CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Cybernetics

2. Empirical analysis and comparisons of stochastic optimization algorithms

Petr Pošík

Substantial part of this material is based on slides provided with the book
'Stochastic Local Search: Foundations and Applications'
by Holger H. Hoos and Thomas Stützle (Morgan Kaufmann, 2004)
See `www.sls-book.net` for further information.

Contents

- No-Free-Lunch Theorem
- What is so hard about the comparison of stochastic methods?
- Simple statistical comparisons
- Comparisons based on running length distributions

Motivation

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.
- The meaning of the adage: *It is impossible to get something for nothing.*

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.
- The meaning of the adage: *It is impossible to get something for nothing.*
- If something appears to be free, there is always a cost to the person or to society as a whole even though that *cost may be hidden or distributed*.

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.
- The meaning of the adage: *It is impossible to get something for nothing.*
- If something appears to be free, there is always a cost to the person or to society as a whole even though that *cost may be hidden or distributed*.

No-Free-Lunch theorem in search and optimization [WM97]

- Informally, for discrete spaces: "Any two algorithms are equivalent when their performance is averaged across all possible problems."

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.
- The meaning of the adage: *It is impossible to get something for nothing.*
- If something appears to be free, there is always a cost to the person or to society as a whole even though that *cost may be hidden or distributed*.

No-Free-Lunch theorem in search and optimization [WM97]

- Informally, for discrete spaces: "Any two algorithms are equivalent when their performance is averaged across all possible problems."
- For a particular problem (or a particular class of problems), different search algorithms may obtain different results.

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.
- The meaning of the adage: *It is impossible to get something for nothing.*
- If something appears to be free, there is always a cost to the person or to society as a whole even though that *cost may be hidden or distributed*.

No-Free-Lunch theorem in search and optimization [WM97]

- Informally, for discrete spaces: "Any two algorithms are equivalent when their performance is averaged across all possible problems."
- For a particular problem (or a particular class of problems), different search algorithms may obtain different results.
- If an algorithm achieves superior results on some problems, it must pay with inferiority on other problems.

No-Free-Lunch Theorem

"There is no such thing as a free lunch."

- Refers to the nineteenth century practice in American bars of offering a "free lunch" with drinks.
- The meaning of the adage: *It is impossible to get something for nothing.*
- If something appears to be free, there is always a cost to the person or to society as a whole even though that *cost may be hidden or distributed*.

No-Free-Lunch theorem in search and optimization [WM97]

- Informally, for discrete spaces: "Any two algorithms are equivalent when their performance is averaged across all possible problems."
- For a particular problem (or a particular class of problems), different search algorithms may obtain different results.
- If an algorithm achieves superior results on some problems, it must pay with inferiority on other problems.

It makes sense to study which algorithms are suitable for which kinds of problems!!!

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1):67–82, 1997.

Monte Carlo vs. Las Vegas Algorithms

EOA belong to the class of *Monte Carlo* or *Las Vegas algorithms* (LVAs):

Monte Carlo vs. Las Vegas Algorithms

EOA belong to the class of *Monte Carlo* or *Las Vegas algorithms* (LVAs):

- **Monte Carlo algorithm:** It always stops and provides a solution, but the solution may not be correct. The solution quality is a random variable.

Monte Carlo vs. Las Vegas Algorithms

EOA belong to the class of *Monte Carlo* or *Las Vegas algorithms* (LVAs):

- **Monte Carlo algorithm:** It always stops and provides a solution, but the solution may not be correct. The solution quality is a random variable.
- **Las Vegas algorithm:** It always produces a correct solution, but needs a priori unknown time to find it. The running time is a random variable.

Monte Carlo vs. Las Vegas Algorithms

EOA belong to the class of *Monte Carlo* or *Las Vegas algorithms* (LVAs):

- **Monte Carlo algorithm:** It always stops and provides a solution, but the solution may not be correct. The solution quality is a random variable.
- **Las Vegas algorithm:** It always produces a correct solution, but needs a priori unknown time to find it. The running time is a random variable.
- LVA can be turned to MCA by bounding the allowed running time.

Monte Carlo vs. Las Vegas Algorithms

EOA belong to the class of *Monte Carlo* or *Las Vegas algorithms* (LVAs):

- **Monte Carlo algorithm:** It always stops and provides a solution, but the solution may not be correct. The solution quality is a random variable.
- **Las Vegas algorithm:** It always produces a correct solution, but needs a priori unknown time to find it. The running time is a random variable.
- LVA can be turned to MCA by bounding the allowed running time.
- MCA can be turned to LVA by restarting the algorithm from randomly chosen states.

Las Vegas algorithms

Las Vegas algorithms:

- An algorithm A for a decision problem class Π is a *Las Vegas algorithm* iff it has the following properties:
  - If A terminates for certain $\pi \in \Pi$ and returns a solution s, then s is guaranteed to be a correct solution of π.
  - For any given instance $\pi \in \Pi$, the runtime of A applied to π, $RT_{A,\pi}$, is a random variable.

Las Vegas algorithms

Las Vegas algorithms:

- An algorithm A for a decision problem class Π is a *Las Vegas algorithm* iff it has the following properties:
  - If A terminates for certain $\pi \in \Pi$ and returns a solution s, then s is guaranteed to be a correct solution of π.
  - For any given instance $\pi \in \Pi$, the runtime of A applied to π, $RT_{A,\pi}$, is a random variable.
- An algorithm A for an optimization problem class Π is an *optimization Las Vegas algorithm* iff it has the following properties:
  - For any given instance $\pi \in \Pi$, the runtime of A applied to π needed to find a solution with certain quality q, $RT_{A,\pi}(q)$, is a random variable.
  - For any given instance $\pi \in \Pi$, the solution quality achieved by A applied to π after certain time t, $SQ_{A,\pi}(t)$, is a random variable.

Las Vegas algorithms

Las Vegas algorithms:

- An algorithm A for a decision problem class Π is a *Las Vegas algorithm* iff it has the following properties:
  - If A terminates for certain $\pi \in \Pi$ and returns a solution s, then s is guaranteed to be a correct solution of π.
  - For any given instance $\pi \in \Pi$, the runtime of A applied to π, $RT_{A,\pi}$, is a random variable.
- An algorithm A for an optimization problem class Π is an *optimization Las Vegas algorithm* iff it has the following properties:
  - For any given instance $\pi \in \Pi$, the runtime of A applied to π needed to find a solution with certain quality q, $RT_{A,\pi}(q)$, is a random variable.
  - For any given instance $\pi \in \Pi$, the solution quality achieved by A applied to π after certain time t, $SQ_{A,\pi}(t)$, is a random variable.
- LVAs are typically *incomplete* or at most *asymptotically complete*.

Runtime Behaviour for Decision Problems

Definitions:

- A is an algorithm for a class Π of decision problems.
- $P_s\left(RT_{A,\pi} \leq t\right)$ is a probability that A finds a solution for a problem instance $\pi \in \Pi$ in time less than or equal to t.

Runtime Behaviour for Decision Problems

Definitions:

- A is an algorithm for a class Π of decision problems.
- $P_s\left(RT_{A,\pi} \leq t\right)$ is a probability that A finds a solution for a problem instance $\pi \in \Pi$ in time less than or equal to t.

Complete algorithm A can provably solve any solvable decision problem instance $\pi \in \Pi$ *after a finite time*, i.e. A is complete if and only if

$$\forall \pi \in \Pi,\ \exists t_{\max} : P_s\left(RT_{A,\pi} \leq t_{\max}\right) = 1. \tag{1}$$

Runtime Behaviour for Decision Problems

Definitions:

- A is an algorithm for a class Π of decision problems.
- $P_s\left(RT_{A,\pi} \leq t\right)$ is a probability that A finds a solution for a problem instance $\pi \in \Pi$ in time less than or equal to t.

Complete algorithm A can provably solve any solvable decision problem instance $\pi \in \Pi$ *after a finite time*, i.e. A is complete if and only if

$$\forall \pi \in \Pi,\ \exists t_{\max} : P_s\left(RT_{A,\pi} \leq t_{\max}\right) = 1. \tag{1}$$

Asymptotically complete algorithm A can solve any solvable problem instance $\pi \in \Pi$ with arbitrarily high probability *when allowed to run long enough*, i.e. A is asymptotically complete if and only if

$$\forall \pi \in \Pi : \lim_{t \to \infty} P_s\left(RT_{A,\pi} \leq t\right) = 1. \tag{2}$$

Runtime Behaviour for Decision Problems

Definitions:

- A is an algorithm for a class Π of decision problems.
- $P_s\left(RT_{A,\pi} \leq t\right)$ is a probability that A finds a solution for a problem instance $\pi \in \Pi$ in time less than or equal to t.

Complete algorithm A can provably solve any solvable decision problem instance $\pi \in \Pi$ *after a finite time*, i.e. A is complete if and only if

$$\forall \pi \in \Pi,\ \exists t_{\max} : P_s\left(RT_{A,\pi} \leq t_{\max}\right) = 1. \tag{1}$$

Asymptotically complete algorithm A can solve any solvable problem instance $\pi \in \Pi$ with arbitrarily high probability *when allowed to run long enough*, i.e. A is asymptotically complete if and only if

$$\forall \pi \in \Pi : \lim_{t \to \infty} P_s\left(RT_{A,\pi} \leq t\right) = 1. \tag{2}$$

Incomplete algorithm A cannot be guaranteed to find the solution even if allowed to run indefinitely long, i.e. if it is not asymptotically complete, i.e. A is incomplete if and only if

$$\exists \text{ solvable } \pi \in \Pi : \lim_{t \to \infty} P_s\left(RT_{A,\pi} \leq t\right) < 1. \tag{3}$$

Runtime Behaviour for Optimization Problems

Simple generalization based on transforming the optimization problem to related decision problem by setting the solution quality bound to $q = r \cdot q^*(\pi)$:

- A is an algorithm for a class Π of optimization problems.
- $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q)$ is the probability that A finds a solution of quality better than or equal to q for a solvable problem instance $\pi \in \Pi$ in time less than or equal to t.
- $q^*(\pi)$ is the quality of optimal solution to problem π.
- $r \geq 1$, $q > 0$.

