Experimental analysis
Heuristic Algorithms

Giovanni Righini

University of Milan
Department of Computer Science (Crema)
Analysis of heuristic algorithms

A heuristic algorithm can be analyzed from a theoretical and an experimental viewpoint.

- **theoretical analysis** is *a priori*: the goal is to assess some formal property of the algorithm, such as its (worst-case) time complexity and possibly its (worst-case) approximation ratio.

- **experimental analysis** is *a posteriori*: the goal is to describe how the algorithm actually works in practice.

They can produce quite different results when algorithms are compared.
The experimental method is a cornerstone in modern science.

Mathematics does not study natural phenomena, but abstract concepts (including algorithms); therefore it does not rely on an experimental method.

The experimental analysis of heuristic algorithm is an exception: algorithms are studied as they were kind of natural phenomena.

The experimental method followed in natural sciences is:
1. start from quantitative observations (measures)
2. formulate a model to interpret the observations
3. design experiments to validate the model
4. do the experiments and measure the results
5. test whether the model fits with the results or not
6. improve the model until it is satisfactory.

What is a “model” in our context?
Quantitative observations

The experimental analysis of heuristic algorithms aims at investigating:

- the performance of the algorithms in terms of computing time (instead of computational complexity);
- the performance of the algorithms in terms of quality of the solutions (instead of approximation guarantees);
- the robustness of the algorithms with respect to some relevant parameters of the instances;
- the robustness of the algorithms with respect to their own parameters;
- the comparison between different algorithms for the same problem;
- how we can improve an algorithm.

For these purposes we mainly observe two indicators:

- the value/cost of the heuristic solution;
- the computing time needed to compute it.
Designing experiments

Since it is impossible to test an algorithm on all possible instances of a problem, it is necessary to define an experimental data-set.

It should include

- instances of different size
- instances with different parameterized characteristics (e.g. graphs with different density, degree, diameter, . . .)
- instances of different type
  - of application: logistics, telecommunications, production, . . .
  - of generation method: realistic, artificial, transformed from other problems, . . .
  - of probability distribution: uniform, normal, exponential, . . .

If possible, we should use instances for which the optimal value is known; otherwise, we must compute at least a dual bound.
Doing reproducible experiments

A scientific approach requires that experimental results be reproducible.

- **Instances:**
  - already available (known benchmarks) or
  - newly generated and made publicly available.

- The description of the **algorithm** must be exhaustive and precise and it must include the **parameter setting**.

- The description of the **experiments** must specify
  - the **software implementation details**, such as the **programming language** and the **compiler**;
  - the **hardware** used (processor, parallelism, ...);
  - the **operating system**;
  - the amount of available **memory** (RAM).

It may be very difficult to reproduce other researchers’ results. Every effort must be done to make the results reproducible, including making the **source code** available to others.
Analyzing the results: solution quality

We consider the execution of the algorithm

- on a particular instance,
- with a particular setting of the parameters,
- with a particular seed of a pseudo-random numbers generator (if any),

as an experiment, whose outcome is the solution cost.

We repeat the experiments and we collect the results.

Then we try to build a model, i.e. to give a synthetic and formal description of the results and their relationship with the instances and the parameters.

For this purpose we use concepts from statistics.
Solution Quality Distribution \textit{(SQD)}

The most commonly used indicators are

- the average value of the relative error,
- the variance of the relative error

Drawbacks:

- the results are often affected by a few “bad” results.
- the relative error can only be estimated when we do not know the optimal value.

The first drawback is (partially) tackled by comparing different algorithms, instead of evaluating a single algorithm.
Solution Quality Distribution (SQD)

A distribution function of the relative error $\delta^A(I)$ indicates the frequency with which $\delta^A(I) \leq \alpha$ for any $\alpha \in \mathbb{R}$. 

![Cumulative frequency distribution graph](image)
Solution Quality Distribution (SQD)

For any algorithm $A$, the distribution function of the error $\delta^A(I)$
- is null for $\alpha < 0$;
- tends to 1 for $\alpha \to +\infty$;
- is piece-wise constant, because $I$ is a discrete set;
- is monotone non-decreasing.

If $A$ is an exact optimization algorithm, then it is a step function, equal to 1 for all $\alpha \geq 0$.
If $A$ is a $K$-approximating algorithm, then it is equal to 1 for any $\alpha \geq K$.

