STOCHASTIC LOCAL SEARCH
FOUNDATIONS AND APPLICATIONS

Empirical Analysis of SLS Algorithms

Holger H. Hoos & Thomas Stützle

Outline

1. Las Vegas Algorithms
2. Run-Time Distributions
3. RTD-Based Analysis of LVA Behaviour
4. Characterising and Improving LVA Behaviour

Las Vegas Algorithms

SLS algorithms are typically *incomplete*: there is no guarantee that an (optimal) solution for a given problem instance will eventually be found.

Las Vegas Algorithms

SLS algorithms are typically *incomplete*: there is no guarantee that an (optimal) solution for a given problem instance will eventually be found.

But: For decision problems, any solution returned is guaranteed to be correct.

Las Vegas Algorithms

SLS algorithms are typically *incomplete*: there is no guarantee that an (optimal) solution for a given problem instance will eventually be found.

But: For decision problems, any solution returned is guaranteed to be correct.

Also: The run-time required for finding a solution (in case one is found) is subject to random variation.

Las Vegas Algorithms

SLS algorithms are typically *incomplete*: there is no guarantee that an (optimal) solution for a given problem instance will eventually be found.

But: For decision problems, any solution returned is guaranteed to be correct.

Also: The run-time required for finding a solution (in case one is found) is subject to random variation.

$\rightsquigarrow$ These properties define the class of *(generalised) Las Vegas algorithms*, of which SLS algorithms are a subset.

Definition: (Generalised) Las Vegas Algorithm (LVA)

An algorithm A for a problem class Π is a *(generalised) Las Vegas algorithm (LVA)* iff it has the following properties:

(1) If for a given problem instance $\pi \in \Pi$, algorithm A terminates returning a solution s, s is guaranteed to be a correct solution of π.

Definition: (Generalised) Las Vegas Algorithm (LVA)

An algorithm A for a problem class Π is a *(generalised) Las Vegas algorithm (LVA)* iff it has the following properties:

(1) If for a given problem instance $\pi \in \Pi$, algorithm A terminates returning a solution s, s is guaranteed to be a correct solution of π.

(2) For any given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable $RT_{A,\pi}$.

Definition: (Generalised) Las Vegas Algorithm (LVA)

An algorithm A for a problem class Π is a *(generalised) Las Vegas algorithm (LVA)* iff it has the following properties:

(1) If for a given problem instance $\pi \in \Pi$, algorithm A terminates returning a solution s, s is guaranteed to be a correct solution of π.

(2) For any given instance $\pi \in \Pi$, the run-time of A applied to π is a random variable $RT_{A,\pi}$.

Note: This is a slight generalisation of the definition of a Las Vegas algorithm known from theoretical computing science (our definition includes algorithms that are not guaranteed to return a solution).

Note:

- Any SLS algorithm for a decision problem is also a Las Vegas algorithm. (Condition 1 is trivially satisfied because solutions are checked to be correct before they are returned.)

Note:

- Any SLS algorithm for a decision problem is also a Las Vegas algorithm. (Condition 1 is trivially satisfied because solutions are checked to be correct before they are returned.)
- Las Vegas algorithms can be deterministic, since deterministic run-time is modelled by a *degenerate probability distribution* (aka *Dirac delta distribution*).

Note: For SLS algorithms for optimisation problems, the *solution quality achieved within bounded run-time* as well as the *run-time required for reaching a given solution quality* are random variables.

Note: For SLS algorithms for optimisation problems, the *solution quality achieved within bounded run-time* as well as the *run-time required for reaching a given solution quality* are random variables.

Definition: Optimisation Las Vegas Algorithm (OLVA)

An algorithm A for an optimisation problem Π' is a *(generalised) optimisation Las Vegas algorithm (OLVA)* iff it has the following properties:

(1) A is a (generalised) Las Vegas algorithm.

Note: For SLS algorithms for optimisation problems, the *solution quality achieved within bounded run-time* as well as the *run-time required for reaching a given solution quality* are random variables.

Definition: Optimisation Las Vegas Algorithm (OLVA)

An algorithm A for an optimisation problem Π' is a *(generalised) optimisation Las Vegas algorithm (OLVA)* iff it has the following properties:

(1) A is a (generalised) Las Vegas algorithm.

(2) For any given instance $\pi' \in \Pi'$, the solution quality achieved by A applied to π' after any given run-time t is a random variable $SQ_{A,\pi'}(t)$.

Note:

- Las Vegas algorithms are prominent in many areas of computing science and operations research.

Note:

- Las Vegas algorithms are prominent in many areas of computing science and operations research.
- There are successful types of LVAs other than SLS algorithms, *e.g.*, *randomised systematic search algorithms*.

Note:

- Las Vegas algorithms are prominent in many areas of computing science and operations research.
- There are successful types of LVAs other than SLS algorithms, *e.g.*, *randomised systematic search algorithms*.
- LVAs can be seen as special cases of *Monte Carlo Algorithms*, *i.e.*, randomised algorithms that can sometimes return an incorrect solution to the given problem instance (*false positive result*).

Note:

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically.

Note:

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically.
- 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 (*e.g.*, convergence results for Simulated Annealing);

Note:

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically.
- 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 (*e.g.*, convergence results for Simulated Annealing);
  - apply to *worst-case* or *highly idealised average-case behaviour* only;

Note:

- Practically relevant Las Vegas algorithms are typically difficult to analyse theoretically.
- 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 (*e.g.*, convergence results for Simulated Annealing);
  - apply to *worst-case* or *highly idealised average-case behaviour* only;
  - capture only *asymptotic behaviour* and do not reflect *actual behaviour* with sufficient accuracy.

Therefore:

Analyse the behaviour of Las Vegas algorithms using *empirical methodology*, ideally based on the *scientific method*:

Therefore:

Analyse the behaviour of Las Vegas algorithms using *empirical methodology*, ideally based on the *scientific method*:

- make observations

Therefore:

Analyse the behaviour of Las Vegas algorithms using *empirical methodology*, ideally based on the *scientific method*:

- make observations
- formulate hypothesis/hypotheses (*model*)

Therefore:

Analyse the behaviour of Las Vegas algorithms 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

Therefore:

Analyse the behaviour of Las Vegas algorithms 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

Therefore:

Analyse the behaviour of Las Vegas algorithms 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

Therefore:

Analyse the behaviour of Las Vegas algorithms 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

Asymptotic run-time behaviour of LVAs

- *completeness:*
  for each soluble problem instance π there is a time bound $t_{max}(\pi)$ for the time required to find a solution.

Asymptotic run-time behaviour of LVAs

- *completeness:*
  for each soluble problem instance π there is a time bound $t_{max}(\pi)$ for the time required to find a solution.

- *probabilistic approximate completeness (PAC property):*
  for each soluble problem instance a solution is found with probability $\rightarrow 1$ as run-time $\rightarrow \infty$.

  Note: Do not confuse with *probably approximately correct (PAC) learning*.

Asymptotic run-time behaviour of LVAs

- *completeness:*
  for each soluble problem instance π there is a time bound $t_{max}(\pi)$ for the time required to find a solution.

- *probabilistic approximate completeness (PAC property):*
  for each soluble problem instance a solution is found with probability $\rightarrow 1$ as run-time $\rightarrow \infty$.

  Note: Do not confuse with *probably approximately correct (PAC) learning*.

- *essential incompleteness:*
  for some soluble problem instances, the probability for finding a solution is strictly smaller than 1 for run-time $\rightarrow \infty$.

Examples:

- Many randomised tree search algorithms are complete, *e.g.*, Satz-Rand [Gomes *et al.*, 1998].

Examples:

- Many randomised tree search algorithms are complete, *e.g.*, Satz-Rand [Gomes *et al.*, 1998].

- *Uninformed Random Walk* and *Randomised Iterative Improvement* are probabilistically approximately complete (PAC).

Examples:

- Many randomised tree search algorithms are complete, *e.g.*, Satz-Rand [Gomes *et al.*, 1998].
- *Uninformed Random Walk* and *Randomised Iterative Improvement* are probabilistically approximately complete (PAC).
- *Iterative Improvement* is essentially incomplete.

Note:

- Completeness of SLS algorithms can be achieved by using a restart mechanism that systematically initialises the search at all candidate solutions.

Note:

- Completeness of SLS algorithms can be achieved by using a restart mechanism that systematically initialises the search at all candidate solutions.

  $\leadsto$ Typically very ineffective, due to large size of search space.

Note:

- Completeness of SLS algorithms can be achieved by using a restart mechanism that systematically initialises the search at all candidate solutions.

  $\rightsquigarrow$ Typically very ineffective, due to large size of search space.

- Essential incompleteness of SLS algorithms is typically caused by inability to escape from attractive local minima regions of search space.

Note:

- Completeness of SLS algorithms can be achieved by using a restart mechanism that systematically initialises the search at all candidate solutions.

  $\rightsquigarrow$ Typically very ineffective, due to large size of search space.

- Essential incompleteness of SLS algorithms is typically caused by inability to escape from attractive local minima regions of search space.

  Remedy: Use *diversification mechanisms* such as random restart, random walk, probabilistic tabu tenure, ...

Note:

- Completeness of SLS algorithms can be achieved by using a restart mechanism that systematically initialises the search at all candidate solutions.

  $\rightsquigarrow$ Typically very ineffective, due to large size of search space.

- Essential incompleteness of SLS algorithms is typically caused by inability to escape from attractive local minima regions of search space.

  Remedy: Use *diversification mechanisms* such as random restart, random walk, probabilistic tabu tenure, ...

  In many cases, these can render algorithms provably PAC; but effectiveness in practice can vary widely.

Asymptotic behaviour of OLVAs

- Simple generalisation based on associated decision problems for given solution quality bound $q := r \cdot q^*$, where $q^* =$ optimal solution quality for given problem instance:
  - *completeness* $\rightsquigarrow$ *r-completeness*

Asymptotic behaviour of OLVAs

- Simple generalisation based on associated decision problems for given solution quality bound $q := r \cdot q^*$, where $q^* =$ optimal solution quality for given problem instance:
  - *completeness* $\rightsquigarrow$ *r-completeness*
  - *probabilistic approximate completeness* $\rightsquigarrow$ *probabilistic approximate r-completeness (r-PAC property)*
  - *essential incompleteness* $\rightsquigarrow$ *essential r-incompleteness*

Asymptotic behaviour of OLVAs

- Simple generalisation based on associated decision problems for given solution quality bound $q := r \cdot q^*$, where $q^* =$ optimal solution quality for given problem instance:
  - *completeness* $\rightsquigarrow$ *r-completeness*
  - *probabilistic approximate completeness* $\rightsquigarrow$ *probabilistic approximate r-completeness (r-PAC property)*
  - *essential incompleteness* $\rightsquigarrow$ *essential r-incompleteness*
- Terminology for optimal solution qualities:

  complete = 1-*complete*, *PAC* = 1-*PAC*,
  essentially incomplete = *essentially* 1-*incomplete*.

Application scenarios and evaluation criteria (1)

Evaluation criteria for LVAs depend on the application context:

- **Type 1:** 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).

  $\rightsquigarrow$ evaluation criterion: expected run-time

Application scenarios and evaluation criteria (1)

Evaluation criteria for LVAs depend on the application context:

- **Type 1:** 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).

  $\rightsquigarrow$ evaluation criterion: expected run-time

- **Type 2:** 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).

  $\rightsquigarrow$ evaluation criterion: solution probability at time t_{max}

Application scenarios and evaluation criteria (2)

In many real applications, utility of solutions depends in more complex ways on time required for finding them:

- **Type 3:** Characterised by *utility function* $U : \mathbb{R}^+ \mapsto [0, 1]$, where $U(t)$ = utility of solution found at time t.