Runtime Behaviour for Optimization Problems

Simple generalization based on transforming the optimization problem to related decision problem by setting the solution quality bound to $q = r \cdot q^*(\pi)$:

- A is an algorithm for a class Π of optimization problems.
- $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q)$ is the probability that A finds a solution of quality better than or equal to q for a solvable problem instance $\pi \in \Pi$ in time less than or equal to t.
- $q^*(\pi)$ is the quality of optimal solution to problem π.
- $r \geq 1, q > 0$.

Algorithm A is r-complete if and only if

$$\forall \pi \in \Pi,\ \exists t_{\max} : P_s(RT_{A,\pi} \leq t_{\max}, SQ_{A,\pi} \leq r \cdot q^*(\pi)) = 1. \tag{4}$$

Runtime Behaviour for Optimization Problems

Simple generalization based on transforming the optimization problem to related decision problem by setting the solution quality bound to $q = r \cdot q^*(\pi)$:

- A is an algorithm for a class Π of optimization problems.
- $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q)$ is the probability that A finds a solution of quality better than or equal to q for a solvable problem instance $\pi \in \Pi$ in time less than or equal to t.
- $q^*(\pi)$ is the quality of optimal solution to problem π.
- $r \geq 1$, $q > 0$.

Algorithm A is r-complete if and only if

$$\forall \pi \in \Pi, \exists t_{\max} : P_s(RT_{A,\pi} \leq t_{\max}, SQ_{A,\pi} \leq r \cdot q^*(\pi)) = 1. \tag{4}$$

Algorithm A is asymptotically r-complete if and only if

$$\forall \pi \in \Pi : \lim_{t\to\infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq r \cdot q^*(\pi)) = 1. \tag{5}$$

Runtime Behaviour for Optimization Problems

Simple generalization based on transforming the optimization problem to related decision problem by setting the solution quality bound to $q = r \cdot q^*(\pi)$:

- A is an algorithm for a class Π of optimization problems.
- $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q)$ is the probability that A finds a solution of quality better than or equal to q for a solvable problem instance $\pi \in \Pi$ in time less than or equal to t.
- $q^*(\pi)$ is the quality of optimal solution to problem π.
- $r \geq 1, q > 0$.

Algorithm A is r-complete if and only if

$$\forall \pi \in \Pi,\ \exists t_{\max} : P_s(RT_{A,\pi} \leq t_{\max}, SQ_{A,\pi} \leq r \cdot q^*(\pi)) = 1. \tag{4}$$

Algorithm A is asymptotically r-complete if and only if

$$\forall \pi \in \Pi : \lim_{t \to \infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq r \cdot q^*(\pi)) = 1. \tag{5}$$

Algorithm A is r-incomplete if and only if

$$\exists \text{ solvable } \pi \in \Pi : \lim_{t \to \infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq r \cdot q^*(\pi)) < 1. \tag{6}$$

Some Tweaks

- Incompleteness of many LVAs is typically caused by their inability to escape from attractive local minima regions of the search space.
  - Remedy: use diversification mechanisms such as random restart, random walk, tabu, . . .
  - In many cases, these can render algorithms provably asymptotically complete, but effectiveness in practice can vary widely.
- Completeness can be achived by restarting an incomplete method from a solution generated by a complete (exhaustive) algorithm.
  - Typically very ineffective due to large size of the search space.

Theoretical vs. Empirical Analysis of LVAs

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically. (Algorithms are often non-deterministic.)

Theoretical vs. Empirical Analysis of LVAs

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically. (Algorithms are often non-deterministic.)
- Cases in which theoretical results are available are often of limited practical relevance, because they
  - rely on idealised assumptions that do not apply to practical situations,

Theoretical vs. Empirical Analysis of LVAs

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically. (Algorithms are often non-deterministic.)
- Cases in which theoretical results are available are often of limited practical relevance, because they
  - rely on idealised assumptions that do not apply to practical situations,
  - apply to worst-case or highly idealised average-case behaviour only, or

Theoretical vs. Empirical Analysis of LVAs

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically. (Algorithms are often non-deterministic.)
- Cases in which theoretical results are available are often of limited practical relevance, because they
  - rely on idealised assumptions that do not apply to practical situations,
  - apply to worst-case or highly idealised average-case behaviour only, or
  - capture only asymptotic behaviour and do not reflect actual behaviour with sufficient accuracy.

Theoretical vs. Empirical Analysis of LVAs

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically. (Algorithms are often non-deterministic.)
- Cases in which theoretical results are available are often of limited practical relevance, because they
  - rely on idealised assumptions that do not apply to practical situations,
  - apply to worst-case or highly idealised average-case behaviour only, or
  - capture only asymptotic behaviour and do not reflect actual behaviour with sufficient accuracy.

Therefore, **analyse the behaviour of LVAs using empirical methodology**, ideally based on the *scientific method*:

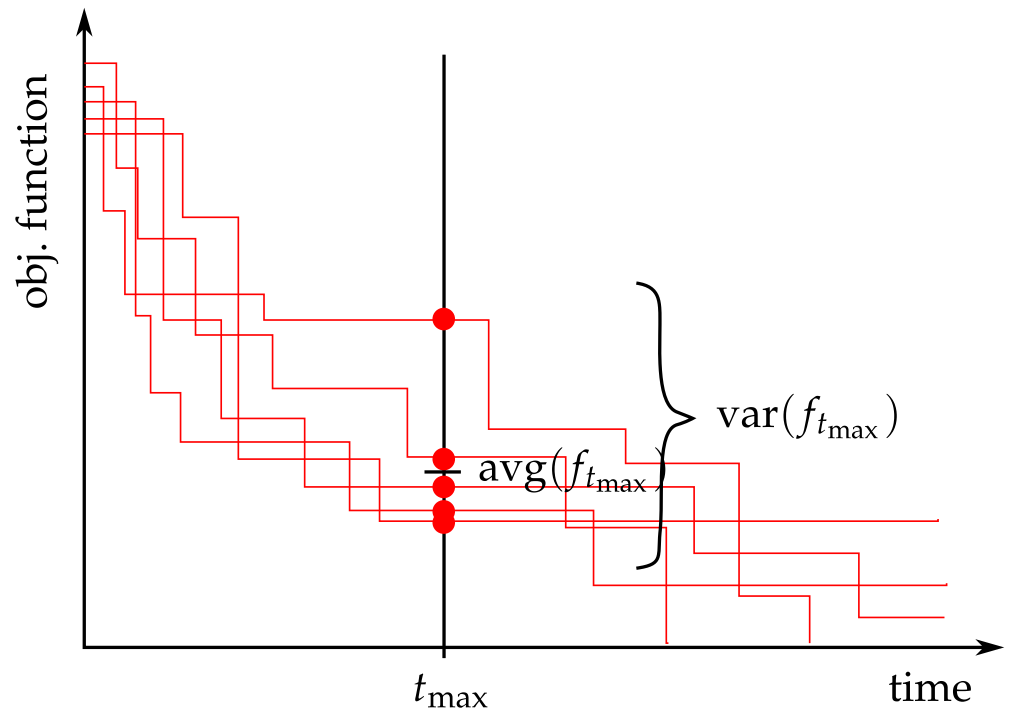
- make observations
- formulate hypothesis/hypotheses (model)
- While not satisfied with model (and deadline not exceeded):
  1. design computational experiment to test model
  2. conduct computational experiment
  3. analyse experimental results
  4. revise model based on results

Application Scenarios and Evaluation Criteria

Type 1: Hard time limit t_{max} for finding solution; solutions found later are useless (real-time environments with strict deadlines, e.g., dynamic task scheduling or on-line robot control).

⇒ Evaluation criterion:

- dec. prob.: solution probability at time t_{max}, $P_s(RT \leq t_{max})$
- opt. prob.: expected quality of the solution found at time t_{max}, $E(SQ(t_{max}))$



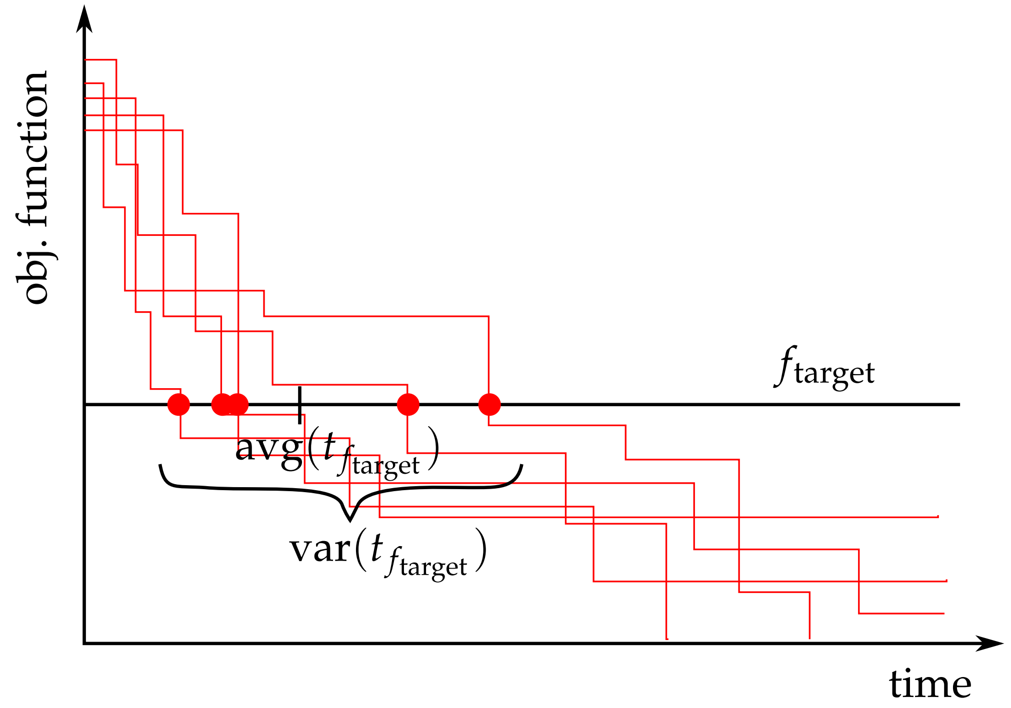
- Possible problem: What does "The expected solution quality of algorithm A is 2 times better than for algorithm B" actually mean?

Application Scenarios and Evaluation Criteria (cont.)

