$

If a boxplot is fully below the other one, strict dominance holds (e. g., $A_7 - A_8$)

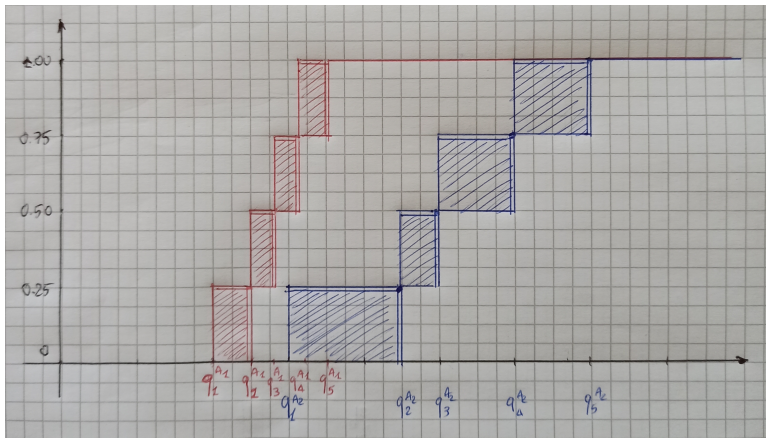
$$q_i \leq q'_{i-1}\ (i = 2, \ldots, 5) \Rightarrow \text{Probabilistic dominance}$$

If each of the five quartiles is below the preceding one of the other algorithm, probabilistic dominance holds (e. g., $A_1 - A_2$ or $A_6 - A_8$)

Necessary condition



Sufficient condition



Relation between quality and computational time

Many heuristic algorithms find several solutions during their execution, instead of a single one, and consequently can be terminated prematurely

In particular, metaheuristics (using random steps or memory mechanisms) have a computational time t fixed by the user and potentially unlimited

Let $\delta_A(I, t)$ be the relative difference reached by A at time t on instance I

As a function of time t, $\delta_A(I, t)$ is

- $+\infty$ if A has not yet found a feasible solution at time t
- stepwise monotone nonincreasing
- constant after the regular termination $(t \geq T(I))$

Randomised algorithms

For randomised algorithms the relative difference $\delta_A(I, \omega, t)$ depends on

1. the instance $I \in \mathcal{I}$
2. the outcome $\omega \in \Omega$ of the random experiment guiding the algorithm (*that is the random seed*)
3. the execution time t

Given a fixed time, these algorithms can be tested

1. on a sample of instances $\bar{\mathcal{I}}$ with a fixed seed ω
2. on a fixed instance I with a batch of seeds $\bar{\Omega}$ (different runs)
3. on several instances with several seeds on each instance

The results of multiple runs ($\bar{\Omega}$) are usually summarised providing both:

- the minimum relative difference $\delta^*_A(I, t)$ and the total time $\left|\bar{\Omega}\right| t$
- the average relative difference $\bar{\delta}_A(I, t)$ and the single-run time t

Classification

The relation between solution quality and computational time allows to classify the algorithms into:

- complete: for each instance $I \in \mathcal{I}$, find the optimum in finite time

$$\exists \bar{t}_I \in \mathbb{R}^+ : \delta_A(I, t) = 0 \text{ for each } t \geq \bar{t}_I, I \in \mathcal{I}$$

(*It is another name for exact algorithms*)

- probabilistically approximately complete: for each instance $I \in \mathcal{I}$, find the optimum with probability converging to 1 as $t \to +\infty$

$$\lim_{t \to +\infty} Pr[\delta_A(I, t) = 0] = 1 \text{ for each } I \in \mathcal{I}$$

(*many randomised metaheuristics*)

- essentially incomplete: for some instances $I \in \mathcal{I}$, find the optimum with probability strictly < 1 as $t \to +\infty$

$$\exists I \in \mathcal{I} : \lim_{t \to +\infty} Pr[\delta_A(I, t) = 0] < 1$$

(*most greedy algorithms, local search algorithms, . . .*)

A generalisation

An obvious generalisation replaces the search for the optimum with that for a given level of approximation

$$\delta_A(I,t) = 0 \;\rightarrow\; \delta_A(I,t) \leq \alpha$$

- α-complete algorithms: for each instance $I \in \mathcal{I}$, find an α-approximated solution in finite time (α-*approximated algorithms*)
- probabilistically approximately α-complete algorithms: for each instance $I \in \mathcal{I}$, find an α-approximated solution with probability converging to 1 as $t \rightarrow +\infty$
- essentially α-incomplete algorithms: for some instances $I \in \mathcal{I}$, find an α-approximated solution with probability strictly < 1 as $t \rightarrow +\infty$