Application scenarios and evaluation criteria (2)

In many real applications, utility of solutions depends in more complex ways on time required for finding them:

- **Type 3:** Characterised by *utility function* $U : \mathbb{R}^+ \mapsto [0, 1]$, where $U(t) =$ utility of solution found at time t.

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

Application scenarios and evaluation criteria (2)

In many real applications, utility of solutions depends in more complex ways on time required for finding them:

- **Type 3:** Characterised by *utility function* $U : \mathbb{R}^+ \mapsto [0, 1]$, where $U(t)$ = utility of solution found at time t.

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

  Evaluation criterion for type 3 scenario:
  utility-weighted solution probability $U(t) \cdot P_s(RT \leq t)$
  $\rightsquigarrow$ requires detailed knowledge of $P_s(RT \leq t)$ for arbitrary t.

Application scenarios and evaluation criteria (2)

In many real applications, utility of solutions depends in more complex ways on time required for finding them:

- **Type 3:** Characterised by *utility function* $U : \mathbb{R}^+ \mapsto [0, 1]$, where $U(t) =$ utility of solution found at time t.

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

  Evaluation criterion for type 3 scenario:
  utility-weighted solution probability $U(t) \cdot P_s(RT \leq t)$
  $\rightsquigarrow$ requires detailed knowledge of $P_s(RT \leq t)$ for arbitrary t.

Note: Type 3 is a generalisation of types 1 and 2.

For optimisation Las Vegas algorithms, solution quality also has to be considered.

For optimisation Las Vegas algorithms, solution quality also has to be considered.

Some scenarios:

- Run-time is unconstrained, given solution quality threshold must be reached (generalisation of type 1 scenario)

For optimisation Las Vegas algorithms, solution quality also has to be considered.

Some scenarios:

- Run-time is unconstrained, given solution quality threshold must be reached (generalisation of type 1 scenario)
- Hard time-limit is given, during which best possible solution quality should be found (generalisation of type 2 scenario).

In many cases, tradeoffs between run-time and solution quality are more complex.

- **Generalisation of type 3 scenario:** Utility of solution depends on quality and time needed for finding it; characterised by utility function $U : \mathbb{R}^+ \times \mathbb{R}^+ \mapsto [0, 1]$, where $U(t, q) =$ utility of solution of quality q found at time t.

In many cases, tradeoffs between run-time and solution quality are more complex.

- **Generalisation of type 3 scenario:** Utility of solution depends on quality and time needed for finding it; characterised by utility function $U : \mathbb{R}^+ \times \mathbb{R}^+ \mapsto [0,1]$, where $U(t,q) =$ utility of solution of quality q found at time t.

  Evaluation criterion: utility-weighted solution probability $U(t,q) \cdot P_s(RT \leq t, SQ \leq q)$
  $\rightsquigarrow$ requires detailed knowledge of $P_s(RT \leq t, SQ \leq q)$.

Run-Time Distributions

Las Vegas algorithms are often designed and evaluated without *a priori* knowledge of the application scenario; therefore:

Run-Time Distributions

Las Vegas algorithms are often designed and evaluated without *a priori* knowledge of the application scenario; therefore:

- assume most general scenario: type 3 with unknown utility function;

Run-Time Distributions

Las Vegas algorithms are often designed and evaluated without *a priori* knowledge of the application scenario; therefore:

- assume most general scenario: type 3 with unknown utility function;
- evaluate based on solution probabilities $P_s(RT \leq t)$ or $P_s(RT \leq t, SQ \leq q)$ for arbitrary run-times t and solution qualities q.

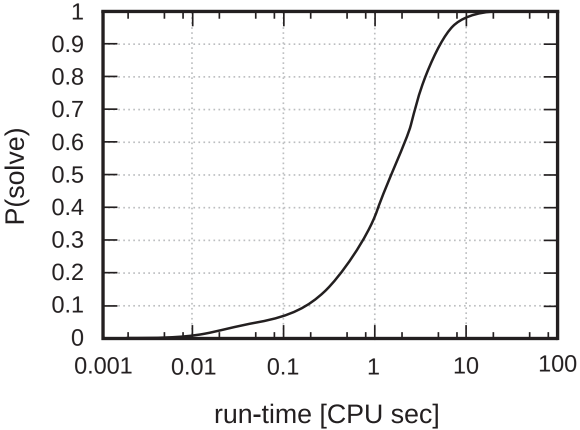
Run-Time Distributions

Las Vegas algorithms are often designed and evaluated without *a priori* knowledge of the application scenario; therefore:

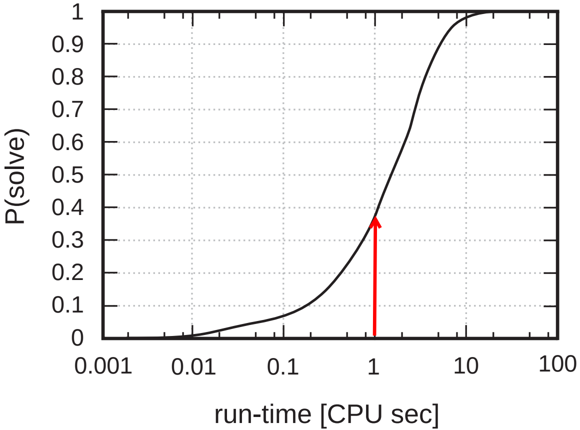
- assume most general scenario: type 3 with unknown utility function;
- evaluate based on solution probabilities $P_s(RT \leq t)$ or $P_s(RT \leq t, SQ \leq q)$ for arbitrary run-times t and solution qualities q.

$\rightsquigarrow$ study distributions of *random variables* characterising run-time and solution quality of algorithm on given problem instance.

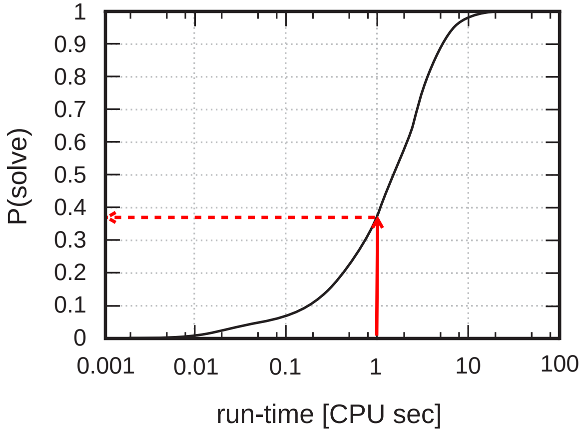
Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial decision problem:



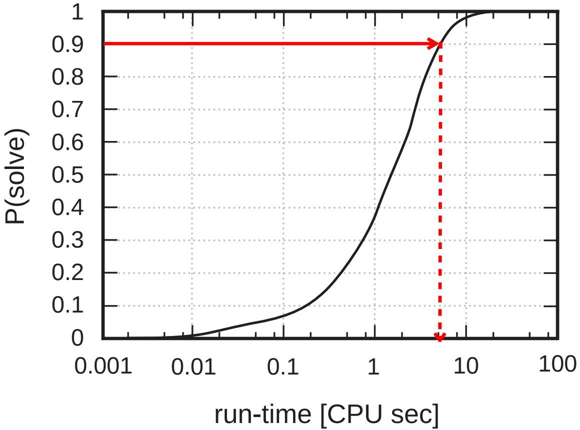
Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial decision problem:



Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial decision problem:



Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial decision problem:



Definition: **Run-Time Distribution (1)**

Given Las Vegas algorithm A for decision problem Π:

- The *success probability* $P_s(RT_{A,\pi} \leq t)$ is the probability that A finds a solution for a soluble instance $\pi \in \Pi$ in time $\leq t$.

Definition: **Run-Time Distribution (1)**

Given Las Vegas algorithm A for decision problem Π:

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

Definition: **Run-Time Distribution (1)**

Given Las Vegas algorithm A for decision problem Π:

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

Definition: **Run-Time Distribution (2)**

Given OLVA A' for optimisation problem Π':

- The *success probability* $P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$ is the probability that A' finds a solution for a soluble instance $\pi' \in \Pi'$ of quality $\leq q$ in time $\leq t$.

Definition: **Run-Time Distribution (2)**

Given OLVA A' for optimisation problem Π':

- The *success probability* $P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$ is the probability that A' finds a solution for a soluble 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 $(RT_{A',\pi'}, SQ_{A',\pi'})$.

Definition: **Run-Time Distribution (2)**

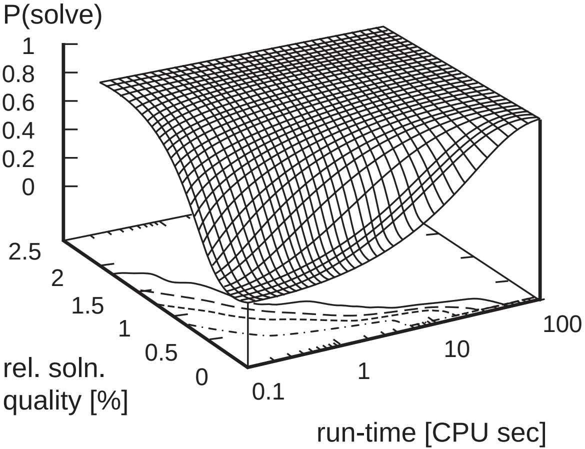
Given OLVA A' for optimisation problem Π':

- The *success probability* $P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$ is the probability that A' finds a solution for a soluble instance $\pi' \in \Pi'$ of quality $\leq q$ in time $\leq t$.

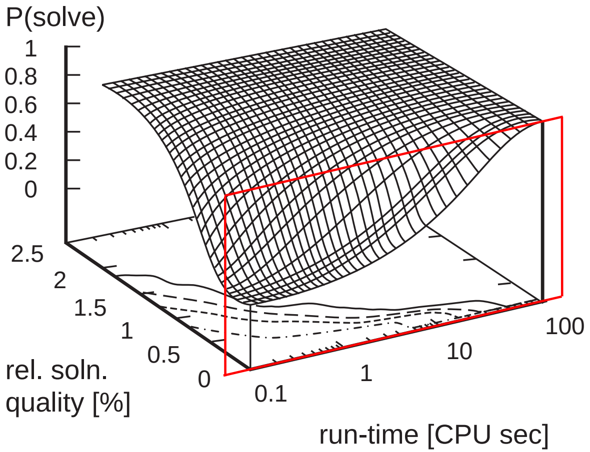
- The *run-time distribution (RTD) of A' on π'* is the probability distribution of the bivariate random variable $(RT_{A',\pi'}, SQ_{A',\pi'})$.

- The *run-time distribution function rtd* $: \mathbb{R}^+ \times \mathbb{R}^+ \mapsto [0,1]$, defined as $rtd(t,q) = P_s(RT_{A,\pi} \leq t, SQ_{A',\pi'} \leq q)$, completely characterises the RTD of A' on π'.