Type 2: No time limits given, algorithm can be run until a solution is found (off-line computations, non-realtime environments, e.g., configuration of production facility).

⇒ Evaluation criterion:

- dec. prob.: expected runtime to solve a problem
- opt. prob.: expected runtime to reach solution of certain quality



- Is there any problem with "The expected runtime of algorithm A is 2 times larger than for algorithm B"?

Application Scenarios and Evaluation Criteria (cont.)

Type 3: Utility of solutions depends in more complex ways on the time required to find them; characterised by a utility function U:

- dec. prob.: $U : R^+ \mapsto \langle 0, 1 \rangle$, where $U(t) =$ utility of solution found at time t
- opt. prob.: $U : R^+ \times R^+ \mapsto \langle 0, 1 \rangle$, where $U(t, q) =$ utility of solution with quality q found at time t

Application Scenarios and Evaluation Criteria (cont.)

Type 3: Utility of solutions depends in more complex ways on the time required to find them; characterised by a utility function U:

- dec. prob.: $U : R^+ \mapsto \langle 0, 1\rangle$, where $U(t)$ = utility of solution found at time t
- opt. prob.: $U : R^+ \times R^+ \mapsto \langle 0, 1\rangle$, where $U(t, q)$ = utility of solution with quality q found at time t

Example: The direct benefit of a solution is invariant over time, but the cost of computing time diminishes the final payoff according to $U(t) = \max\{u_0 - c \cdot t, 0\}$ (constant discounting).

Application Scenarios and Evaluation Criteria (cont.)

Type 3: Utility of solutions depends in more complex ways on the time required to find them; characterised by a utility function U:

- dec. prob.: $U : R^+ \mapsto \langle 0, 1 \rangle$, where $U(t)$ = utility of solution found at time t
- opt. prob.: $U : R^+ \times R^+ \mapsto \langle 0, 1 \rangle$, where $U(t, q)$ = utility of solution with quality q found at time t

Example: The direct benefit of a solution is invariant over time, but the cost of computing time diminishes the final payoff according to $U(t) = \max\{u_0 - c \cdot t, 0\}$ (constant discounting).

$\Rightarrow$ Evaluation criterion: utility-weighted solution probability

- dec. prob.: $U(t) \cdot P_s(RT \leq t)$, or
- opt. prob.: $U(t, q) \cdot P_s(RT \leq t, SQ \leq q)$

requires detailed knowledge of $P_s(\ldots)$ for arbitrary t (and arbitrary q).

Empirical Algorithm Comparison

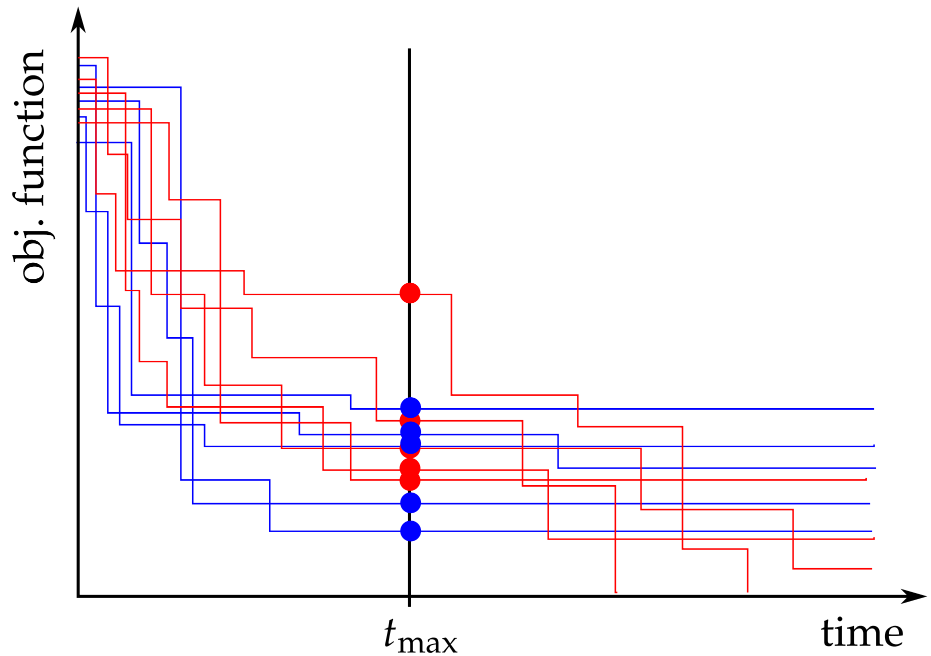
CPU Runtime vs Operation Counts

Remark: Is it better to measure the time in *seconds* or e.g. in *function evaluations*?

- Results of experiments should be comparable.
- Wall-clock time depends on the machine configuration, computer language, and on the operating system used to run the experiments.
- Since the objective function is often the most time-consuming operation in the optimization cycle, many authors use the *number of objective function evaluations* as the primary measure of "time".

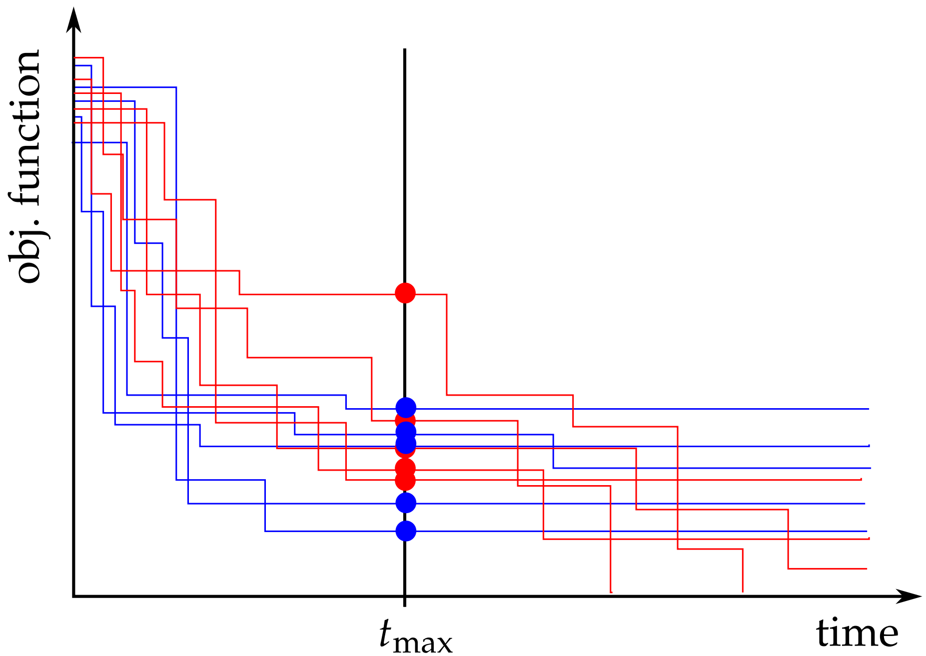
Scenario 1: Limited time

- Let them run for certain time $t_{\max}$ and compare the average quality of returned solution, $\text{ave}(SQ)$



Scenario 1: Limited time

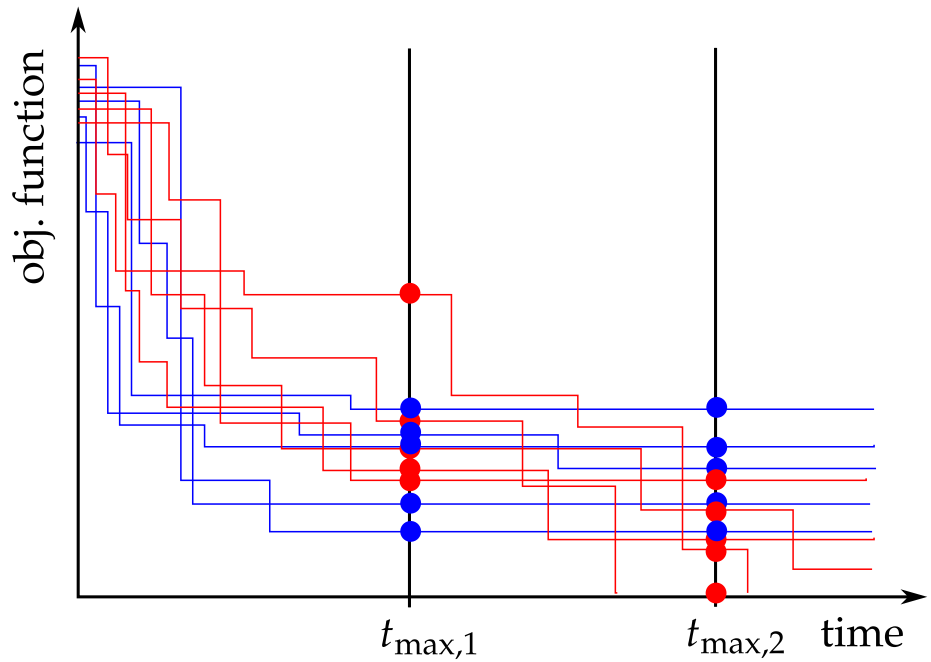
- Let them run for certain time $t_{\max}$ and compare the average quality of returned solution, $\text{ave}(SQ)$



- For $t_{\max,1}$, blue algorithm is better than red.