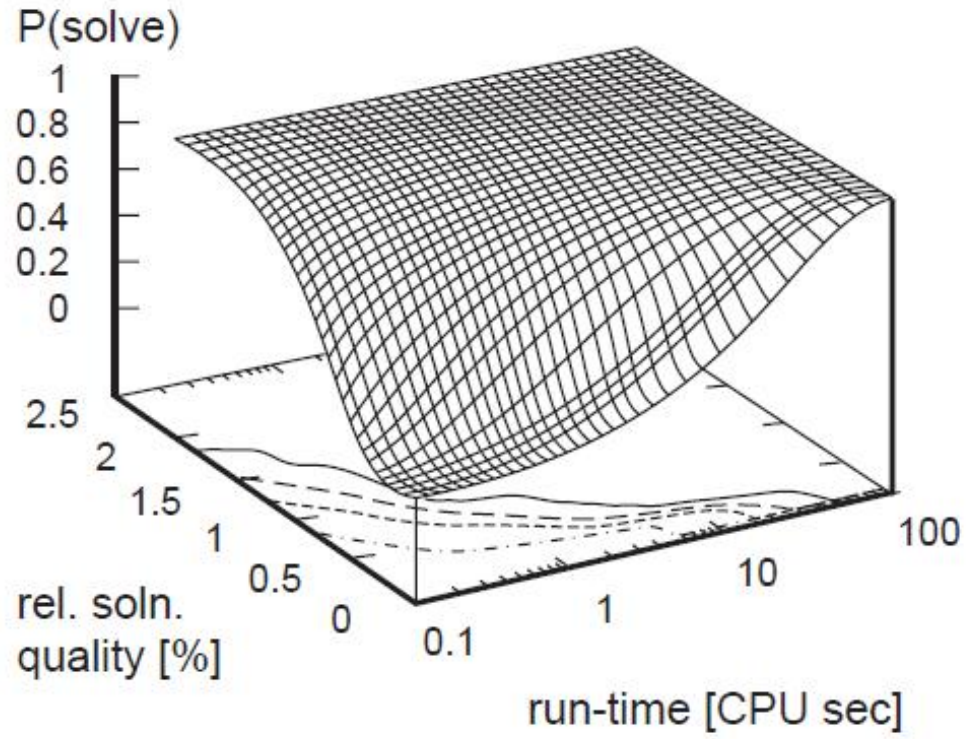
In conclusion, every algorithm provides compromises between

- a quality measure, described by the threshold α
- a time measure, described by the threshold t

The probability of success

Let the success probability $\pi_{A,n}(\alpha, t)$ be the probability that algorithm A find in time $\leq t$ a solution with a gap $\leq \alpha$ on an instance of size n

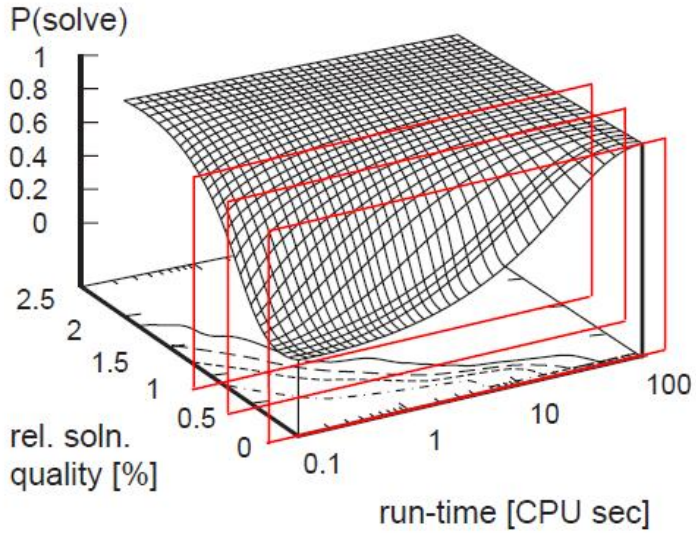
$$\pi_{A,n}(\alpha, t) = Pr[\delta_A(I, t) \leq \alpha | I \in \mathcal{I}_n, \omega \in \Omega]$$