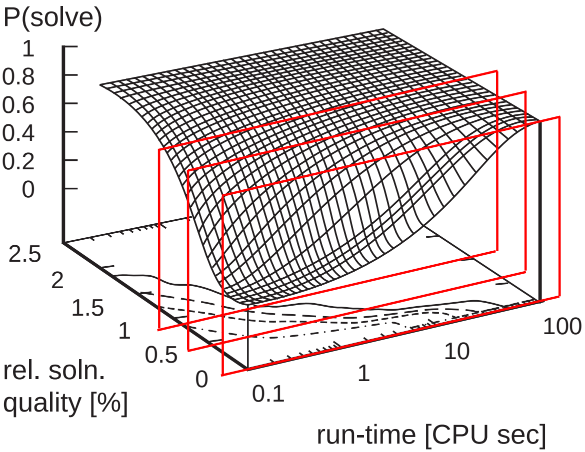
Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



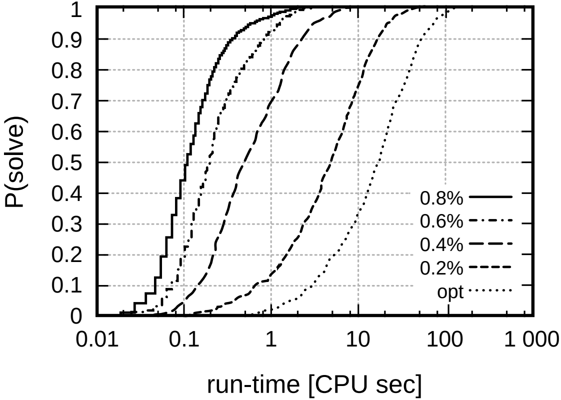
Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Typical run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Qualified RTDs for various solution qualities:



Qualified run-time distributions (QRTDs)

- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$qrtd_{q'}(t) := rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

Qualified run-time distributions (QRTDs)

- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$qrtd_{q'}(t) := rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

- QRTDs correspond to cross-sections of the two-dimensional bivariate RTD graph.

Qualified run-time distributions (QRTDs)

- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t,q)$ defined by:

$$qrtd_{q'}(t) := rtd(t,q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

- QRTDs correspond to cross-sections of the two-dimensional bivariate RTD graph.

- QRTDs characterise the ability of a given SLS algorithm for a combinatorial optimisation problem to solve the associated decision problems.

Qualified run-time distributions (QRTDs)

- A *qualified run-time distribution (QRTD) of an OLVA A' applied to a given problem instance π' for solution quality q'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

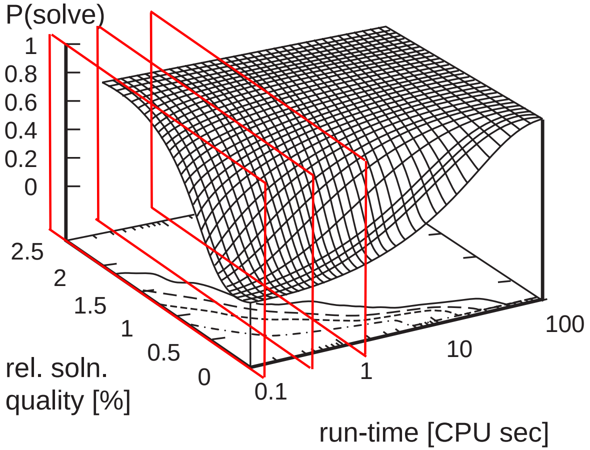
  $$qrtd_{q'}(t) := rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

- QRTDs correspond to cross-sections of the two-dimensional bivariate RTD graph.

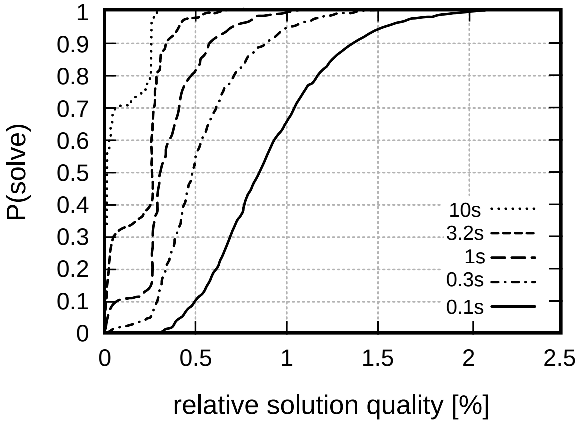
- QRTDs characterise the ability of a given SLS algorithm for a combinatorial optimisation problem to solve the associated decision problems.

Note: Solution qualities q are often expressed as *relative solution qualities* $q/q^* - 1$, where q^* = optimal solution quality for given problem instance.

Typical solution quality distributions for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Solution quality distributions for various run-times:



Solution quality distributions (SQDs)

- A *solution quality distribution (SQD) of an OLVA A' applied to a given problem instance π' for run-time t'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$sqd_{t'}(q) := rtd(t', q) = P_s(RT_{A',\pi'} \leq t', SQ_{A',\pi'} \leq q).$$

Solution quality distributions (SQDs)

- A *solution quality distribution (SQD) of an OLVA A' applied to a given problem instance π' for run-time t'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

$$sqd_{t'}(q) := rtd(t', q) = P_s(RT_{A',\pi'} \leq t', SQ_{A',\pi'} \leq q).$$

- SQDs correspond to cross-sections of the two-dimensional bivariate RTD graph.

Solution quality distributions (SQDs)

- A *solution quality distribution (SQD) of an OLVA A' applied to a given problem instance π' for run-time t'* is a marginal distribution of the bivariate RTD $rtd(t, q)$ defined by:

  $$sqd_{t'}(q) := rtd(t', q) = P_s(RT_{A',\pi'} \leq t', SQ_{A',\pi'} \leq q).$$

- SQDs correspond to cross-sections of the two-dimensional bivariate RTD graph.
- SQDs characterise the solution qualities achieved by a given SLS algorithm for a combinatorial optimisation problem within a given run-time bound (useful for type 2 application scenarios).

Note:

- For sufficiently long run-times, increase in mean solution quality is often accompanied by decrease in solution quality variability.

Note:

- For sufficiently long run-times, increase in mean solution quality is often accompanied by decrease in solution quality variability.
- For PAC algorithms, the SQDs for very large time-limits t' approach degenerate distributions that concentrate all probability on the optimal solution quality.

Note:

- For sufficiently long run-times, increase in mean solution quality is often accompanied by decrease in solution quality variability.
- For PAC algorithms, the SQDs for very large time-limits t' approach degenerate distributions that concentrate all probability on the optimal solution quality.
- For any essentially incomplete algorithm A' (such as Iterative Improvement) applied to a problem instance π', the SQDs for sufficiently large time-limits t' approach a non-degenerate distribution called the *asymptotic SQD of A' on π'*.

Solution quality statistics over time (SQTs)

- The development of solution quality over the run-time of a given OLVA is reflected in time-dependent SQD statistics (*solution quality over time (SQT) curves*).

Solution quality statistics over time (SQTs)

- The development of solution quality over the run-time of a given OLVA is reflected in time-dependent SQD statistics (*solution quality over time (SQT) curves*).

- SQT curves based on SQD quantiles (such as median solution quality) correspond to contour lines of the two-dimensional bivariate RTD graph.

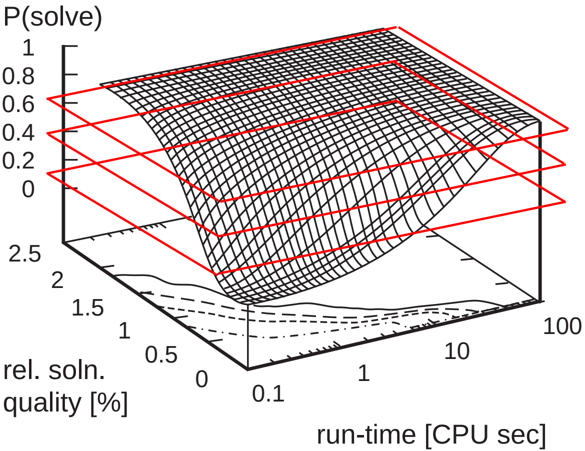
Solution quality statistics over time (SQTs)

- The development of solution quality over the run-time of a given OLVA is reflected in time-dependent SQD statistics (*solution quality over time (SQT) curves*).
- SQT curves based on SQD quantiles (such as median solution quality) correspond to contour lines of the two-dimensional bivariate RTD graph.
- SQT curves are widely used to illustrate the trade-off between run-time and solution quality for a given OLVA.

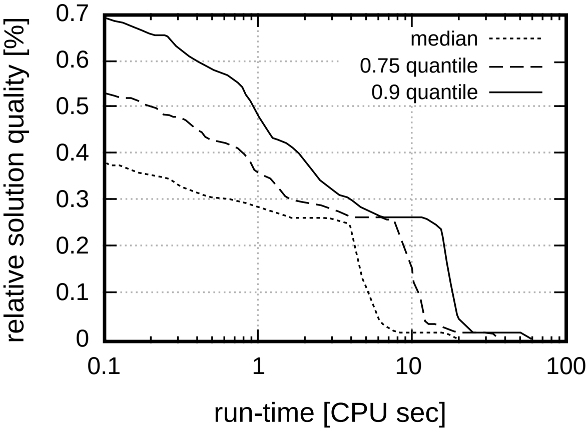
Solution quality statistics over time (SQTs)

- The development of solution quality over the run-time of a given OLVA is reflected in time-dependent SQD statistics (*solution quality over time (SQT) curves*).
- SQT curves based on SQD quantiles (such as median solution quality) correspond to contour lines of the two-dimensional bivariate RTD graph.
- SQT curves are widely used to illustrate the trade-off between run-time and solution quality for a given OLVA.
- **But:** Important aspects of an algorithm's run-time behaviour may be easily missed when basing an analysis solely on a single SQT curve.

Typical SQT curves for SLS optimisation algorithms applied to instance of hard combinatorial optimisation problem:



Typical SQT curves for SLS optimisation algorithms applied to instance of hard combinatorial optimisation problem:



Empirically measuring RTDs

- Except for very simple algorithms, where they can be derived analytically, RTDs are measured empirically.

Empirically measuring RTDs

- Except for very simple algorithms, where they can be derived analytically, RTDs are measured empirically.
- Empirical RTDs are approximations of an algorithm's true RTD.

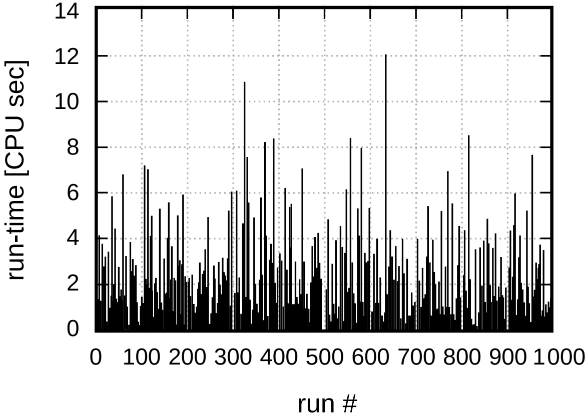
Empirically measuring RTDs

- Except for very simple algorithms, where they can be derived analytically, RTDs are measured empirically.
- Empirical RTDs are approximations of an algorithm's true RTD.
- Empirical RTDs are determined from a number of independent, successful runs of the algorithm on a given problem instance (*samples of theoretical RTD*).

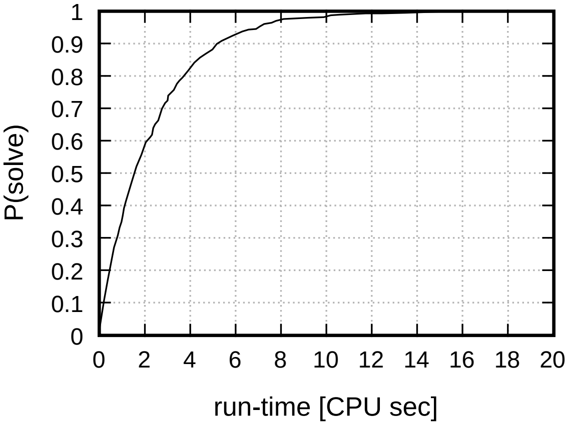
Empirically measuring RTDs

- Except for very simple algorithms, where they can be derived analytically, RTDs are measured empirically.
- Empirical RTDs are approximations of an algorithm's true RTD.
- Empirical RTDs are determined from a number of independent, successful runs of the algorithm on a given problem instance (*samples of theoretical RTD*).
- Higher numbers of runs (larger *sample sizes*) give more accurate approximations of a true RTD.