We would like to know the distribution of the probability of the outcomes of the experiments.
What we know from the experiments is the distribution of the frequencies of the results observed.

For larger and more representative data-sets, frequencies tend to reproduce probabilities more and more reliably.
The experimental $SQD$ diagram
Parametric SQD diagrams

Given the difficulty in determining a suitable data-set, the diagrams are often parameterized with respect to a (possibly) significant parameter.

In this way we investigate how the performances depend on the chosen parameter.
Descriptive statistics and box-plots

Another compact graphical representation is provided by descriptive statistics such as

- the average value and the variance (or the mean square error)
- the median and the quantiles
- the box-plot diagram (also called box and whiskers plot), that reports
  - the median value ($q_{0.5}$)
  - the first and last quartiles ($q_{0.25}$ e $q_{0.75}$)
  - the extreme values ("outliers" are often excluded)
Box-plot diagrams

- Median
  - One half of the sample
  - The other half of the sample

- Lower Quartile
  - One quarter of the sample
  - Middle half of the sample

- Upper Quartile
  - One quarter of the sample

Minimum: Smallest sample value

Maximum: Largest sample value

Interquartile Range: box length

Whiskers: extend to the minimum and maximum values that are not outliers.

Outliers: any points outside the whiskers.
Comparing algorithms through \textit{SQD} diagrams

When we compare two or more algorithms (and parameter settings) on the same data-set, the following cases may occur:

- \textbf{strict dominance}: algorithm $A'$ always computes better solutions than algorithm $A''$

\[
\delta_{A'}(I) \leq \delta_{A''}(I) \quad \text{for each } I \in \mathcal{I}
\]

This usually happens only in trivial cases, i.e. when $A''$ is a simplified version of $A'$.

- \textbf{probabilistic dominance}: The distribution function of $A'$ is always above that of $A''$

\[
F_{A'}(\alpha) \geq F_{A''}(\alpha) \quad \text{for each } \alpha \in \mathbb{R}
\]

- \textbf{none of the above}: there is no domination. Further investigation is needed to assess when $A'$ is better or worse than $A''$ (for which parameter choice, for which size, for which type of instances,…).
Comparison through SQD diagrams
Comparison through *box-plot* diagrams

Two or more algorithms can also be compared through their *box-plots*.

If two diagrams are completely separated, then there is strict dominance (the converse is not necessary true).

Probabilistic dominance can be argued from the quantiles \((q_0, q_{0.25}, q_{0.5}, q_{0.75}, q_1)\).
Diagrams are only empirical: we would like to know whether the difference in performance between two algorithms is significant or it is just due to statistical fluctuations. We can compare

- distributions with the Kolmogorov-Smirnov test
- median values with the Wilcoxon test.
The Wilcoxon test

The median values $q_{0.5}$ of the relative errors of the two algorithms, $\delta_{A'}(I)$ and $\delta_{A''}(I)$ are considered as random variables defined in the space of the instances $\mathcal{I}$.

- We formulate the null hypothesis $H_0$: the two probability distributions have the same median (i.e. there is not reason to prefer any of the two algorithms).
- We select a significance level $\alpha$, representing the maximum acceptable probability of refusing $H_0$ if it is true (i.e. of considering an algorithm preferable to the other, even if it is not). Typical values are $\alpha = 5\%$ or $\alpha = 1\%$.
- We extract a sample $B$ of instances and we run the two algorithms.
- We do the Wilcoxon test and we obtain the probability $p$ of the observed relation between the results, assuming $H_0$ is true.
- If $p < \alpha$, we refuse $H_0$ (i.e. we state that there is a significant statistical dominance between them).
The Wilcoxon test (assumptions)

The Wilcoxon test is non-parametric, i.e. it makes no assumptions on the probability distributions (this is good because we do not know the probability distribution of $\delta^A(I)$).