Scenario 1: Limited time

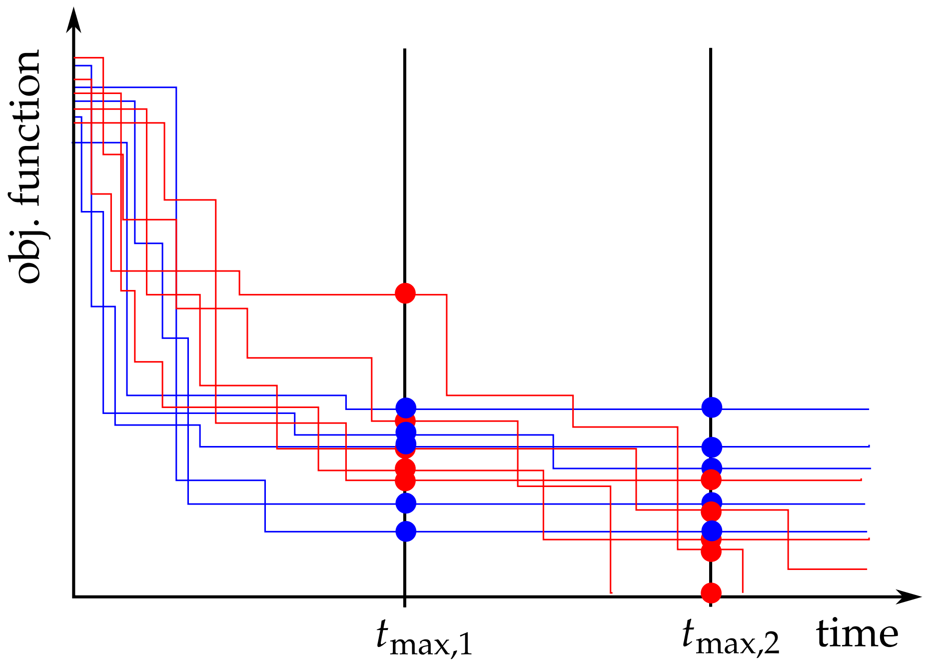
- Let them run for certain time $t_{\max}$ and compare the average quality of returned solution, $\text{ave}(SQ)$



- For $t_{\max,1}$, blue algorithm is better than red.
- For $t_{\max,2}$, blue algorithm is worse than red.
- WARNING! The figure can change when $t_{\max}$ changes!!!

Scenario 1: Limited time

- Let them run for certain time $t_{\max}$ and compare the average quality of returned solution, $\text{ave}(SQ)$



- For $t_{\max,1}$, blue algorithm is better than red.
- For $t_{\max,2}$, blue algorithm is worse than red.
- WARNING! The figure can change when $t_{\max}$ changes!!!
- Can our claims be false? What is the probability that our claims are wrong?

Student's t-test

Independent two-sample t-test:

- Statistical method used to test if the means of 2 normally distributed populations are equal.
- The larger the difference between means, the higher the probability the means are different.
- The lower the variance inside the populations, the higher the probability the means are different.
- For details, see e.g. [Luk09, sec. 11.1.2].
- Implemented in most mathematical and statistical software, e.g. in MATLAB.
- Can be easily implemented in any language.

Assumptions:

- Both populations should have normal distribution.
- Almost never fulfilled with the populations of solution qualities.

Remedy: a non-parametric test!

[Luk09] Sean Luke. *Essentials of Metaheuristics*. 2009. available at http://cs.gmu.edu/~sean/book/metaheuristics/.

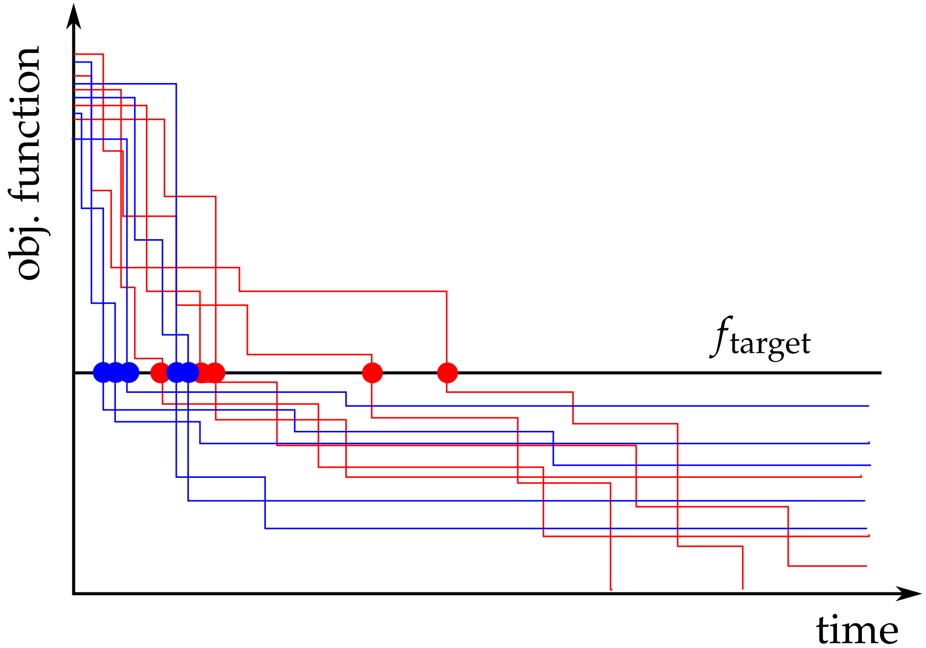
Mann-Whitney-Wilcoxon rank-sum test

Non-parametric test assessing whether two independent samples of observations have equally large values.

- Virtually identical to:
  - combine both samples (for each observation, remember its original group),
  - sort the values,
  - replace the values by ranks,
  - use the ranks with ordinary parametric two-sample t-test.
- The measurements must be at least ordinal:
  - We must be able to sort them.
  - This allows us to merge results from runs which reached the target level with the results of runs which did not.

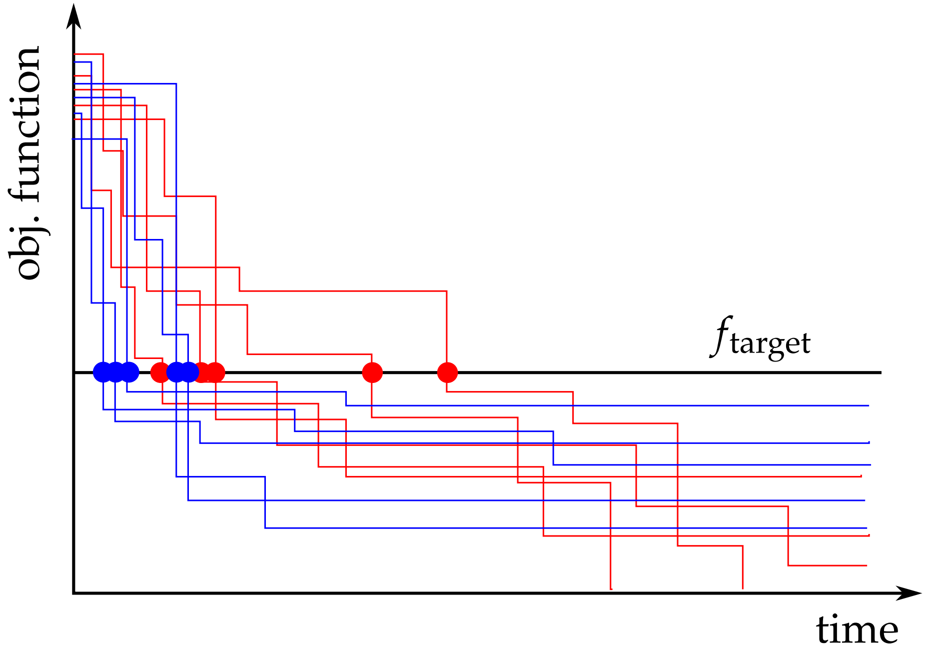
Scenario 2: Prescribed target level

- Let them run until they find a solution of certain quality f_{target} and compare the average runtime, $\text{ave}(RT)$



Scenario 2: Prescribed target level

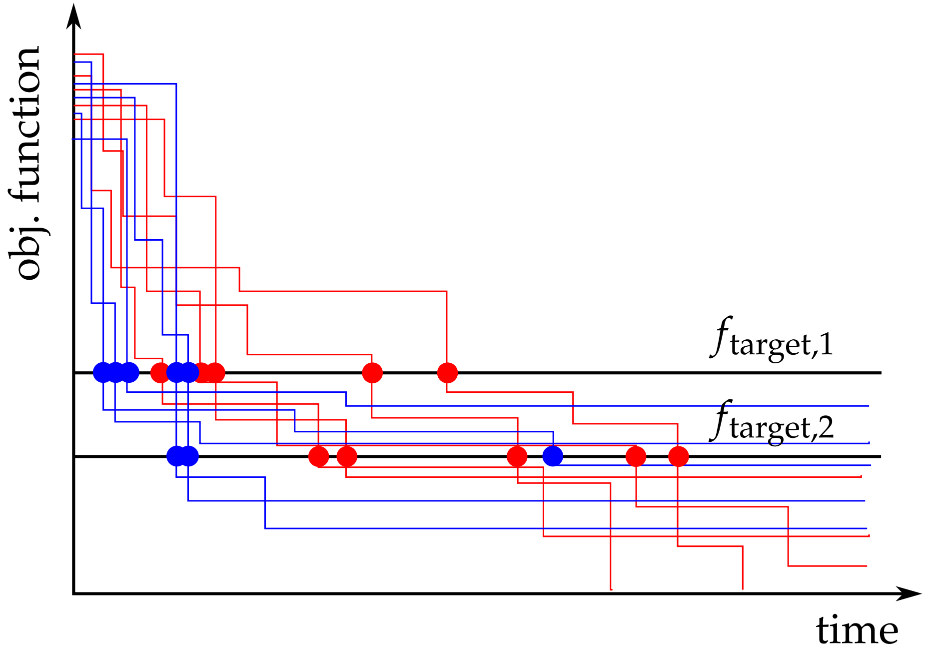
- Let them run until they find a solution of certain quality f_{target} and compare the average runtime, $\text{ave}(RT)$



- For $f_{\text{target},1}$, blue algorithm is better than red.