Typical sample of run-times for an SLS algorithm applied to an instance of a hard decision problem:



Corresponding empirical RTD:



Protocol for obtaining the empirical RTD for an LVA A applied to a given instance π of a decision problem:

- Perform k independent runs of A on π with cutoff time t'. (For most purposes, k should be at least 50–100, and t' should be high enough to obtain at least a large fraction of successful runs.)

Protocol for obtaining the empirical RTD for an LVA A applied to a given instance π of a decision problem:

- Perform k independent runs of A on π with cutoff time t'. (For most purposes, k should be at least 50–100, and t' should be high enough to obtain at least a large fraction of successful runs.)

- Record number k' of successful runs, and for each run, record its run-time in a list L.

Protocol for obtaining the empirical RTD for an LVA A applied to a given instance π of a decision problem:

- Perform k independent runs of A on π with cutoff time t'. (For most purposes, k should be at least 50–100, and t' should be high enough to obtain at least a large fraction of successful runs.)

- Record number k' of successful runs, and for each run, record its run-time in a list L.

- Sort L according to increasing run-time; let $rt(j)$ denote the run-time from entry j of the sorted list $(j = 1, \ldots, k')$.

Protocol for obtaining the empirical RTD for an LVA A applied to a given instance π of a decision problem:

- Perform k independent runs of A on π with cutoff time t'. (For most purposes, k should be at least 50–100, and t' should be high enough to obtain at least a large fraction of successful runs.)

- Record number k' of successful runs, and for each run, record its run-time in a list L.

- Sort L according to increasing run-time; let $rt(j)$ denote the run-time from entry j of the sorted list $(j = 1, \ldots, k')$.

- Plot the graph $(rt(j), j/k)$, *i.e.*, the cumulative empirical RTD of A on π.

Note:

- The fraction of successful runs, $sr := k'/k$, is called the *success ratio*; for large run-times t', it approximates the *asymptotic success probability* $p_s^* := lim_{t \to \infty} P_s(RT_{a,\pi} \leq t)$.

Note:

- The fraction of successful runs, $sr := k'/k$, is called the *success ratio*; for large run-times t', it approximates the *asymptotic success probability* $p_s^* := lim_{t \to \infty} P_s(RT_{a,\pi} \leq t)$.
- In cases where the success ratio sr for a given cutoff time t' is smaller than 1, quantiles up to sr can still be estimated from the respective truncated RTD.

Note:

- The fraction of successful runs, $sr := k'/k$, is called the *success ratio*; for large run-times t', it approximates the *asymptotic success probability* $p_s^* := lim_{t \to \infty} P_s(RT_{a,\pi} \leq t)$.

- In cases where the success ratio sr for a given cutoff time t' is smaller than 1, quantiles up to sr can still be estimated from the respective truncated RTD.

  The mean run-time for a variant of the algorithm that restarts after time t' can be estimated as:

  $$\widehat{E}(RT_s) + (1/sr - 1) \cdot \widehat{E}(RT_f)$$

  where $\widehat{E}(RT_s)$ and $\widehat{E}(RT_f)$ are the average times of successful and failed runs, respectively.

  Note: $1/sr - 1$ is the expected number of failed runs required before a successful run is observed.

Protocol for obtaining the empirical RTD for an OLVA A' applied to a given instance π' of an optimisation problem:

- Perform k independent runs of A' on π' with cutoff time t'.

Protocol for obtaining the empirical RTD for an OLVA A' applied to a given instance π' of an optimisation problem:

- Perform k independent runs of A' on π' with cutoff time t'.
- During each run, whenever the incumbent solution is improved, record the quality of the improved incumbent solution and the time at which the improvement was achieved in a *solution quality trace*.

Protocol for obtaining the empirical RTD for an OLVA A' applied to a given instance π' of an optimisation problem:

- Perform k independent runs of A' on π' with cutoff time t'.
- During each run, whenever the incumbent solution is improved, record the quality of the improved incumbent solution and the time at which the improvement was achieved in a *solution quality trace*.
- Let $sq(t', j)$ denote the best solution quality encountered in run j up to time t'. The cumulative empirical RTD of A' on π' is defined by $\widehat{P}_s(RT \leq t', SQ \leq q') := \#\{j \mid sq(t', j) \leq q'\}/k$.

Note: Qualified RTDs, SQDs and SQT curves can be easily derived from the same solution quality traces.

Measuring run-times (1):

- CPU time measurements are based on a specific *implementation* and *run-time environment* (machine, operating system) of the given algorithm.

Measuring run-times (1):

- CPU time measurements are based on a specific *implementation* and *run-time environment* (machine, operating system) of the given algorithm.

- To ensure reproducibility and comparability of empirical results, CPU times should be measured in a way that is as independent as possible from machine load.

Measuring run-times (1):

- CPU time measurements are based on a specific *implementation* and *run-time environment* (machine, operating system) of the given algorithm.

- To ensure reproducibility and comparability of empirical results, CPU times should be measured in a way that is as independent as possible from machine load.

  When reporting CPU times, the run-time environment should be specified (at least CPU type, model, speed and cache size; amount of RAM; OS type and version); ideally, the implementation of the algorithm should be made available.

Measuring run-times (2):

To achieve better abstraction from the implementation and run-time environment, it is often preferable to measure run-time using

- *operation counts* that reflect the number of operations that contribute significantly towards an algorithms performance,

Measuring run-times (2):

To achieve better abstraction from the implementation and run-time environment, it is often preferable to measure run-time using

- *operation counts* that reflect the number of operations that contribute significantly towards an algorithms performance, and
- *cost models* that specify the CPU time for each such operation for a given implementation and run-time environment.

Example:

For a given SLS algorithm for SAT applied to a specific SAT instance we observe

- a median run-time of 38 911 search steps (*operation count*);

Example:

For a given SLS algorithm for SAT applied to a specific SAT instance we observe

- a median run-time of 38 911 search steps (*operation count*);
- the CPU time required for each search step is 0.027ms, while initialisation takes 0.8ms (*cost model*)

Example:

For a given SLS algorithm for SAT applied to a specific SAT instance we observe

- a median run-time of 38 911 search steps (*operation count*);
- the CPU time required for each search step is 0.027ms, while initialisation takes 0.8ms (*cost model*)

  when running the algorithm on an Intel Xeon 2.4GHz CPU with 512KB cache and 1GB RAM running Red Hat Linux, Version 2.4smp (*run-time environment*).

Run-length distributions:

- RTDs based on run-times measured in terms of elementary operations of the given algorithm are also called *run-length distributions (RLDs)*.

Run-length distributions:

- RTDs based on run-times measured in terms of elementary operations of the given algorithm are also called *run-length distributions (RLDs)*.

- **Caution:** RLDs should be based on elementary operations that either require constant CPU time (for the given problem instance), or on aggregate counts in which operations that require different amounts of CPU time (*e.g.*, two types of search steps) are weighted appropriately.

Run-length distributions:

- RTDs based on run-times measured in terms of elementary operations of the given algorithm are also called *run-length distributions (RLDs)*.

- **Caution:** RLDs should be based on elementary operations that either require constant CPU time (for the given problem instance), or on aggregate counts in which operations that require different amounts of CPU time (*e.g.*, two types of search steps) are weighted appropriately.

- Elementary operations commonly used as the basis for RLD and other run-time measurements of SLS algorithms include search steps, objective function evaluations and updates of data structures used for implementing the step function.

RTD-based Analysis of LVA Behaviour

Run-time distributions (and related concepts) provide an excellent basis for

- analysis and characterisation of LVA behaviour;

RTD-based Analysis of LVA Behaviour

Run-time distributions (and related concepts) provide an excellent basis for

- analysis and characterisation of LVA behaviour;
- comparative performance analyses of two or more LVAs;

RTD-based Analysis of LVA Behaviour

Run-time distributions (and related concepts) provide an excellent basis for

- analysis and characterisation of LVA behaviour;
- comparative performance analyses of two or more LVAs;
- investigations of the effects of parameters, problem instance features, *etc.* on the behaviour of an LVA.

RTD-based Analysis of LVA Behaviour

Run-time distributions (and related concepts) provide an excellent basis for

- analysis and characterisation of LVA behaviour;
- comparative performance analyses of two or more LVAs;
- investigations of the effects of parameters, problem instance features, *etc.* on the behaviour of an LVA.

RTD-based empirical analysis in combination with proper statistical techniques (hypothesis tests) is a state-of-the-art approach in empirical algorithmics.

RTD plots are useful for the *qualitative analysis* of LVA behaviour:

- *Semi-log plots* give a better view of the distribution over its entire range.

RTD plots are useful for the *qualitative analysis* of LVA behaviour:

- *Semi-log plots* give a better view of the distribution over its entire range.
- Uniform performance differences characterised by a constant factor correspond to shifts along horizontal axis.

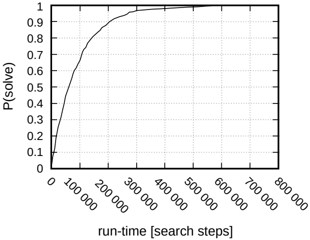
RTD plots are useful for the *qualitative analysis* of LVA behaviour:

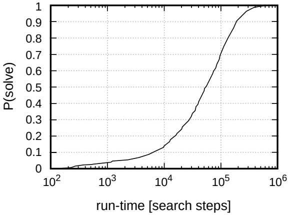
- *Semi-log plots* give a better view of the distribution over its entire range.

- Uniform performance differences characterised by a constant factor correspond to shifts along horizontal axis.

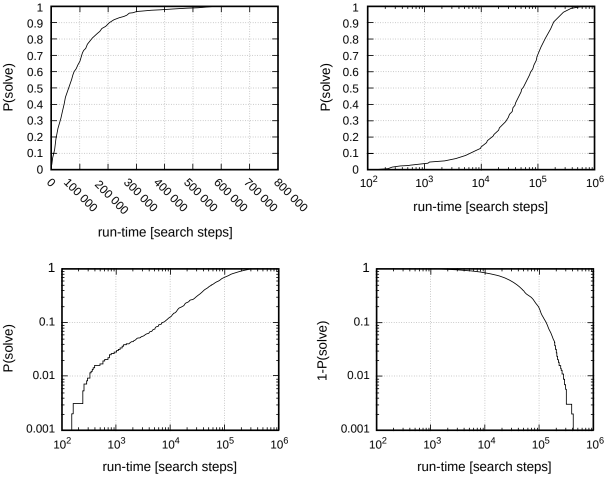
- *Log-log plots* of an RTD or its associated *failure rate decay function*, $1 - rtd(t)$, are often useful for examining behaviour for very short or very long runs.