Basic assumptions:

- data are paired and they come from the same input data (i.e. the results are referred to the same instances);
- each pair is independent of the others (i.e. instances are generated independently);
- data are ordinal (i.e. values are not important: their relative order is).
The Wilcoxon test (execution)

The Wilcoxon test does the following:

1. compute the absolute differences $|\delta^A'(I_i) - \delta^A''(I_i)|$ for each instance $i = 1, \ldots, k$;
2. sort the differences in non-decreasing order and assign a rank $R_i$ to each of them; in case of ties, assign the tied pairs their average rank;
3. sum the ranks of the pairs with positive difference and the pairs with negative differences.

$$\begin{cases} W^+ = \sum_{i: \delta^A'(I_i) > \delta^A''(I_i)} R_i \\ W^- = \sum_{i: \delta^A'(I_i) < \delta^A''(I_i)} R_i \end{cases}$$

If $H_0$ were true, than the two sums should coincide.

4. the absolute difference between $W^+$ and $W^-$ is compared with a value $W_\alpha$ from a table, which depends on the significance level $\alpha$ we have selected.

5. if $|W^+ - W^-| > W_\alpha$, then the difference is significant and $H_0$ is rejected.
The Wilcoxon test: an example

<table>
<thead>
<tr>
<th>i</th>
<th>$\delta^A$</th>
<th>$\delta^{A''}$</th>
<th>sgn</th>
<th>abs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.2%</td>
<td>3.7%</td>
<td>+</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>3.8%</td>
<td>3.9%</td>
<td>-</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>6.1%</td>
<td>5.4%</td>
<td>+</td>
<td>0.7</td>
</tr>
<tr>
<td>4</td>
<td>0.2%</td>
<td>0.7%</td>
<td>-</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>0.4%</td>
<td>0.1%</td>
<td>+</td>
<td>0.3</td>
</tr>
<tr>
<td>6</td>
<td>1.4%</td>
<td>2.5%</td>
<td>-</td>
<td>1.1</td>
</tr>
<tr>
<td>7</td>
<td>2.5%</td>
<td>2.6%</td>
<td>-</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>2.6%</td>
<td>1.7%</td>
<td>+</td>
<td>0.9</td>
</tr>
<tr>
<td>9</td>
<td>3.6%</td>
<td>3.0%</td>
<td>+</td>
<td>0.6</td>
</tr>
<tr>
<td>10</td>
<td>3.0%</td>
<td>3.5%</td>
<td>-</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>sgn</th>
<th>abs</th>
<th>rank</th>
<th>sgn rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-</td>
<td>0.1</td>
<td>1.5</td>
<td>-1.5</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>0.1</td>
<td>1.5</td>
<td>-1.5</td>
</tr>
<tr>
<td>5</td>
<td>+</td>
<td>0.3</td>
<td>3</td>
<td>+ 3</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>0.5</td>
<td>5</td>
<td>+ 5</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>0.5</td>
<td>5</td>
<td>- 5</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>0.5</td>
<td>5</td>
<td>- 5</td>
</tr>
<tr>
<td>9</td>
<td>+</td>
<td>0.6</td>
<td>7</td>
<td>+ 7</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>0.7</td>
<td>8</td>
<td>+ 8</td>
</tr>
<tr>
<td>8</td>
<td>+</td>
<td>0.9</td>
<td>9</td>
<td>+ 9</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>1.1</td>
<td>10</td>
<td>-10</td>
</tr>
</tbody>
</table>

$$|W^+| = 3 + 5 + 8 + 9 = 25 \quad |W^-| = 1.5 + 1.5 + 5 + 5 + 10 = 23$$

$$|W^+ - W^-| = |25 - 23| = 2$$
Comparing algorithms according to the computing time

The comparison between algorithms that require different computing time is in general unfair, unless time is not a scarce resource.

An algorithm dominates another when

1. provides better solutions and
2. requires less computing time.

Usually, a trade-off is observed between accuracy and speed.

Also the computing time $T(I)$ can be seen as a random variable in the space $\mathcal{I}_n$ of the instances: the distribution function yields the Run Time Distribution (RTD) diagram.
Analysis of the computing time (RTD diagram)

Meaningful diagrams are obtained only by keeping the size $n$ and other “relevant parameters” fixed. “Relevant parameters” are those suggested by the worst-case analysis, i.e. those that significantly affect the computing time.

In the (ideal) case in which all relevant parameters were known, the RTD diagram would degenerate into a step function with threshold value $E[T(I)|I \in I_n]$. 
Analysis of the computing time (*scaling* diagrams)

The growth rate of \( E[T_n] \) as a function of \( n \) can turn out to be much better than the estimate provided by the worst-case analysis (the theoretical time complexity of the algorithm).