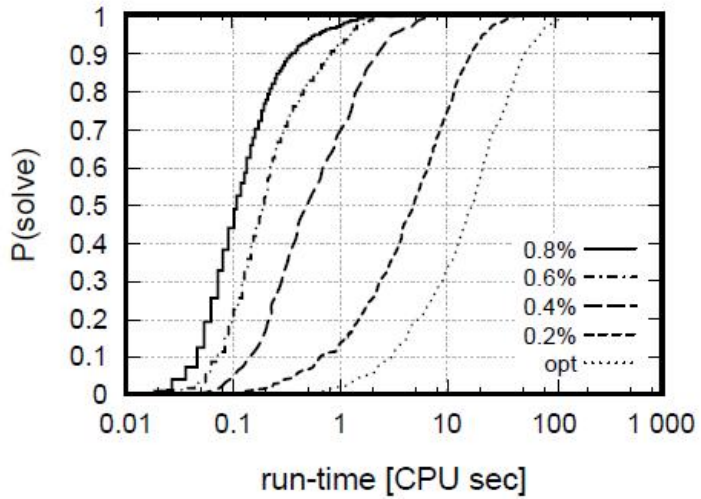


This yields different secondary diagrams

Qualified Run Time Distribution (*QRTD*) diagrams

The *QRTD* diagrams describe the profile of the time required to reach a specified level of quality





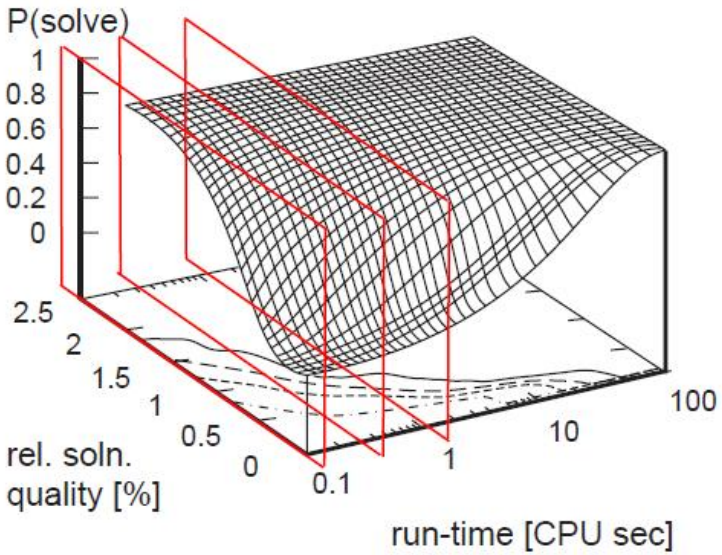
They are useful when the computational time is not a tight resource

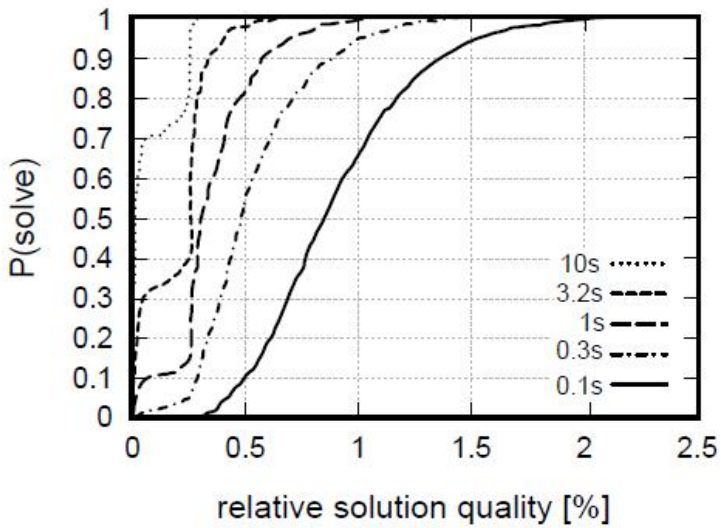
If the algorithm is

- complete, all diagrams reach 1 in finite time
- $\bar{\alpha}$-complete, all diagrams with $\alpha \geq \bar{\alpha}$ reach 1 in finite time
- $\bar{\alpha}$-incomplete, all diagrams with $\alpha \leq \bar{\alpha}$ do not reach 1

Timed Solution Quality Distribution (*TSQD*) diagrams

The *TSQD* diagrams describe the profile of the level of quality reached in a given computational time