Various graphical representations of a typical RTD:





Various graphical representations of a typical RTD:



Quantitative RTD analysis is typically based on *basic descriptive statistics*, such as:

- mean;

Quantitative RTD analysis is typically based on *basic descriptive statistics*, such as:

- mean;
- median ($q_{0.5}$) and other quantiles (*e.g.*, $q_{0.25}, q_{0.75}, q_{0.9}$);

Quantitative RTD analysis is typically based on *basic descriptive statistics*, such as:

- mean;
- median ($q_{0.5}$) and other quantiles (*e.g.*, $q_{0.25}, q_{0.75}, q_{0.9}$);
- standard deviation or (better) *variation coefficient* $vc := stddev / mean$;

Quantitative RTD analysis is typically based on *basic descriptive statistics*, such as:

- mean;
- median ($q_{0.5}$) and other quantiles (*e.g.*, $q_{0.25}, q_{0.75}, q_{0.9}$);
- standard deviation or (better) *variation coefficient* $vc := stddev/mean$;
- *quantile ratios*, such as $q_{0.75}/q_{0.5}$ or $q_{0.9}/q_{0.1}$.

Quantitative RTD analysis is typically based on *basic descriptive statistics*, such as:

- mean;
- median ($q_{0.5}$) and other quantiles (*e.g.*, $q_{0.25}, q_{0.75}, q_{0.9}$);
- standard deviation or (better) *variation coefficient* $vc := stddev/mean$;
- *quantile ratios*, such as $q_{0.75}/q_{0.5}$ or $q_{0.9}/q_{0.1}$.

Note: SLS algorithms typically show very high variability in run-time; therefore, reporting a measure of variability along with the mean or median run-time is important.

Example:

The empirical RLD of a given SLS algorithm for SAT on a specific SAT instance is characterised by the following basic descriptive statistics:

mean	57 606.23	median	38 911
min	107	$q_{0.25}$; $q_{0.1}$	16 762; 5 332
max	443 496	$q_{0.75}$; $q_{0.9}$	80 709; 137 863
stddev	58 953.60	$q_{0.75}/q_{0.25}$	4.81
vc	1.02	$q_{0.9}/q_{0.1}$	25.86

Note:

- Quantiles (such as the median) are more stable w.r.t. extreme values than the mean.

Note:

- Quantiles (such as the median) are more stable w.r.t. extreme values than the mean.
- Unlike the standard deviation (or variance), the variation coefficient and quantile ratios are invariant under multiplication by constants.

Note:

- Descriptive statistics can be easily calculated from empirical RTDs.

Note:

- Descriptive statistics can be easily calculated from empirical RTDs.
- Obtaining sufficiently stable descriptive statistics requires the same number of runs of the given algorithm as measuring reasonably accurate empirical RTDs.

Note:

- Descriptive statistics can be easily calculated from empirical RTDs.
- Obtaining sufficiently stable descriptive statistics requires the same number of runs of the given algorithm as measuring reasonably accurate empirical RTDs.
- QRTDs and SQDs can be handled analogously to RTDs; along with SQT curves they can be easily determined from the same solution quality traces that provide the basis for empirical bivariate RTDs of a given optimisation LVA.

Basic quantitative analysis for ensembles of instances (1)

- In principle, the same approach as for individual instances is applicable: Measure empirical RTD for each instance, analyse using RTD plots or descriptive statistics.

Basic quantitative analysis for ensembles of instances (1)

- In principle, the same approach as for individual instances is applicable: Measure empirical RTD for each instance, analyse using RTD plots or descriptive statistics.

- In many cases, the RTDs for set of instances have similar shapes or share important features (*e.g.*, being uni- or bi-modal, or having a prominent right tail).

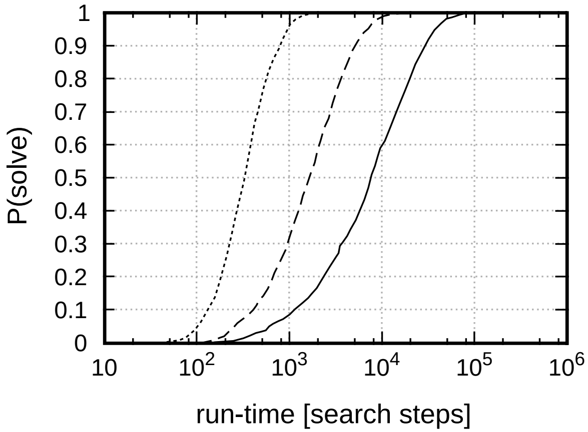
Basic quantitative analysis for ensembles of instances (1)

- In principle, the same approach as for individual instances is applicable: Measure empirical RTD for each instance, analyse using RTD plots or descriptive statistics.

- In many cases, the RTDs for set of instances have similar shapes or share important features (*e.g.*, being uni- or bi-modal, or having a prominent right tail).

  $\rightsquigarrow$ Select typical instance for presentation or further analysis, briefly summarise data for remaining instances.

RTDs for WalkSAT/SKC, a prominent SLS algorithm for SAT, on three hard 3-SAT instances:



Basic quantitative analysis for ensembles of instances (2)

- For bigger sets of instances (*e.g.*, samples from random instance distributions), it is important to characterise the performance of the given algorithm *on individual instances* as well as *across the entire ensemble*.

Basic quantitative analysis for ensembles of instances (2)

- For bigger sets of instances (*e.g.*, samples from random instance distributions), it is important to characterise the performance of the given algorithm *on individual instances* as well as *across the entire ensemble*.

  $\rightsquigarrow$ Report and analyse run-time distributions on representative instance(s) as well as *search cost distribution (SCD)*, *i.e.*, distribution of basic RTD statistics (*e.g.*, median or mean) across given instance ensemble.

Basic quantitative analysis for ensembles of instances (2)

- For bigger sets of instances (*e.g.*, samples from random instance distributions), it is important to characterise the performance of the given algorithm *on individual instances* as well as *across the entire ensemble*.

  $\rightsquigarrow$ Report and analyse run-time distributions on representative instance(s) as well as *search cost distribution (SCD)*, *i.e.*, distribution of basic RTD statistics (*e.g.*, median or mean) across given instance ensemble.

- For sets of instances that have been generated by systematically varying a parameter (*e.g.*, problem size), study RTD characteristics in dependence of the parameter value.

Basic quantitative analysis for ensembles of instances (2)

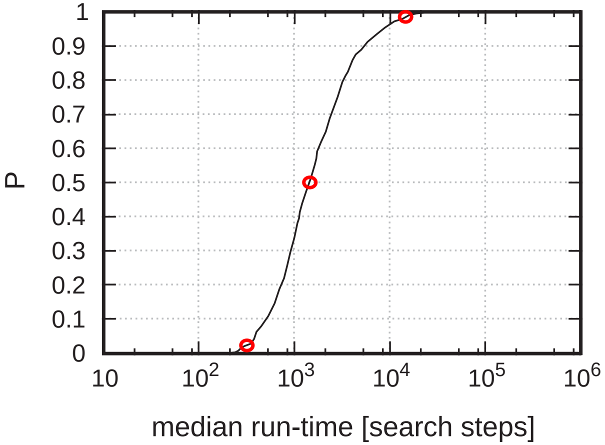
- For bigger sets of instances (*e.g.*, samples from random instance distributions), it is important to characterise the performance of the given algorithm *on individual instances* as well as *across the entire ensemble*.

  $\rightsquigarrow$ Report and analyse run-time distributions on representative instance(s) as well as *search cost distribution (SCD)*, *i.e.*, distribution of basic RTD statistics (*e.g.*, median or mean) across given instance ensemble.

- For sets of instances that have been generated by systematically varying a parameter (*e.g.*, problem size), study RTD characteristics in dependence of the parameter value.

  Useful fact: *Exponential* and *polynomial functions* appear as straight lines in *semi-log plots* and *log-log plots*, respectively.

Distribution of median search cost for WalkSAT/SKC over set of 1000 randomly generated, hard 3-SAT instances:



Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)
- instance size (provide range, scaling studies)

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)
- instance size (provide range, scaling studies)
- instance type (provide variety):

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)
- instance size (provide range, scaling studies)
- instance type (provide variety):
  - individual application instances
  - hand-crafted instances (realistic, artificial)

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on hard instances)
- instance size (provide range, scaling studies)
- instance type (provide variety):
  - individual application instances
  - hand-crafted instances (realistic, artificial)
  - ensembles of instances from random distributions ($\rightsquigarrow$ random instance generators)
  - encodings of various other types of problems (*e.g.*, SAT-encodings of graph colouring problems)

To ensure comparability and reproducibility of results:

- use established benchmark sets from public benchmark libraries (such as TSPLIB, SATLIB, *etc.*) and/or related literature;

To ensure comparability and reproducibility of results:

- use established benchmark sets from public benchmark libraries (such as TSPLIB, SATLIB, *etc.*) and/or related literature;
- make newly created test-sets available to other researchers.

To ensure comparability and reproducibility of results:

- use established benchmark sets from public benchmark libraries (such as TSPLIB, SATLIB, *etc.*) and/or related literature;
- make newly created test-sets available to other researchers.

Note:

Careful selection and good understanding of benchmark sets are often crucial for the relevance of an empirical study!

Comparing algorithms based on RTDs (1)

- Many empirical studies aim to establish the superiority of one Las Vegas algorithm over another.

Comparing algorithms based on RTDs (1)

- Many empirical studies aim to establish the superiority of one Las Vegas algorithm over another.

- For an instance of a decision problem, LVA A is superior to LVA B if for any run-time, A consistently gives a higher solution probability than B (*probabilistic domination*).

Comparing algorithms based on RTDs (1)

- Many empirical studies aim to establish the superiority of one Las Vegas algorithm over another.
- For an instance of a decision problem, LVA A is superior to LVA B if for any run-time, A consistently gives a higher solution probability than B (*probabilistic domination*).
- For an instance of an optimisation problem, OLVA A' probabilistically dominates OLVA B' on a given problem instance iff for all solution quality bounds, A' probabilistically dominates B' on the respective associated decision problem.

Comparing algorithms based on RTDs (2)

- A probabilistic domination relation holds between two Las Vegas algorithms on a given problem instance iff their respective (qualified) RTDs do not cross each other.

Comparing algorithms based on RTDs (2)

- A probabilistic domination relation holds between two Las Vegas algorithms on a given problem instance iff their respective (qualified) RTDs do not cross each other.

- Even for single problem instances, a probabilistic domination relation does not always hold (*i.e.*, there is a cross-over between the respective RTDs).

Comparing algorithms based on RTDs (2)

- A probabilistic domination relation holds between two Las Vegas algorithms on a given problem instance iff their respective (qualified) RTDs do not cross each other.

- Even for single problem instances, a probabilistic domination relation does not always hold (*i.e.*, there is a cross-over between the respective RTDs).

  In this situation, which of two given algorithms is superior depends on the time both algorithms are allowed to run.

Comparing algorithms based on RTDs (2)

- A probabilistic domination relation holds between two Las Vegas algorithms on a given problem instance iff their respective (qualified) RTDs do not cross each other.

- Even for single problem instances, a probabilistic domination relation does not always hold (*i.e.*, there is a cross-over between the respective RTDs).

  In this situation, which of two given algorithms is superior depends on the time both algorithms are allowed to run.

- To assess the *statistical significance* of observed performance differences, an appropriate *statistical hypothesis test* must be applied.

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.

- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.

- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.

  Example: For the Mann-Whitney U-test, the null hypothesis is 'the two distributions underlying two given samples have the same median'.

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.

- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.

  Example: For the Mann-Whitney U-test, the null hypothesis is 'the two distributions underlying two given samples have the same median'.

- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.

Background: Statistical hypothesis tests (1)

- *Statistical hypothesis tests* are used to assess the validity of statements about properties of or relations between sets of statistical data.

- The statement to be tested (or its negation) is called the *null hypothesis* (H_0) of the test.

  Example: For the Mann-Whitney U-test, the null hypothesis is 'the two distributions underlying two given samples have the same median'.