To experimentally study the computing time of an algorithm we need:

- to select a sequence \( B_n \) of instances with different values of \( n \);
- run the algorithm and compute the average values of the computing time \( T_n \) for each \( n \);
- \( \sum_{I \in B_n} T(I) \);
- plot the points \( (n, \frac{\sum_{I \in B_n} T(I)}{|B_n|}) \);
- assume a type of interpolating function;
- find the optimal parameters of that function.
Analysis of the computing time (scaling diagrams)
Relationship between quality and computing time

In general it is not fair to compare slow and accurate algorithms with fast and inaccurate algorithms.

Many algorithms (typically local search algorithms) find multiple solutions during their execution and the best one is not necessarily the last one.

For some algorithms the computing time can be arbitrarily fixed.

We can plot the best value found as a function of the elapsed computing time.

Let $\delta^A(I, t)$ the relative difference corresponding to the best solution found by $A$ at time $t$.

- we set $\delta^A(I, t) = +\infty$ when no feasible solutions have been found at time $t$;
- $\delta^A(I, t)$ is a monotone non-increasing function of $t$;
- when the execution of $A$ is over, $\delta^A(I, t)$ remains constant.
Classification

The asymptotic (when \( t \to \infty \)) relationship between the relative error and the computing time allows classifying algorithms as follows:

- **complete**: for each instance \( I \in \mathcal{I} \) \( A \) finds the optimal solution in finite time

  \[
  \exists \bar{t}_i \in \mathbb{R}^+ : \delta^A(I, t) = 0 \quad \forall t \geq \bar{t}_i, I \in \mathcal{I}
  \]

  Example: exact optimization algorithms.

- **probabilistically approximately complete**: for each instance \( I \in \mathcal{I} \), the probability that \( A \) find the optimal solution tends to 1 for \( t \to \infty \)

  \[
  \lim_{t \to \infty} Pr[\delta^A(I, t) = 0] = 1 \quad \forall I \in \mathcal{I}
  \]

  Example: simulated annealing and other randomized meta-heuristics.

- **incomplete**: for some instances \( I \in \mathcal{I} \), the probability that \( A \) find an optimal solution is strictly \( < 1 \), even for \( t \to \infty \)

  \[
  \exists l \in \mathcal{I} : \lim_{t \to \infty} Pr[\delta^A(l, t) = 0] < 1
  \]

  Example: many constructive algorithms.
An obvious generalization is obtained when replacing the search for the optimum by the search for a certain approximation factor, i.e. replacing $\delta^A(I, t) = 0$ with $\delta^A(I, t) \leq \alpha$.

- **$\alpha$-complete** algorithms: for each instance they find an $\alpha$-approximating solution in finite time. Example: constructive heuristics with approximation guarantee.
- **probabilistically approximately $\alpha$-complete**: for each instance $I \in \mathcal{I}$, the probability to find an $\alpha$-approximating solution tends to 1 for $t \to +\infty$;
- **$\alpha$-incomplete** algorithms: for some instances $I \in \mathcal{I}$, the probability to find an $\alpha$-approximating solution is strictly $< 1$ for $t \to +\infty$.

Now we can (experimentally) characterize the algorithms according to the (observed) trade-off between $t$ and $\alpha$. 
We define as success probability $\pi_{A,n}(\alpha, t)$ the probability that algorithm $A$ find a solution of quality within $\alpha$ within time $t$.

$$\pi_{A,n}(\alpha, t) = Pr[\delta^A(I, t) \leq \alpha | I \in \mathcal{I}_n]$$
The **QRTD diagrams** describe how much time is needed to reach a given approximation factor.

They are useful when time is not a scarce resource. If the algorithm is complete, the diagram is the same for all $\alpha$. If the algorithm is $\bar{\alpha}$-approximating, the diagram is the same for all $\alpha \geq \bar{\alpha}$ and it coincides with the RTD diagram.
Solution Quality Distribution (SQD) diagrams

The SQD diagrams describe the approximation factor reached after a given amount of time.

They are useful when time is a scarce resource. If the algorithm is complete, the diagram tends to a step function with threshold $\alpha = 0$ for $t \to \infty$. 
Finally we can draw the level lines associated with the quantiles. They describe the trade-off between accuracy and time. If an algorithm is robust, the lines are close to one another.