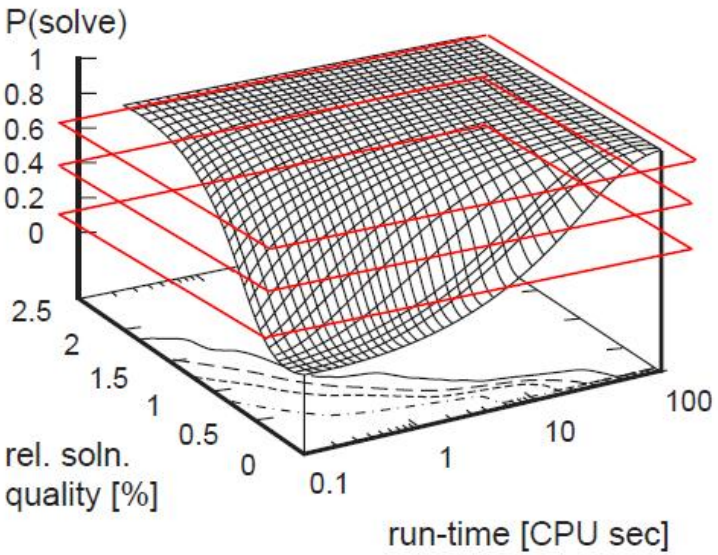
They are useful when the computational time is a tight resource

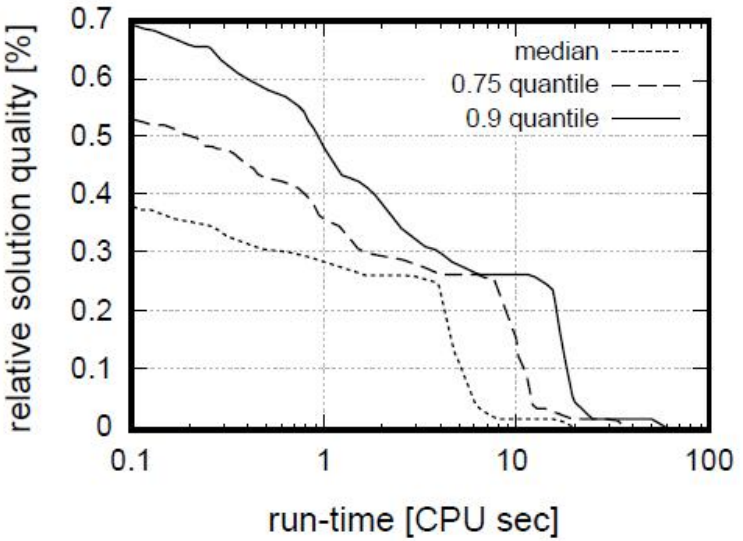
If the algorithm is

- complete, all diagrams with a sufficient t are step functions in $\alpha = 0$
- $\bar{\alpha}$-complete, all diagrams with a sufficient t reach 1 in $\alpha = \bar{\alpha}$
- probab. approx. $\bar{\alpha}$-complete, the diagrams converge to 1 in $\alpha = \bar{\alpha}$
- $\bar{\alpha}$-incomplete, all diagrams keep < 1 in $\alpha = \bar{\alpha}$

Solution Quality statistics over Time (SQT) diagrams

Finally, one can draw the level lines associated to different quantiles





They describe the compromise between quality and computational time

For a robust algorithm the level lines are very close to each other

Statistical tests

Diagrams and boxplots are qualitative: how to evaluate quantitatively if the empirical difference between algorithms A_1 and A_2 is significant?

Wilcoxon's test focuses on effectiveness (neglecting robustness)

- $f_{A_1}(I) - f_{A_2}(I)$ is a random variable defined on the sample space $\mathcal{I}$
- formulate a null hypothesis H_0 according to which the theoretical median of $f_{A_1}(I) - f_{A_2}(I)$ is zero
- extract a sample of instances $\bar{\mathcal{I}}$ and run the two algorithms on it, obtaining a sample of pairs of values (f_{A_1}, f_{A_2})
- compute the probability p of obtaining the observed result or a more "extreme" one, assuming that H_0 is true
- set a significance level $\bar{p}$, that is the
  - maximum acceptable probability to reject H_0 assuming that it is true
  - that is, to consider two identical medians as different
  - that is, to consider two equivalent algorithms as differently effective (referring to the median of the gap)
- reject H_0 when $p < \bar{p}$

Typical values for the significance level are $\bar{p} = 5\%$ or $\bar{p} = 1\%$

Wilcoxon's test (assumptions)

It is a nonparametric test, that is, it does not make assumptions on the probability distribution of the tested values

It is useful to evaluate the performance of heuristic algorithms, because the distribution of the result $f_A(I)$ is unknown

It is based on the following assumptions:

- all data are measured at least on an ordinal scale
  (*the specific values do not matter, only their relative size*)
- the two data sets are matched and derive from the same population
  (*we apply A_1 and A_2 to the same instances, extracted from $\mathcal{I}$*)
- each pair of values is extracted independently from the others
  (*the instances are generated independently from one another*)

Wilcoxon's test (application)