Scenario 2: Prescribed target level

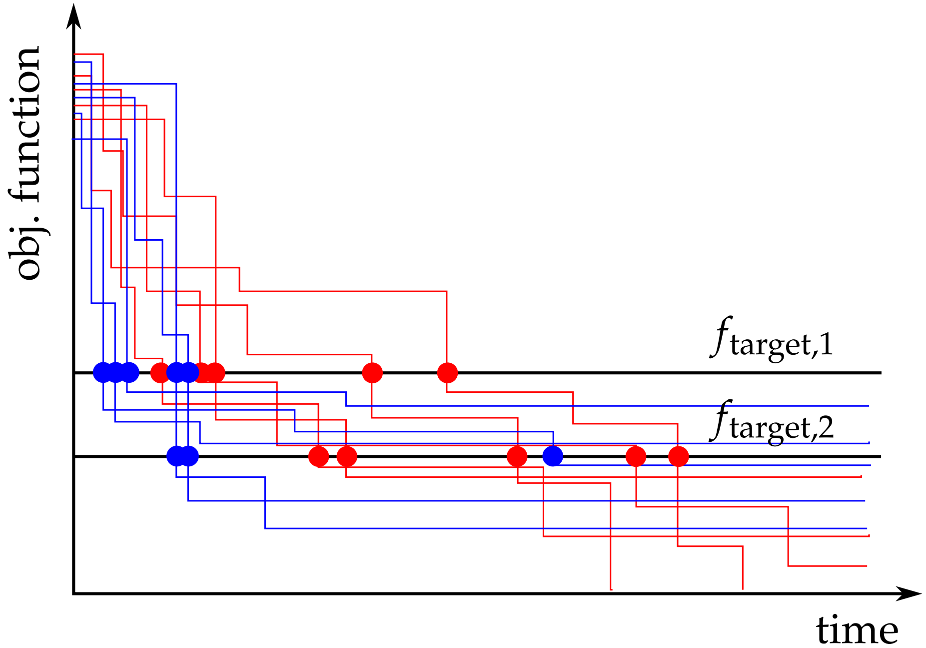
- Let them run until they find a solution of certain quality f_{target} and compare the average runtime, $\text{ave}(RT)$



- For $f_{\text{target},1}$, blue algorithm is better than red.
- For $f_{\text{target},2}$, blue algorithm still seems to better than red (if it finds the solution, it finds it faster), but 2 blue runs did not reach the target level yet, i.e. (we are much less sure that blue is better).
- WARNING! The figure can change when f_{target} changes!!!

Scenario 2: Prescribed target level

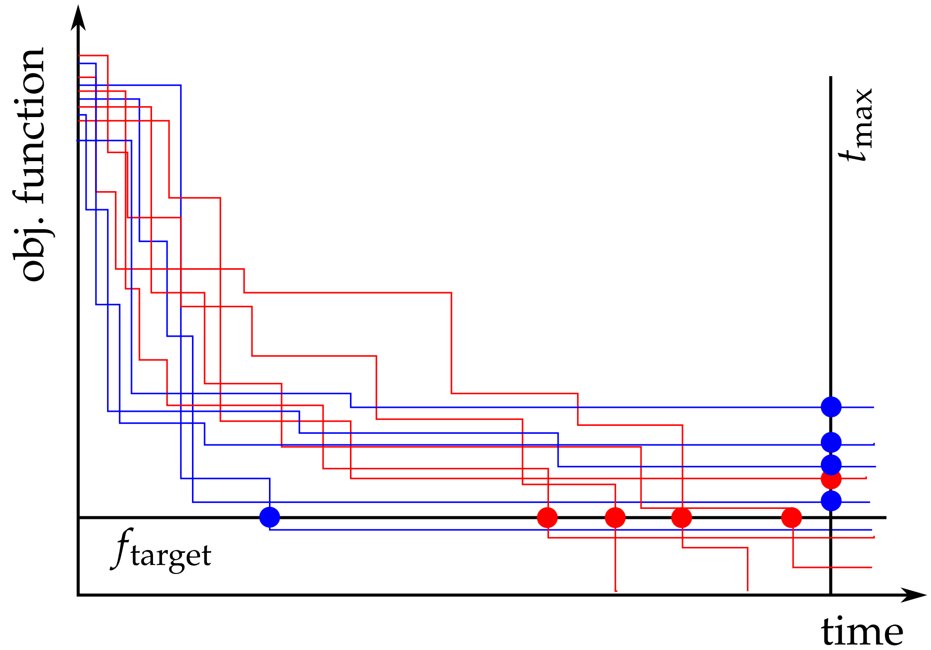
- Let them run until they find a solution of certain quality f_{target} and compare the average runtime, $\text{ave}(RT)$



- For $f_{\text{target},1}$, blue algorithm is better than red.
- For $f_{\text{target},2}$, blue algorithm still seems to better than red (if it finds the solution, it finds it faster), but 2 blue runs did not reach the target level yet, i.e. (we are much less sure that blue is better).
- WARNING! The figure can change when f_{target} changes!!!
- The same statistical tests as for scenario 1 can be used.

Scenarios 1 and 2 combined

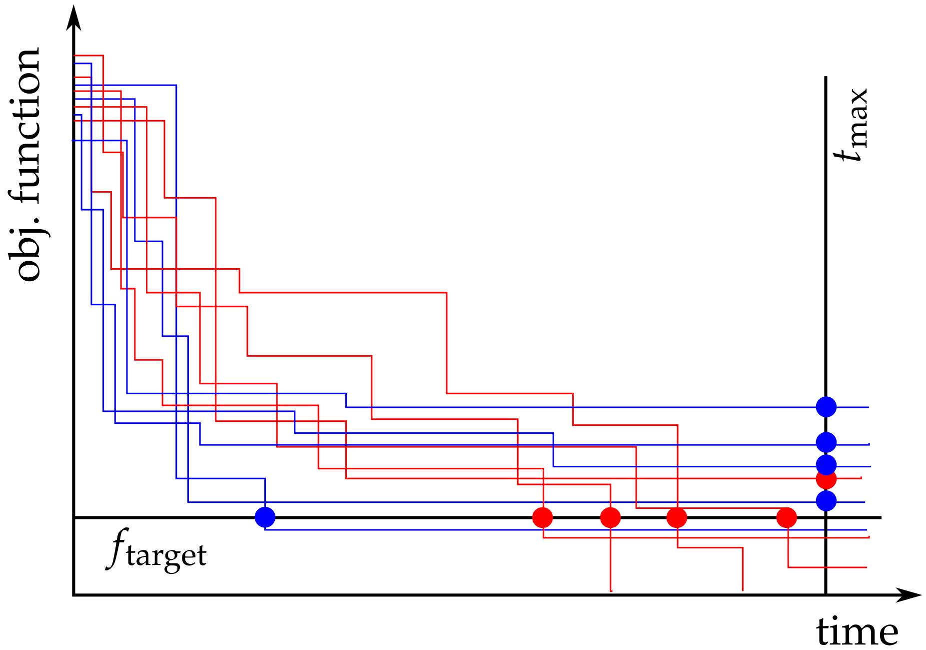
- Let them run until they find a solution of certain quality f_{target} or until they use all the allowed time $t_{\max}$.



- *RT* is measured in seconds or function evaluations, *SQ* is measured in something different; now, how can we test if one algorithm is better than the other?

Scenarios 1 and 2 combined

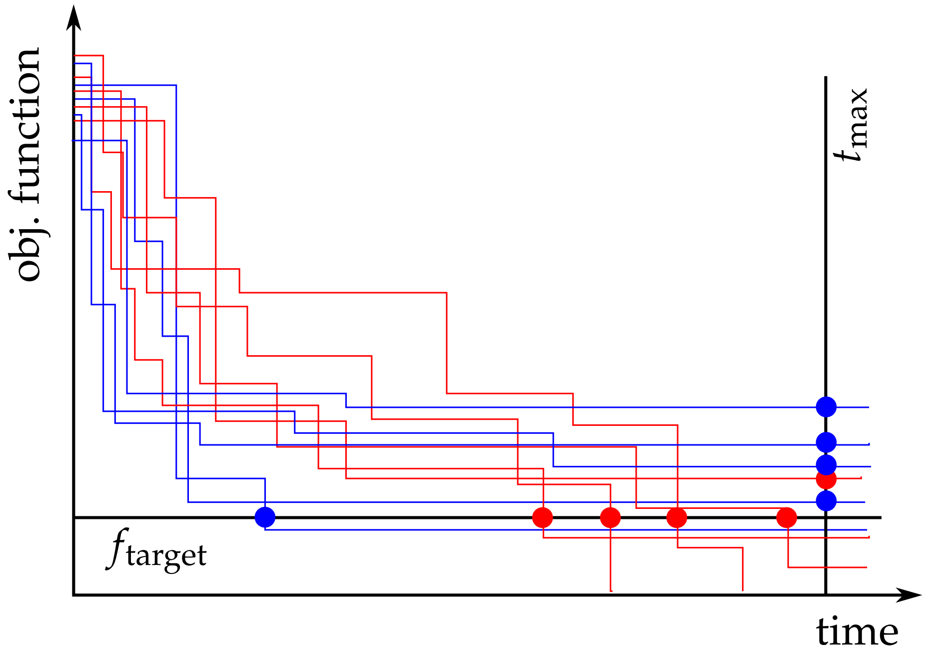
- Let them run until they find a solution of certain quality f_{target} or until they use all the allowed time $t_{\max}$.



- *RT* is measured in seconds or function evaluations, *SQ* is measured in something different; now, how can we test if one algorithm is better than the other?
- The situation when the algorithm reaches f_{target} is better than when it reaches $t_{\max}$. We can still sort the values.
- We can use the Mann-Whitney U-test.

Scenarios 1 and 2 combined

- Let them run until they find a solution of certain quality f_{target} or until they use all the allowed time $t_{\max}$.



- RT is measured in seconds or function evaluations, SQ is measured in something different; now, how can we test if one algorithm is better than the other?
- The situation when the algorithm reaches f_{target} is better than when it reaches $t_{\max}$. We can still sort the values.
- We can use the Mann-Whitney U-test.
- WARNING! Again, if we change f_{target} and/or $t_{\max}$, the figure can change!!!

Analysis based on runtime distribution

Runtime distributions

LVAs are often designed and evaluated without apriori knowledge of the application scenario:

- Assume the most general scenario — type 3 with a utility function (which is often, however, unknown as well).
- Evaluate based on solution probabilities $P_s(RT \leq t, SQ \leq q)$ for arbitrary runtimes t and solution qualities q.

Study distributions of *random variables* characterising runtime and solution quality of an algorithm for the given problem instance.

RTD defintion

Given a Las Vegas alg. A for optimization problem π:

- The *success probability* $P_s\left(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q\right)$ is the probability that A finds a solution for a solvable instance $\pi \in \Pi$ of quality $\leq q$ in time $\leq t$.

RTD defintion

Given a Las Vegas alg. A for optimization problem π:

- The *success probability* $P_s\left(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q\right)$ is the probability that A finds a solution for a solvable instance $\pi \in \Pi$ of quality $\leq q$ in time $\leq t$.
- The *run-time distribution* (RTD) of A on π is the probability distribution of the bivariate random variable $\left(RT_{A,\pi}, SQ_{A,\pi}\right)$.