- The *significance level* (α) determines the maximum allowable probability of incorrectly rejecting the null hypothesis.

  Typical values of α are 0.05 or 0.01.

Background: Statistical hypothesis tests (2)

- The *power* of the test provides a lower bound for the probability of correctly accepting the null hypothesis. The desired power of a test determines the required *sample size*.

Background: Statistical hypothesis tests (2)

- The *power* of the test provides a lower bound for the probability of correctly accepting the null hypothesis. The desired power of a test determines the required *sample size*.

  Typical power values are at least 0.8; in many cases, sample size calculations for given power values are difficult.

Background: Statistical hypothesis tests (2)

- The *power* of the test provides a lower bound for the probability of correctly accepting the null hypothesis. The desired power of a test determines the required *sample size*.

  Typical power values are at least 0.8; in many cases, sample size calculations for given power values are difficult.

- The application of a test to a given data set results in a *p-value*, which represents the probability that the null hypothesis is incorrectly rejected.

Background: Statistical hypothesis tests (2)

- The *power* of the test provides a lower bound for the probability of correctly accepting the null hypothesis. The desired power of a test determines the required *sample size*.

  Typical power values are at least 0.8; in many cases, sample size calculations for given power values are difficult.

- The application of a test to a given data set results in a *p-value*, which represents the probability that the null hypothesis is incorrectly rejected.

  The null hypothesis is rejected iff this p-value is smaller than the previously chosen significance level.

Background: Statistical hypothesis tests (2)

- The *power* of the test provides a lower bound for the probability of correctly accepting the null hypothesis. The desired power of a test determines the required *sample size*.

  Typical power values are at least 0.8; in many cases, sample size calculations for given power values are difficult.

- The application of a test to a given data set results in a *p-value*, which represents the probability that the null hypothesis is incorrectly rejected.

  The null hypothesis is rejected iff this p-value is smaller than the previously chosen significance level.

- Most common statistical hypothesis tests and other statistical analyses can be performed rather conveniently in the free *R software environment* (see http://www.r-project.org/).

Comparing algorithms based on RTDs (3)

- The *Mann Whitney U-test* (aka *Wilcoxon rank sum test*) is used to test whether the medians of two samples (*e.g.*, empirical RTDs) are significantly different.

Comparing algorithms based on RTDs (3)

- The *Mann Whitney U-test* (aka *Wilcoxon rank sum test*) is used to test whether the medians of two samples (*e.g.*, empirical RTDs) are significantly different.

- Unlike the widely used t-test, the U-test is *distribution-free* (or *non-parametric*), *i.e.*, it does not depend on the assumption that the underlying probability distributions are Gaussian. (This assumption is typically violated for the RTDs of SLS algorithms.)

Comparing algorithms based on RTDs (3)

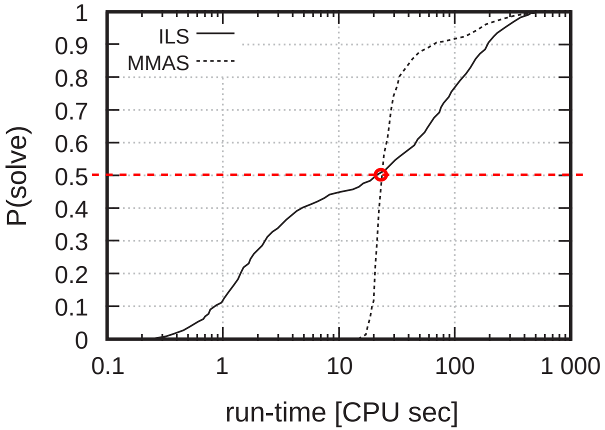
- The *Mann Whitney U-test* (aka *Wilcoxon rank sum test*) is used to test whether the medians of two samples (*e.g.*, empirical RTDs) are significantly different.
- Unlike the widely used t-test, the U-test is *distribution-free* (or *non-parametric*), *i.e.*, it does not depend on the assumption that the underlying probability distributions are Gaussian. (This assumption is typically violated for the RTDs of SLS algorithms.)
- The more specific hypothesis whether the theoretical RTDs (or SQDs) of two algorithms are identical can be tested using the *Kolmogorov-Smirnov test*.

Performance differences detectable by the Mann-Whitney U-test for various sample sizes (sign. level 0.05, power 0.95):

sample size	m_1/m_2
3 010	1.1
1 000	1.18
122	1.5
100	1.6
32	2
10	3

m_1/m_2 is the ratio between the medians of the two empirical distributions.

Example of crossing RTDs for two SLS algorithms for the TSP applied to a standard benchmark instance (1000 runs/RTD):



Comparative analysis for instance ensembles (1)

Goal: Compare performance of Las Vegas algorithms A and B on a given ensemble of instances.

- Use instance-based analysis to partition given ensemble into three subsets:
  - instances on which A probabilistically dominates B;

Comparative analysis for instance ensembles (1)

Goal: Compare performance of Las Vegas algorithms A and B on a given ensemble of instances.

- Use instance-based analysis to partition given ensemble into three subsets:
  - instances on which A probabilistically dominates B;
  - instances on which B probabilistically dominates A;

Comparative analysis for instance ensembles (1)

Goal: Compare performance of Las Vegas algorithms A and B on a given ensemble of instances.

- Use instance-based analysis to partition given ensemble into three subsets:
  - instances on which A probabilistically dominates B;
  - instances on which B probabilistically dominates A;
  - instances on which there is no probabilistic domination between A and B (crossing RTDs).

Comparative analysis for instance ensembles (1)

Goal: Compare performance of Las Vegas algorithms A and B on a given ensemble of instances.

- Use instance-based analysis to partition given ensemble into three subsets:
  - instances on which A probabilistically dominates B;
  - instances on which B probabilistically dominates A;
  - instances on which there is no probabilistic domination between A and B (crossing RTDs).

The size of these subsets gives a rather detailed picture of the algorithms' relative performance on the given ensemble.

Comparative analysis for instance ensembles (2)

- Use statistical tests to assess significance of performance differences across given instance ensemble.

Comparative analysis for instance ensembles (2)

- Use statistical tests to assess significance of performance differences across given instance ensemble.

- The *binomial sign test* measures whether the median of the paired performance differences (*e.g.*, in median run-time) of A and B per instance is significantly different from zero, which means that there is no significant *systematic* performance difference between A and B across the ensemble.

Comparative analysis for instance ensembles (2)

- Use statistical tests to assess significance of performance differences across given instance ensemble.

- The *binomial sign test* measures whether the median of the paired performance differences (*e.g.*, in median run-time) of A and B per instance is significantly different from zero, which means that there is no significant *systematic* performance difference between A and B across the ensemble.

- Note: This test *does not* capture qualitative performance differences such as different shapes of the underlying RTDs and can easily miss interesting variation in relative performance across the ensemble.

Comparative analysis for instance ensembles (3)

- Particularly for large instance ensembles, it is often useful to study the *correlation* between the performance of A and B across the ensemble.

Comparative analysis for instance ensembles (3)

- Particularly for large instance ensembles, it is often useful to study the *correlation* between the performance of A and B across the ensemble.

  Typical performance measures used in this context are RTD or SQD statistics, such as empirical median or mean.

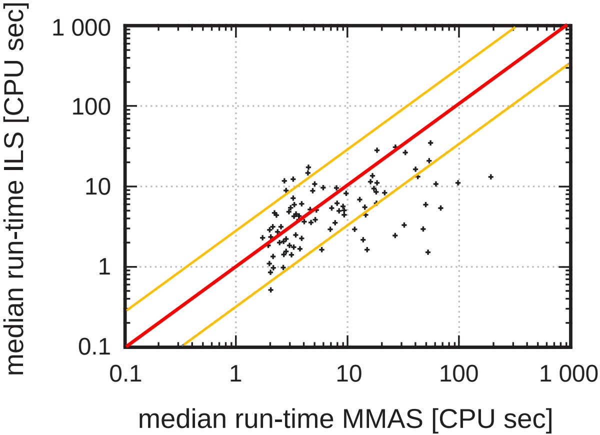
Comparative analysis for instance ensembles (3)

- Particularly for large instance ensembles, it is often useful to study the *correlation* between the performance of A and B across the ensemble.

  Typical performance measures used in this context are RTD or SQD statistics, such as empirical median or mean.

- For qualitative correlation analyses, *scatter plots* in which each instance π is represented by one point whose x and y co-ordinates correspond to the performance of A and B on π.

Correlation between median run-time for two SLS algorithms for the TSP over a set of 100 randomly generated instances:



10 runs per instance.

Comparative analysis for instance ensembles (4)

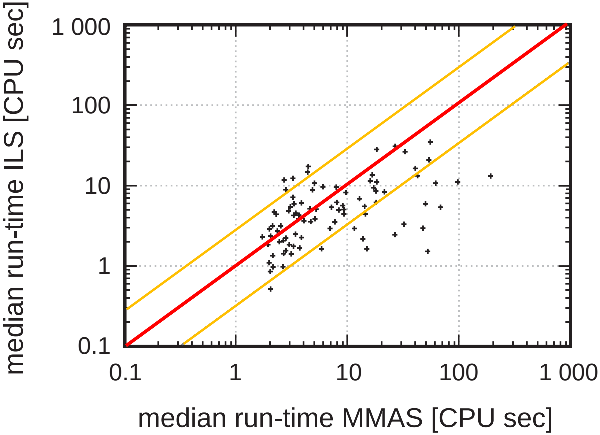
- Quantitatively, the correlation can be summarised using the *empirical correlation coefficient*. Additionally, *regression analysis* can be used to model regular performance relationships.

Comparative analysis for instance ensembles (4)

- Quantitatively, the correlation can be summarised using the *empirical correlation coefficient*. Additionally, *regression analysis* can be used to model regular performance relationships.

- To test the statistical significance of an observed *monotonic* relationship, use non-parametric tests such as *Spearman's rank order test*.

Correlation between median run-time for two SLS algorithms for the TSP over a set of 100 randomly generated instances:



10 runs per instance; correlation coefficient 0.39, significant according to Spearman's rank order test at $\alpha = 0.05$; p-value $= 9 \cdot 10^{-11}$.

Peak Performance *vs* Robustness (1)

- Most high-performance SLS algorithms have parameters that significantly affect their performance.

Peak Performance *vs* Robustness (1)

- Most high-performance SLS algorithms have parameters that significantly affect their performance.

  Examples: Walk probability *wp* in RII, tabu tenure in TS, mutation rate in EAs.

Peak Performance *vs* Robustness (1)

- Most high-performance SLS algorithms have parameters that significantly affect their performance.

  Examples: Walk probability *wp* in RII, tabu tenure in TS, mutation rate in EAs.

- When evaluating parameterised SLS algorithms, *peak performance*, *i.e.*, the performance of a parameterised SLS algorithm for optimised parameter values, is often used as a performance criterion.

Peak Performance *vs* Robustness (1)

- Most high-performance SLS algorithms have parameters that significantly affect their performance.

  Examples: Walk probability *wp* in RII, tabu tenure in TS, mutation rate in EAs.

- When evaluating parameterised SLS algorithms, *peak performance*, *i.e.*, the performance of a parameterised SLS algorithm for optimised parameter values, is often used as a performance criterion.

  Note: Peak performance is a measure of *potential performance*.

Peak Performance *vs* Robustness (1)

- Most high-performance SLS algorithms have parameters that significantly affect their performance.

  Examples: Walk probability *wp* in RII, tabu tenure in TS, mutation rate in EAs.

- When evaluating parameterised SLS algorithms, *peak performance*, *i.e.*, the performance of a parameterised SLS algorithm for optimised parameter values, is often used as a performance criterion.