1. compute the absolute differences $|f_{A_1}(I_i) - f_{A_2}(I_i)|$ for all $I_i \in \bar{\mathcal{I}}$
2. sort them by increasing values and assign a rank R_i to each one
3. separately sum the ranks of the pairs with a positive difference and those of the pairs with a negative difference

$$\begin{cases} W^+ = \sum_{i:f_{A_1}(I_i)>f_{A_2}(I_i)} R_i \\ W^- = \sum_{i:f_{A_1}(I_i)<f_{A_2}(I_i)} R_i \end{cases}$$

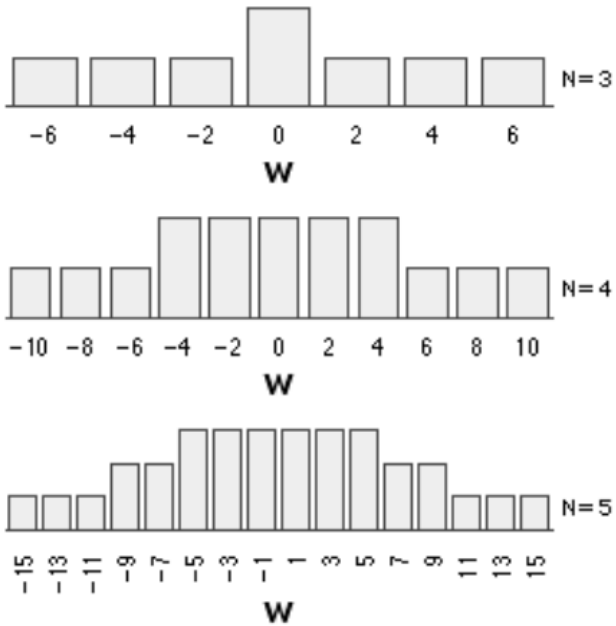
If the null hypothesis H_0 were true, the two sums should be equal

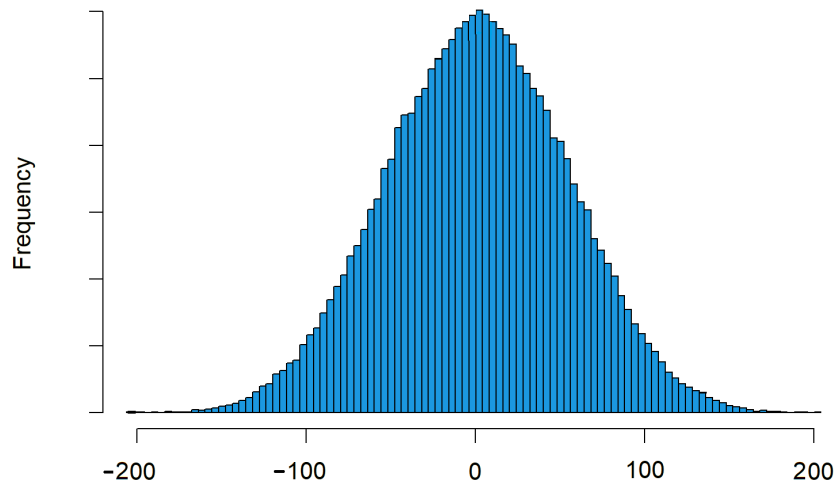
4. the difference $W^+ - W^-$ allows to compute the value of p:
each of the $|\bar{\mathcal{I}}|$ differences can be positive or negative: $2^{|\bar{\mathcal{I}}|}$ outcomes;
p is the fraction with $\left|W^+ - W^-\right|$ equal or larger than the observed value
5. if $p < \bar{p}$, the difference is significant and
   - if $W^+ < W^-$, A_1 is better than A_2
   - if $W^+ > W^-$, A_1 is worse than A_2

Computation of the p-value

The value of p is usually

- computed explicitly by enumeration when $|\bar{\mathcal{I}}| < 20$
- approximated with a normal distribution when $|\bar{\mathcal{I}}| \geq 20$





Of course, precomputed tables also exist

Possible conclusions

Wilcoxon's test can suggest

- that one of the two algorithms is significantly better than the other
- that the two algorithms are statistically equivalent

(*but take it as a stochastic response, and keep an eye on* p)

If the sample includes instances of different kinds, two algorithms could be overall equivalent, but nonequivalent on the single classes of instances

Dividing the sample could reveal

- classes of instances for which A_1 is better
- classes of instances for which A_2 is better
- classes of instances for which the two algorithms are equivalent

but multiplying questions means getting some wrong answers by chance (*FWER* = Family-Wise Error Rate)

Beware the garden of forking paths

What about testing $\delta_A(I)$ *instead of* $f_A(I)$*?*