RTD defintion

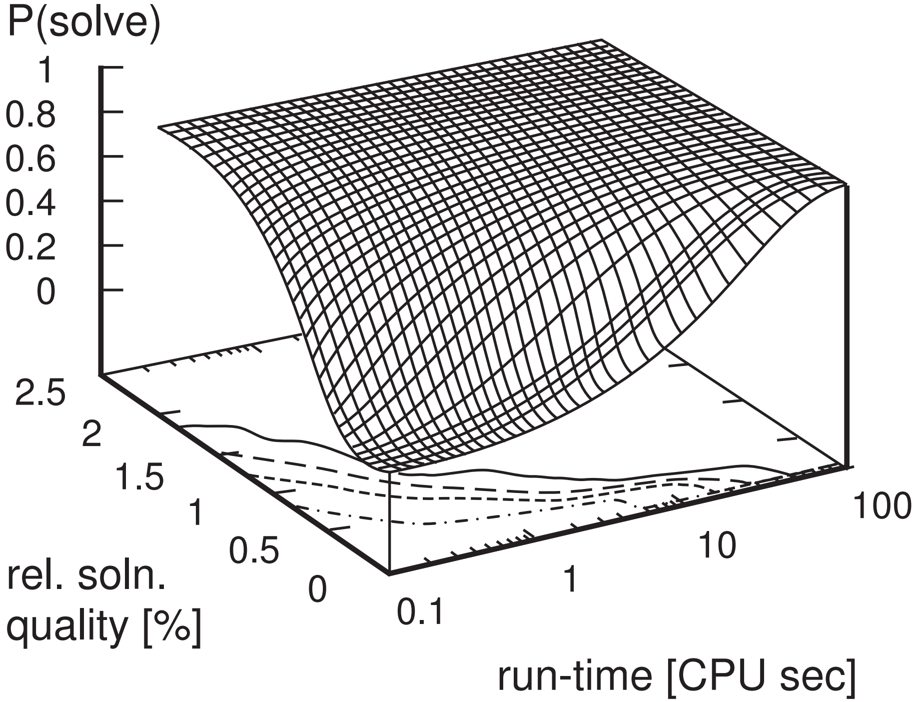
Given a Las Vegas alg. A for optimization problem π:

- The *success probability* $P_s\left(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q\right)$ is the probability that A finds a solution for a solvable instance $\pi \in \Pi$ of quality $\leq q$ in time $\leq t$.
- The *run-time distribution* (RTD) of A on π is the probability distribution of the bivariate random variable $\left(RT_{A,\pi}, SQ_{A,\pi}\right)$.
- The *runtime distribution function* $rtd : R^+ \times R^+ \rightarrow [0, 1]$, defined as $rtd(t, q) = P_s\left(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q\right)$, completely characterises the RTD of A on π.

RTD defintion

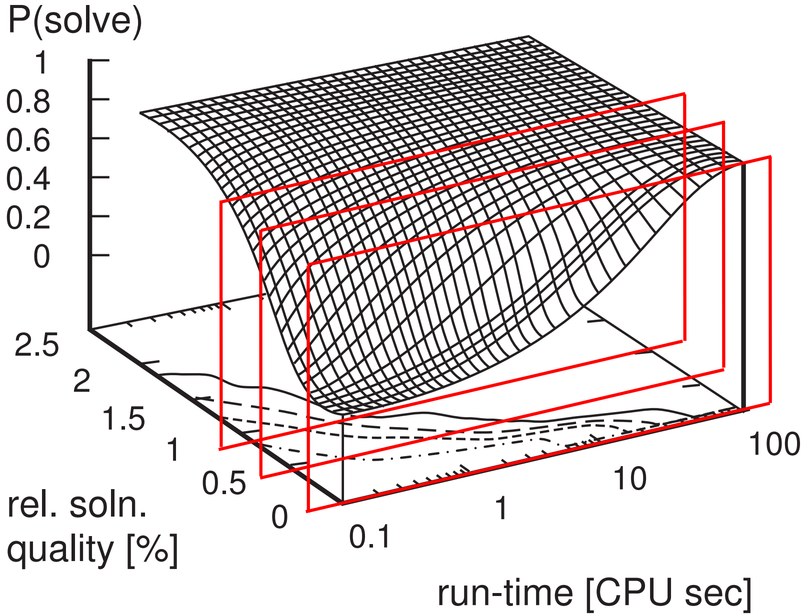
Given a Las Vegas alg. A for optimization problem π:

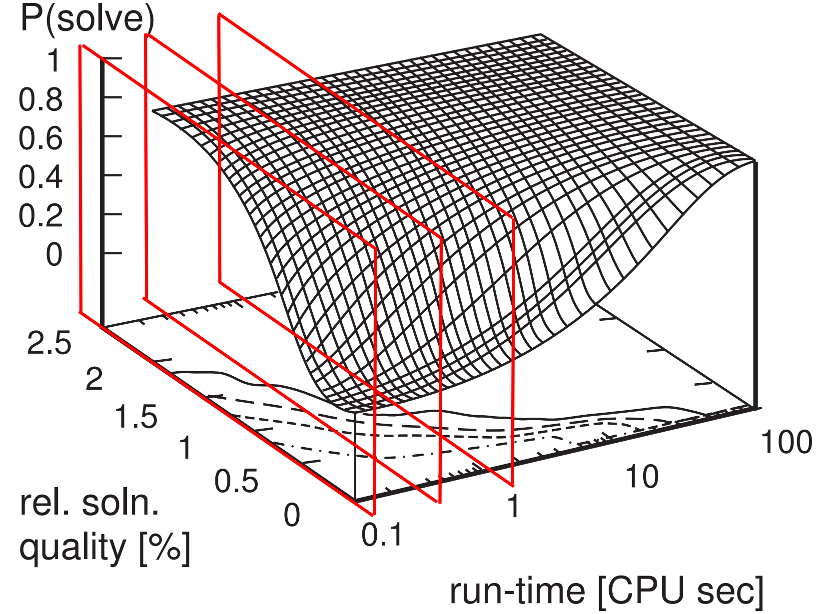
- The *success probability* $P_s\left(RT_{A,\pi} \le t, SQ_{A,\pi} \le q\right)$ is the probability that A finds a solution for a solvable instance $\pi \in \Pi$ of quality $\le q$ in time $\le t$.
- The *run-time distribution* (RTD) of A on π is the probability distribution of the bivariate random variable $(RT_{A,\pi}, SQ_{A,\pi})$.
- The *runtime distribution function* $rtd : R^+ \times R^+ \to [0,1]$, defined as $rtd(t,q) = P_s\left(RT_{A,\pi} \le t, SQ_{A,\pi} \le q\right)$, completely characterises the RTD of A on π.

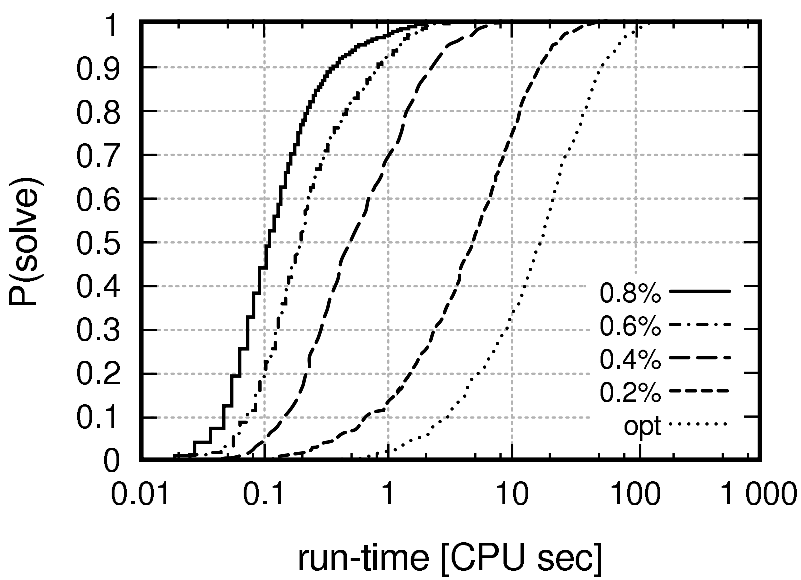


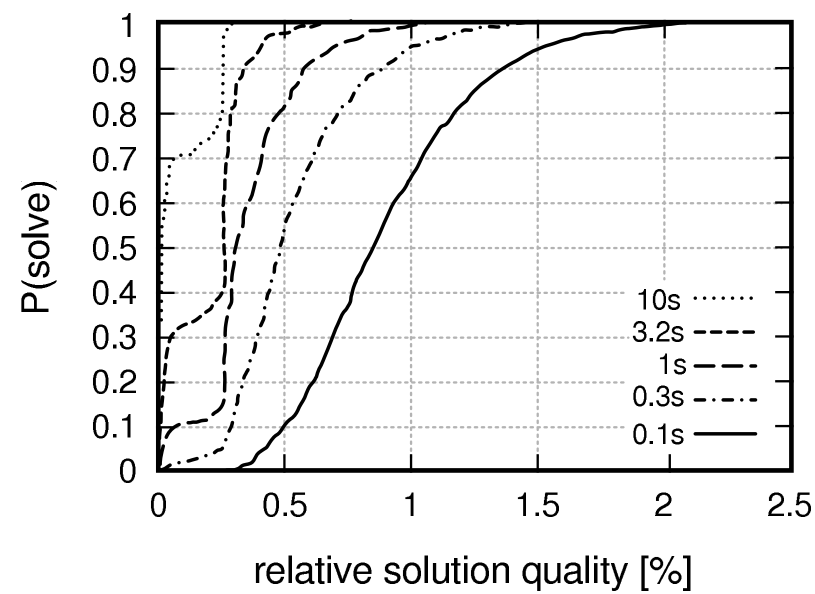
RTD cross-sections

We can study the RTD using cross-sections:



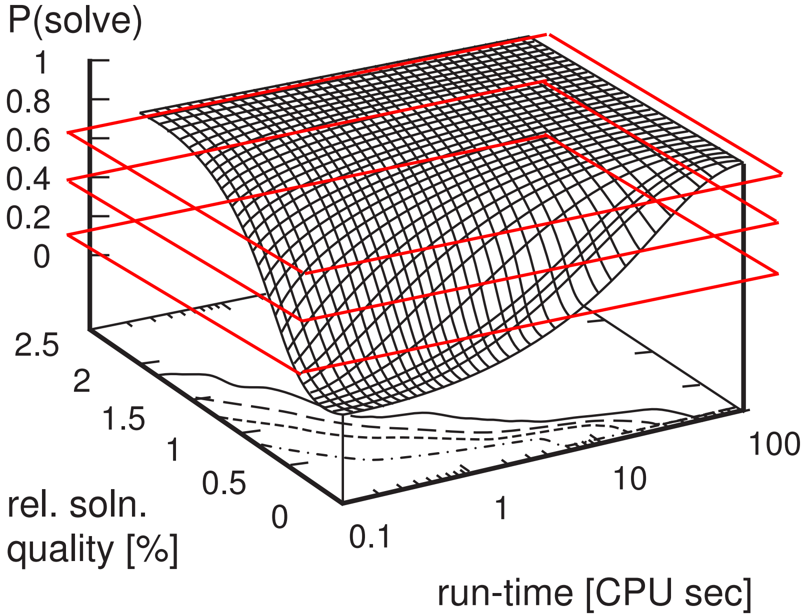


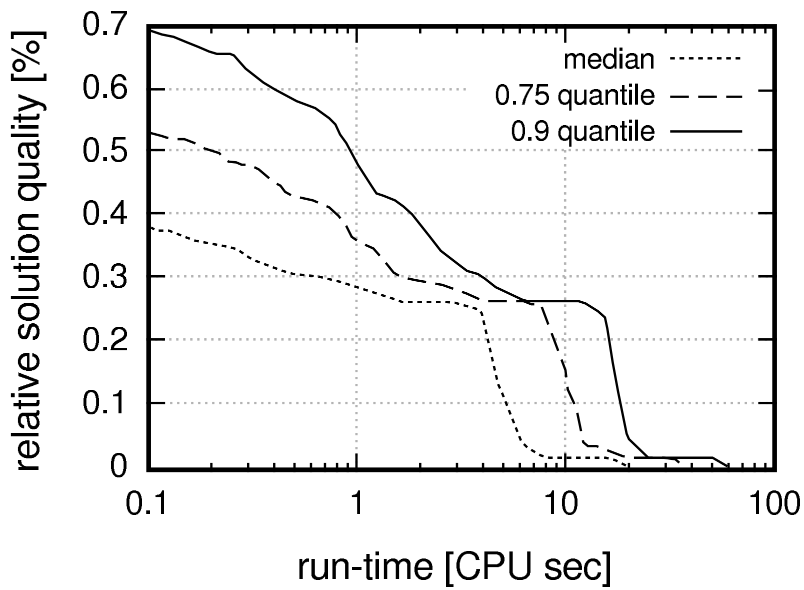




RTD cross-sections (cont.)

We can study the RTD using cross-sections:





Horizontal cross-sections reveal the dependence of SQ on RT:

- The lines represent various quantiles; e.g. for 75%-quantile we can expect that 75% of runs will return a better combination of SQ and RT.

Empirical measurement of RTDs

Empirical estimation of $P_s\left(RT \leq t, SQ \leq q\right)$:

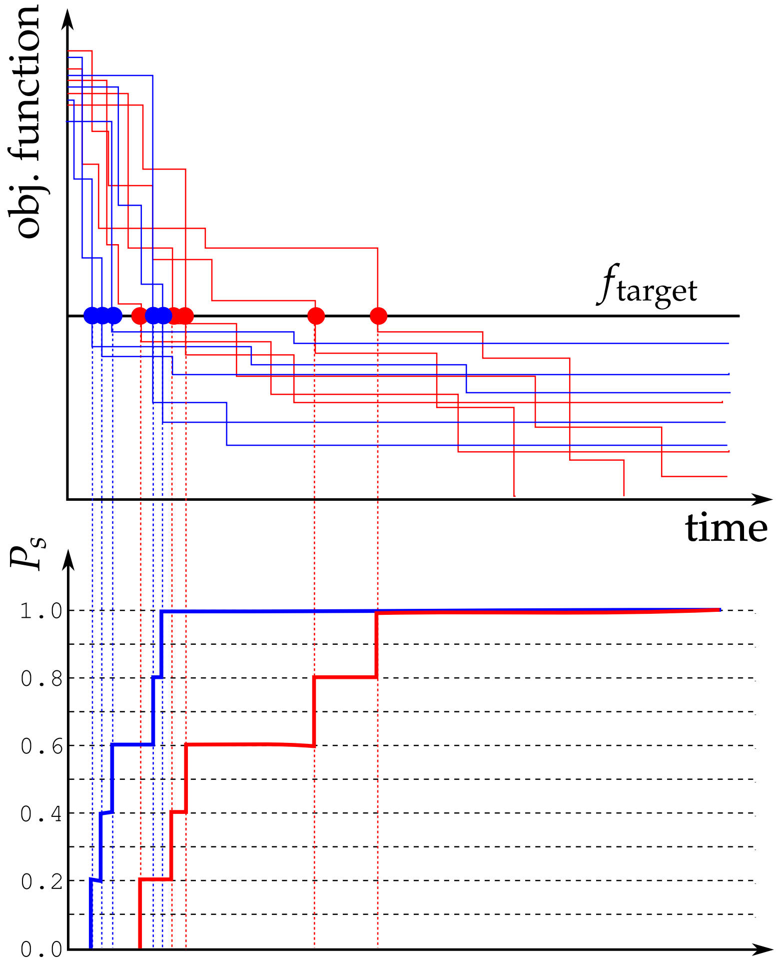
- Perform N independent runs of A on problem π.
- For n^{th} run, $n \in 1, \ldots, N$, store the so-called *solution quality trace*, i.e. $t_{n,i}$ and $q_{n,i}$ each time the quality is improved.
- $\bar{P}_s(t,q) = \frac{n_S(t,q)}{N}$, where $n_S(t,q)$ is the number of runs which provided at least one solution with $t_i \leq t$ and $q_i \leq q$.

Empirical RTDs are approximations of an algorithm's true RTD:

- The larger the N, the better the approximation.

RTD based algorithm comparisons

E.g. type 2 application scenario: set f_{target} and compare RTDs of the algorithms



RTD based algorithm comparisons

Motivation

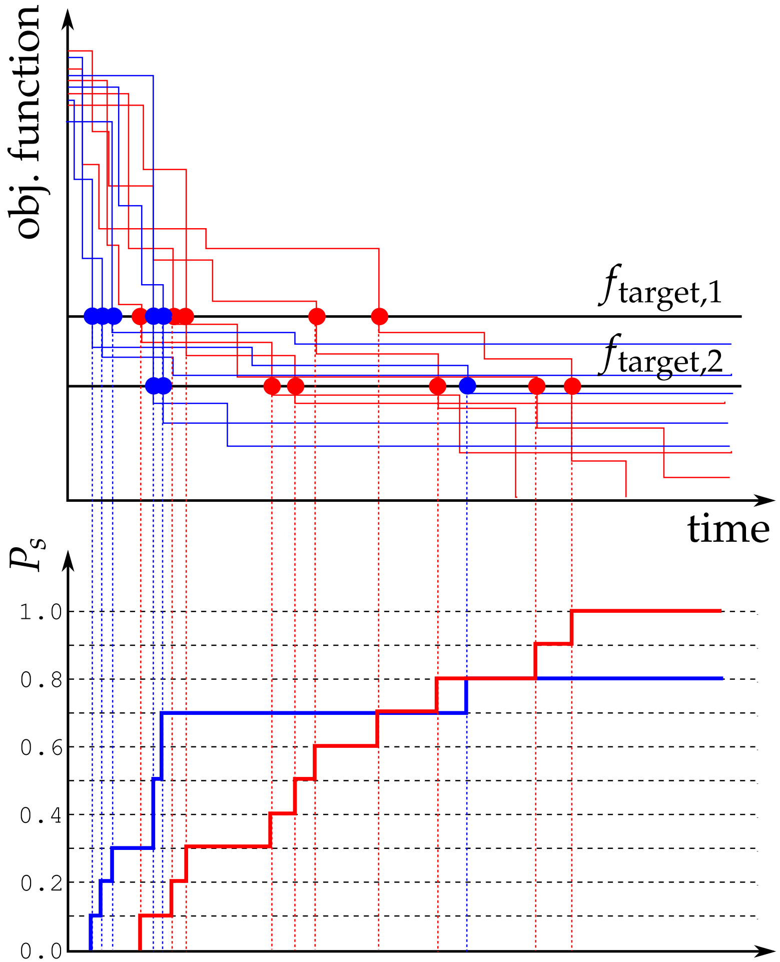
Empirical Algorithm Comparison

Analysis based on runtime distribution
- Runtime distributions
- RTD defintion
- RTD cross-sections
- Empirical measurement of RTDs
- RTD based algorithm comparisons
- Example of comparison

Summary

E.g. type 2 application scenario: set f_{target} and compare RTDs of the algorithms