  Note: Peak performance is a measure of *potential performance*.

- **Pitfall:** *Unfair parameter tuning*, *i.e.*, the use of unevenly optimised parameter settings in comparative studies.

Peak Performance *vs* Robustness (2)

- To avoid unfair parameter tuning, spend approximately the same effort for tuning the parameters of all algorithms participating in a direct performance comparison.

Peak Performance *vs* Robustness (2)

- To avoid unfair parameter tuning, spend approximately the same effort for tuning the parameters of all algorithms participating in a direct performance comparison.

  Alternative: Use *automated parameter tuning techniques* from experimental design.

Peak Performance *vs* Robustness (2)

- To avoid unfair parameter tuning, spend approximately the same effort for tuning the parameters of all algorithms participating in a direct performance comparison.

  Alternative: Use *automated parameter tuning techniques* from experimental design.

- **Note:**
  - Optimal parameter settings often vary substantially between problem instances or instance classes.

Peak Performance *vs* Robustness (2)

- To avoid unfair parameter tuning, spend approximately the same effort for tuning the parameters of all algorithms participating in a direct performance comparison.

  Alternative: Use *automated parameter tuning techniques* from experimental design.

- **Note:**
  - Optimal parameter settings often vary substantially between problem instances or instance classes.
  - Effects of multiple parameters are typically *not* independent.

Peak Performance *vs* Robustness (2)

- To avoid unfair parameter tuning, spend approximately the same effort for tuning the parameters of all algorithms participating in a direct performance comparison.

  Alternative: Use *automated parameter tuning techniques* from experimental design.

- **Note:**
  - Optimal parameter settings often vary substantially between problem instances or instance classes.
  - Effects of multiple parameters are typically *not* independent.

- *Performance robustness*, *i.e.*, the variation in performance due to deviations from optimal parameter settings, is an important performance criterion.

Peak Performance *vs* Robustness (3)

- Performance robustness can be studied empirically by measuring the impact of parameter settings on RTDs (or their descriptive statistics) of a given LVA on a set of problem instances.

Peak Performance *vs* Robustness (3)

- Performance robustness can be studied empirically by measuring the impact of parameter settings on RTDs (or their descriptive statistics) of a given LVA on a set of problem instances.
- More general notions of robustness include performance variation over
  - multiple runs for fixed input (captured in RTD),

Peak Performance *vs* Robustness (3)

- Performance robustness can be studied empirically by measuring the impact of parameter settings on RTDs (or their descriptive statistics) of a given LVA on a set of problem instances.

- More general notions of robustness include performance variation over
  - multiple runs for fixed input (captured in RTD),
  - different problem instances or domains.

Peak Performance *vs* Robustness (3)

- Performance robustness can be studied empirically by measuring the impact of parameter settings on RTDs (or their descriptive statistics) of a given LVA on a set of problem instances.
- More general notions of robustness include performance variation over
  - multiple runs for fixed input (captured in RTD),
  - different problem instances or domains.
- Advanced empirical studies should attempt to relate the latter type of variations to features of the respective instances or domains (*e.g.*, *scaling studies* relate LVA performance to instance size).

Characterising and Improving LVA Behaviour

Advanced aspects of empirical analysis include:

- the analysis of asymptotic and stagnation behaviour,

Characterising and Improving LVA Behaviour

Advanced aspects of empirical analysis include:

- the analysis of asymptotic and stagnation behaviour,
- the use of functional approximations to mathematically characterise entire RTDs.

Characterising and Improving LVA Behaviour

Advanced aspects of empirical analysis include:

- the analysis of asymptotic and stagnation behaviour,
- the use of functional approximations to mathematically characterise entire RTDs.

Such advanced analyses can facilitate improvements in the performance and run-time behaviour of a given LVA

Characterising and Improving LVA Behaviour

Advanced aspects of empirical analysis include:

- the analysis of asymptotic and stagnation behaviour,
- the use of functional approximations to mathematically characterise entire RTDs.

Such advanced analyses can facilitate improvements in the performance and run-time behaviour of a given LVA, *e.g.*, by providing the basis for

- designing or configuring *restart strategies* and other *diversification mechanisms*,

Characterising and Improving LVA Behaviour

Advanced aspects of empirical analysis include:

- the analysis of asymptotic and stagnation behaviour,
- the use of functional approximations to mathematically characterise entire RTDs.

Such advanced analyses can facilitate improvements in the performance and run-time behaviour of a given LVA, *e.g.*, by providing the basis for

- designing or configuring *restart strategies* and other *diversification mechanisms*,
- realising speedups through *multiple independent runs parallelisation*.

Asymptotic behaviour and stagnation

The three previously discussed norms of LVA behaviour, *completeness*, *PAC property* and *essential incompleteness*, correspond to properties of an algorithm's theoretical RTDs.

Asymptotic behaviour and stagnation

The three previously discussed norms of LVA behaviour, *completeness*, *PAC property* and *essential incompleteness*, correspond to properties of an algorithm's theoretical RTDs.

Note:

- *Completeness* can be empirically falsified for a given time-bound, but it cannot empirically verified.

Asymptotic behaviour and stagnation

The three previously discussed norms of LVA behaviour, *completeness*, *PAC property* and *essential incompleteness*, correspond to properties of an algorithm's theoretical RTDs.

Note:

- *Completeness* can be empirically falsified for a given time-bound, but it cannot empirically verified.
- Neither the *PAC property*, nor *essential incompleteness* can be empirically verified or falsified.

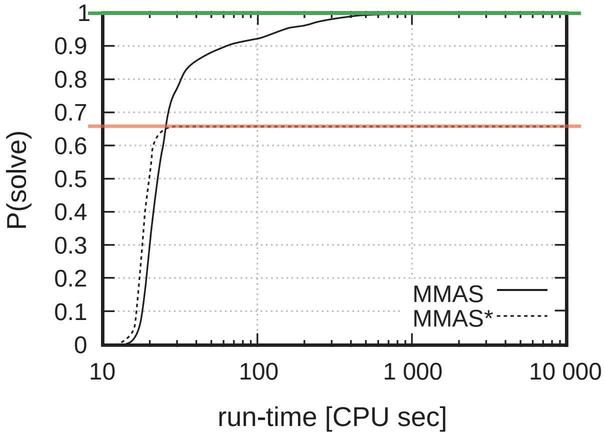
Asymptotic behaviour and stagnation

The three previously discussed norms of LVA behaviour, *completeness*, *PAC property* and *essential incompleteness*, correspond to properties of an algorithm's theoretical RTDs.

Note:

- *Completeness* can be empirically falsified for a given time-bound, but it cannot empirically verified.
- Neither the *PAC property*, nor *essential incompleteness* can be empirically verified or falsified.
- **But:** Empirical RTDs can provide *evidence* (rather than proof) for essential incompleteness or PAC behaviour.

Example of asymptotic behaviour in empirical RTDs:



Note: $\mathcal{MM}$AS is provably PAC, $\mathcal{MM}$AS* is essentially incomplete.

LVA efficiency and stagnation

- In practice, the *rate of decrease in the failure probability*, $\lambda_{A,\pi}(t)$, is more relevant than true asymptotic behaviour.

LVA efficiency and stagnation

- In practice, the *rate of decrease in the failure probability*, $\lambda_{A,\pi}(t)$, is more relevant than true asymptotic behaviour.
- **Note:** *Exponential RTDs* are characterised by a *constant rate of decrease in failure probability*.

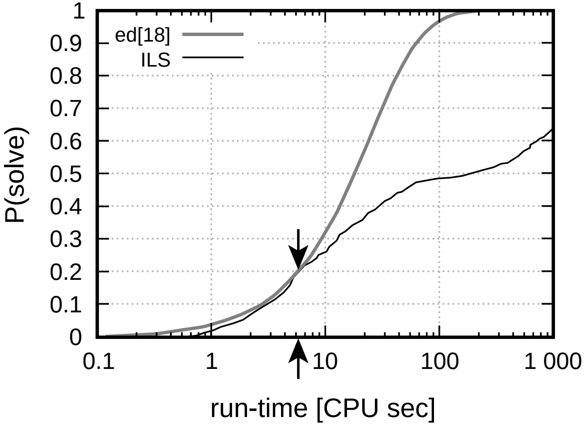
LVA efficiency and stagnation

- In practice, the *rate of decrease in the failure probability*, $\lambda_{A,\pi}(t)$, is more relevant than true asymptotic behaviour.
- **Note:** *Exponential RTDs* are characterised by a *constant rate of decrease in failure probability*.
- A drop in $\lambda_{A,\pi}(t)$ indicates *stagnation* of algorithm A's progress towards finding a solution of instance π.

LVA efficiency and stagnation

- In practice, the *rate of decrease in the failure probability*, $\lambda_{A,\pi}(t)$, is more relevant than true asymptotic behaviour.
- **Note:** *Exponential RTDs* are characterised by a *constant rate of decrease in failure probability*.
- A drop in $\lambda_{A,\pi}(t)$ indicates *stagnation* of algorithm A's progress towards finding a solution of instance π.
- Stagnation can be detected by comparing the RTD against an exponential distribution.

Evidence of stagnation in an empirical RTD:



'ed[18]' is the CDF of an exponential distribution with median 18; the arrows mark the point at which stagnation behaviour becomes apparent.

Note:

- The formal definition of LVA *efficiency* and *stagnation* is based on the idea that an LVA A suffers from stagnation iff its success probability can be increased by restarting A after an appropriately chosen cutoff time.

  (For details, see Definition 4.9 on page 187 of SLS:FA.)

Note:

- The formal definition of LVA *efficiency* and *stagnation* is based on the idea that an LVA A suffers from stagnation iff its success probability can be increased by restarting A after an appropriately chosen cutoff time.

  (For details, see Definition 4.9 on page 187 of SLS:FA.)

- Efficiency and stagnation are *relative measures*; they cannot indicate all situations in which a given LVA's behaviour can be further improved.

Functional characterisation of LVA behaviour (1)

- Empirical RTDs are step functions that approximate the underlying theoretical RTDs.

Functional characterisation of LVA behaviour (1)

- Empirical RTDs are step functions that approximate the underlying theoretical RTDs.

- For reasonably large sample sizes (numbers of runs), empirical RTDs can often be approximated well using much simpler continuous mathematical functions.

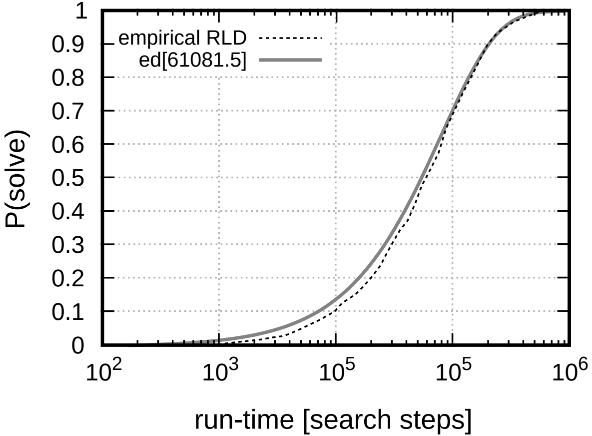
Functional characterisation of LVA behaviour (1)

- Empirical RTDs are step functions that approximate the underlying theoretical RTDs.
- For reasonably large sample sizes (numbers of runs), empirical RTDs can often be approximated well using much simpler continuous mathematical functions.
- Such functional approximations are useful for summarising and mathematically modelling empirically observed behaviour, which often provides deeper insights into LVA behaviour.