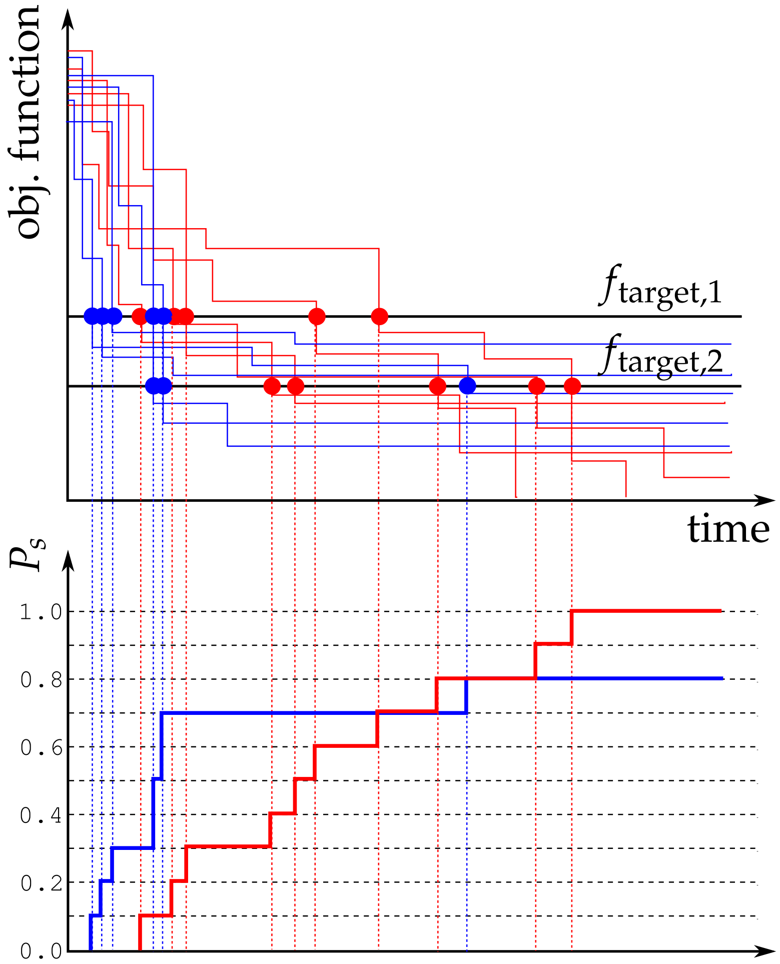
…and add another f_{target} level …



RTD based algorithm comparisons

E.g. type 2 application scenario: set f_{target} and compare RTDs of the algorithms

…and add another f_{target} level …



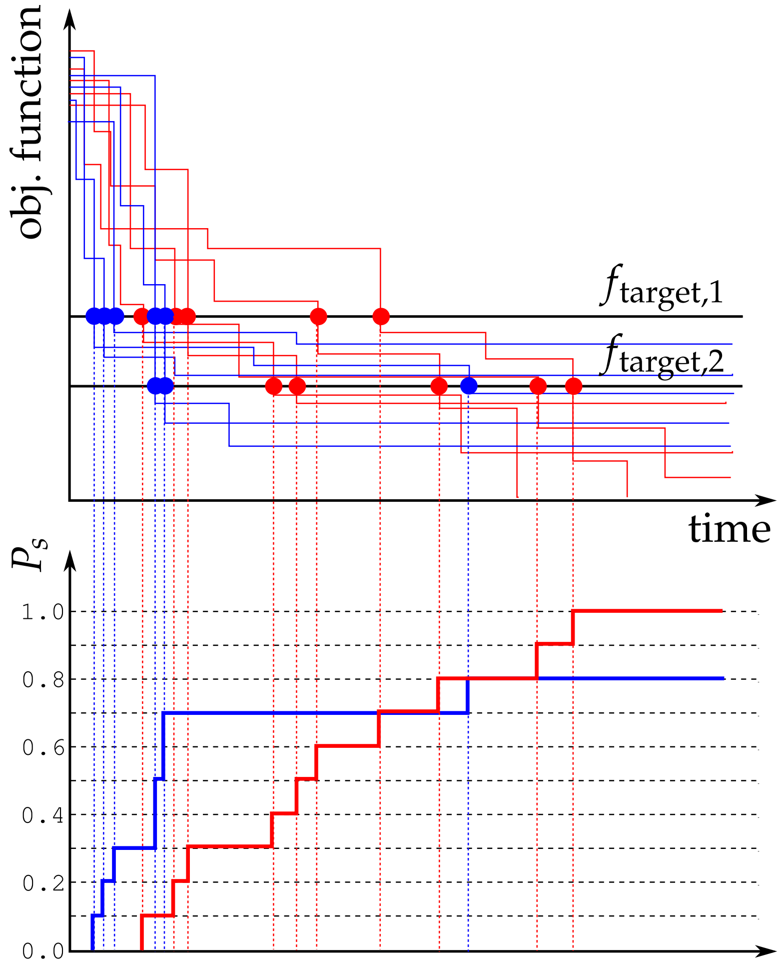
This way we can aggregate RTDs of an algorithm A not only

- over various f_{target} levels, but also

RTD based algorithm comparisons

E.g. type 2 application scenario: set f_{target} and compare RTDs of the algorithms

. . . and add another f_{target} level . . .



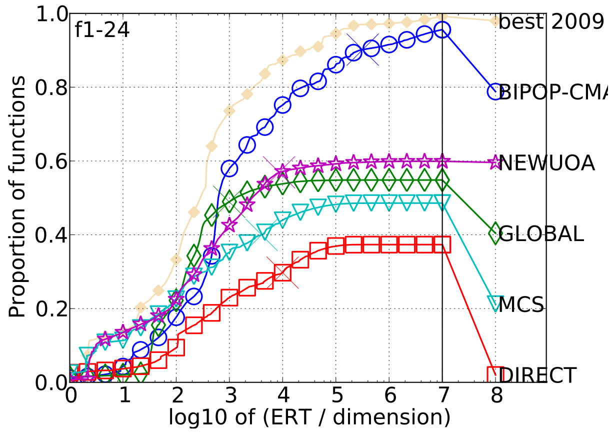
This way we can aggregate RTDs of an algorithm A not only

- over various f_{target} levels, but also
- over different problems $\pi \in \Pi$ (!!!), of course with certain loss of information.

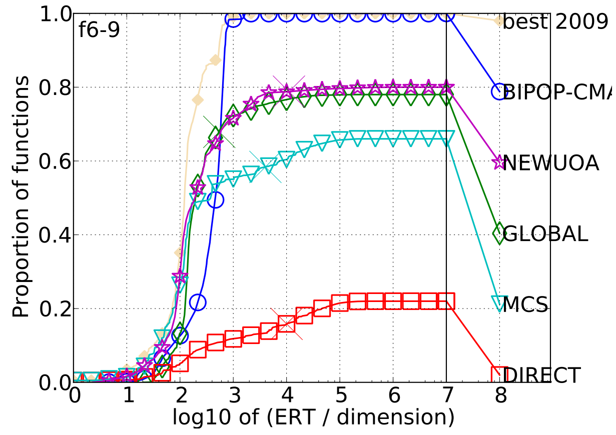
Example of comparison

Workshop on black-box optimization benchmarking (BBOB) at GECCO conference:

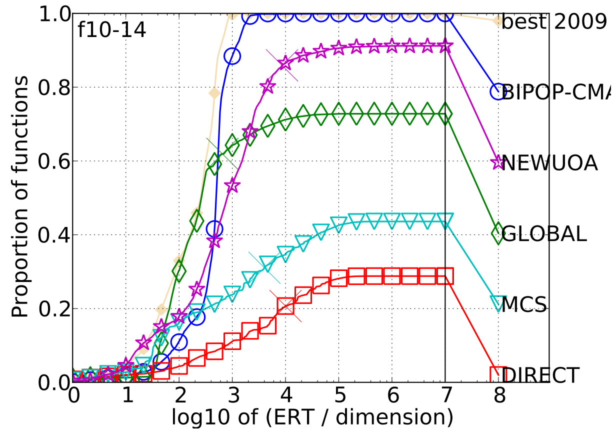
all



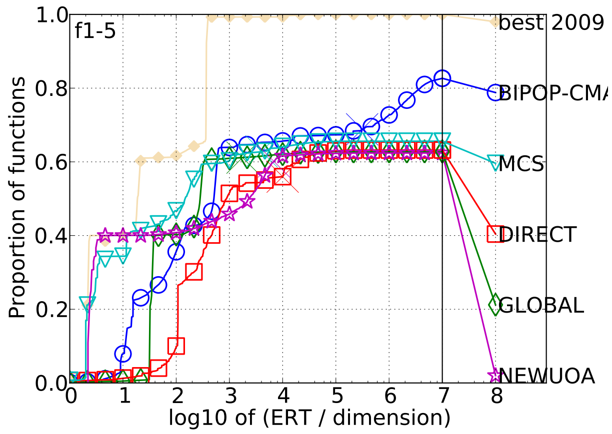
unimodal, low cond.



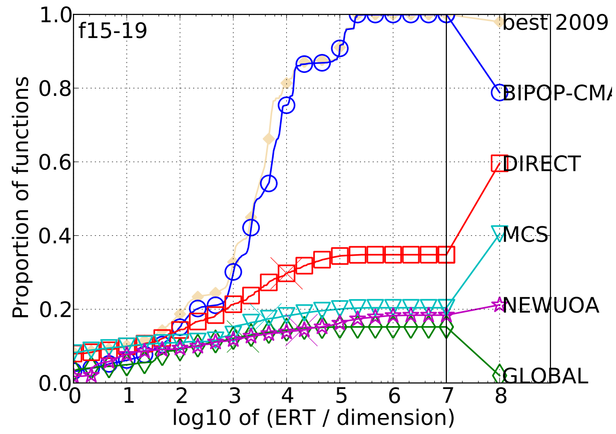
unimodal, high cond.



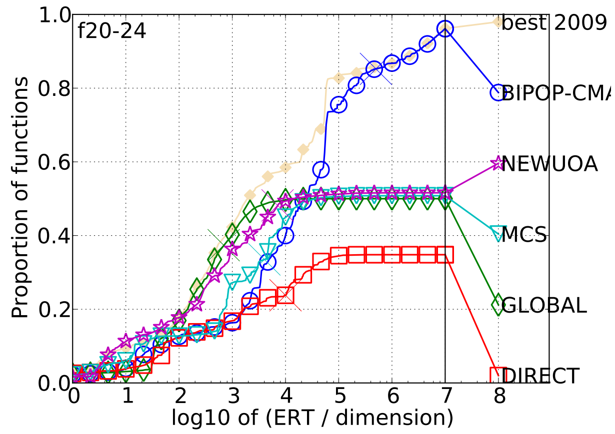
separable



multimodal, structured



multimodal, weak structure



Summary

Summary

- No-free-lunch: all algorithms behave equally on average.
- Comparison of optimization algorithms
  - makes sense only on a well-defined class of problems,
  - is not easy since the chosen measures of algorithm quality are often random variables,
  - is often inconclusive unless the application scenario (utility function) is known.
- The most common scenario is
  - fix available runtime $t_{\max}$,
  - perform several runs and measure the solution quality at the end of each,
  - compare the algorithms based on median (or average) solution quality returned, and
  - asses statistical significance of the difference using Mann-Whitney U test.
- All measures for comparison can be derived from $rtd(t, q)$.