Functional characterisation of LVA behaviour (1)

- Empirical RTDs are step functions that approximate the underlying theoretical RTDs.
- For reasonably large sample sizes (numbers of runs), empirical RTDs can often be approximated well using much simpler continuous mathematical functions.
- Such functional approximations are useful for summarising and mathematically modelling empirically observed behaviour, which often provides deeper insights into LVA behaviour.
- Approximations with parameterised families of continuous distribution functions known from statistics, such as exponential or normal distributions, are particularly useful.

Approximation of an empirical RTD with an exponential distribution $ed[m](x) := 1 - 2^{-x/m}$:



Functional characterisation of LVA behaviour (2)

- *Model fitting techniques*, such as the *Marquardt-Levenberg* or *Expectation Maximisation algorithms*, can be used to find good approximations of empirical RTDs with parameterised cumulative distribution functions.

Functional characterisation of LVA behaviour (2)

- *Model fitting techniques*, such as the *Marquardt-Levenberg* or *Expectation Maximisation algorithms*, can be used to find good approximations of empirical RTDs with parameterised cumulative distribution functions.

- The quality of approximations can be assessed using *statistical goodness-of-fit tests*, such as the χ^2-*test* or the *Kolmogorov-Smirnov test*.

Functional characterisation of LVA behaviour (2)

- *Model fitting techniques*, such as the *Marquardt-Levenberg* or *Expectation Maximisation algorithms*, can be used to find good approximations of empirical RTDs with parameterised cumulative distribution functions.

- The quality of approximations can be assessed using *statistical goodness-of-fit tests*, such as the χ^2-*test* or the *Kolmogorov-Smirnov test*.

- **Note:** Particularly for small or easy problem instances, the quality of optimal functional approximations can sometimes be limited by the inherently discrete nature of empirical RTD data.

Functional characterisation of LVA behaviour (2)

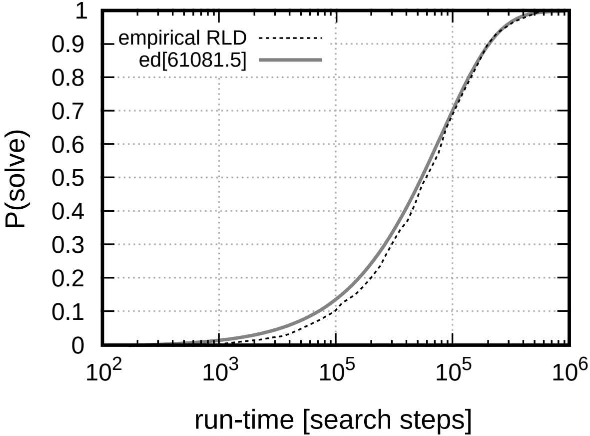
- *Model fitting techniques*, such as the *Marquardt-Levenberg* or *Expectation Maximisation algorithms*, can be used to find good approximations of empirical RTDs with parameterised cumulative distribution functions.

- The quality of approximations can be assessed using *statistical goodness-of-fit tests*, such as the χ^2-*test* or the *Kolmogorov-Smirnov test*.

- **Note:** Particularly for small or easy problem instances, the quality of optimal functional approximations can sometimes be limited by the inherently discrete nature of empirical RTD data.

- This approach can be easily generalised to ensembles of problem instances.

Approximation of an empirical RTD with an exponential distribution $\mathrm{ed}[m](x) := 1 - 2^{-x/m}$:



The optimal fit exponential distribution obtained from the Marquardt-Levenberg algorithm passes the χ^2 goodness-of-fit test at $\alpha = 0.05$.

Performance improvements based on static restarts (1)

- Detailed RTD analyses can often suggest ways of improving the performance of a given SLS algorithm.

Performance improvements based on static restarts (1)

- Detailed RTD analyses can often suggest ways of improving the performance of a given SLS algorithm.
- *Static restarting*, *i.e.*, periodic re-initialisation after all integer multiples of a given cutoff-time t', is one of the simplest methods for overcoming stagnation behaviour.

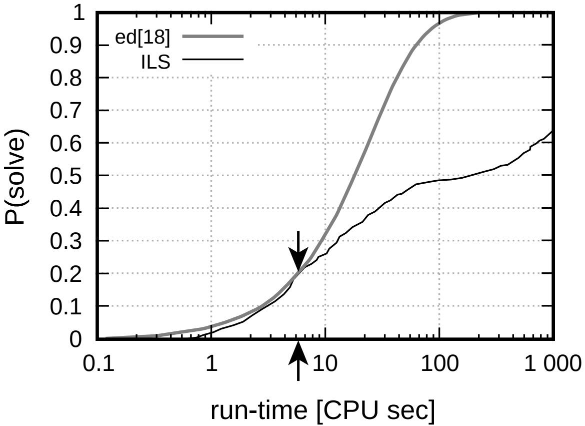
Performance improvements based on static restarts (1)

- Detailed RTD analyses can often suggest ways of improving the performance of a given SLS algorithm.

- *Static restarting*, *i.e.*, periodic re-initialisation after all integer multiples of a given cutoff-time t', is one of the simplest methods for overcoming stagnation behaviour.

- A static restart strategy is effective, *i.e.*, leads to increased solution probability for some run-time t'', if the RTD of the given algorithm and problem instance is less steep than an exponential distribution crossing the RTD at some time $t < t''$.

Example of an empirical RTD of an SLS algorithm on a problem instance for which static restarting is effective:



'ed[18]' is the CDF of an exponential distribution with median 18; the arrows mark the optimal cutoff-time for static restarting.

Performance improvements based on static restarts (2)

- To determine the optimal cutoff-time t_{opt} for static restarts, consider the left-most exponential distribution that touches the given empirical RTD and choose t_{opt} to be the smallest t value at which the two respective distribution curves meet.

  (For a formal derivation of t_{opt}, see page 193 of SLS:FA.)

Performance improvements based on static restarts (2)

- To determine the optimal cutoff-time t_{opt} for static restarts, consider the left-most exponential distribution that touches the given empirical RTD and choose t_{opt} to be the smallest t value at which the two respective distribution curves meet.

  (For a formal derivation of t_{opt}, see page 193 of SLS:FA.)

- **Note:** This method for determining optimal cutoff-times only works *a posteriori*, given an empirical RTD.

Performance improvements based on static restarts (2)

- To determine the optimal cutoff-time t_{opt} for static restarts, consider the left-most exponential distribution that touches the given empirical RTD and choose t_{opt} to be the smallest t value at which the two respective distribution curves meet.

  (For a formal derivation of t_{opt}, see page 193 of SLS:FA.)

- **Note:** This method for determining optimal cutoff-times only works *a posteriori*, given an empirical RTD.

- Optimal cutoff-times for static restarting typically vary considerably between problem instances; for optimisation algorithms, they also depend on the desired solution quality.

Overcoming stagnation using dynamic restarts

- *Dynamic restart strategies* are based on the idea of re-initialising the search process only when needed, *i.e.*, when stagnation occurs.

Overcoming stagnation using dynamic restarts

- *Dynamic restart strategies* are based on the idea of re-initialising the search process only when needed, *i.e.*, when stagnation occurs.

- *Simple dynamic restart strategy:* Re-initialise search when the time interval since the last improvement of the incumbent candidate solution exceeds a given threshold θ. (Incumbent candidate solutions are not carried over restarts.)

Overcoming stagnation using dynamic restarts

- *Dynamic restart strategies* are based on the idea of re-initialising the search process only when needed, *i.e.*, when stagnation occurs.

- *Simple dynamic restart strategy:* Re-initialise search when the time interval since the last improvement of the incumbent candidate solution exceeds a given threshold θ. (Incumbent candidate solutions are not carried over restarts.)

  θ is typically measured in search steps and may be chosen depending on properties of the given problem instance, in particular, instance size.

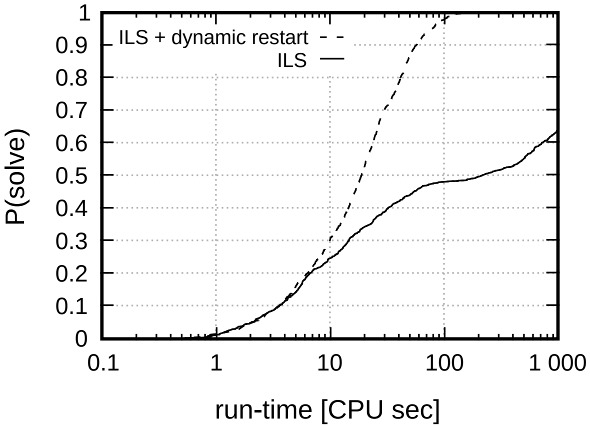
Overcoming stagnation using dynamic restarts

- *Dynamic restart strategies* are based on the idea of re-initialising the search process only when needed, *i.e.*, when stagnation occurs.

- *Simple dynamic restart strategy:* Re-initialise search when the time interval since the last improvement of the incumbent candidate solution exceeds a given threshold θ.
  (Incumbent candidate solutions are not carried over restarts.)

  θ is typically measured in search steps and may be chosen depending on properties of the given problem instance, in particular, instance size.

Example: Effect of simple dynamic restart strategy



Other diversification strategies

- Restart strategies often suffer from the fact that search initialisation can be relatively time-consuming (setup time, time required for reaching promising regions of given search space).

Other diversification strategies

- Restart strategies often suffer from the fact that search initialisation can be relatively time-consuming (setup time, time required for reaching promising regions of given search space).
- This problem can be avoided by using other diversification mechanisms for overcoming search stagnation, such as
  - *random walk extensions* that render a given SLS algorithm provably PAC;

Other diversification strategies

- Restart strategies often suffer from the fact that search initialisation can be relatively time-consuming (setup time, time required for reaching promising regions of given search space).
- This problem can be avoided by using other diversification mechanisms for overcoming search stagnation, such as
  - *random walk extensions* that render a given SLS algorithm provably PAC;
  - adaptive modification of parameters controlling the amount of search diversification, such as temperature in SA or tabu tenure in TS.

Other diversification strategies

- Restart strategies often suffer from the fact that search initialisation can be relatively time-consuming (setup time, time required for reaching promising regions of given search space).
- This problem can be avoided by using other diversification mechanisms for overcoming search stagnation, such as
  - *random walk extensions* that render a given SLS algorithm provably PAC;
  - adaptive modification of parameters controlling the amount of search diversification, such as temperature in SA or tabu tenure in TS.
- Effective techniques for overcoming search stagnation are crucial components of high-performance SLS methods.

Multiple independent runs parallelisation

- Any LVA A can be easily parallelised by performing multiple runs on the same problem instance π in parallel on p processors.

Multiple independent runs parallelisation

- Any LVA A can be easily parallelised by performing multiple runs on the same problem instance π in parallel on p processors.
- The effectiveness of this approach depends on the RTD of A on π:

Multiple independent runs parallelisation

- Any LVA A can be easily parallelised by performing multiple runs on the same problem instance π in parallel on p processors.

- The effectiveness of this approach depends on the RTD of A on π:

  Optimal parallelisation speedup of p is achieved for an exponential RTD.

Multiple independent runs parallelisation

- Any LVA A can be easily parallelised by performing multiple runs on the same problem instance π in parallel on p processors.

- The effectiveness of this approach depends on the RTD of A on π:

  Optimal parallelisation speedup of p is achieved for an exponential RTD.

- The RTDs of many high-performance SLS algorithms are well approximated by exponential distributions; however, deviations for short run-times (due to the effects of search initialisation) limit the maximal number of processors for which optimal speedup can be achieved in practice.

Speedup achieved by multiple independent runs parallelisation of a high-performance SLS algorithm for SAT: