HEURISTIC OPTIMIZATION

Empirical Analysis of SLS Algorithms

adapted and extended from slides for SLS:FA, Chapter 4

Analysis of SLS algorithms

- How long does it take to find a feasible solution?
- How good are the solutions generated by the algorithm?
- Which neighborhood structure should I choose?
- How robust is this heuristic algorithm w.r.t. different instances?
- How robust is this heuristic algorithm w.r.t. different parameter settings?
- How does the algorithm scale to large instance size?
- Which is the best of these 4 heuristic algorithms?

- ideally: do theoretical analysis, but:
- the theoretical analysis of detailed behavior of (advanced) SLS algorithms is often very difficult due to
  - stochasticity of algorithms
  - complexity of problems tackled ($\mathcal{NP}$-hard)
  - many degrees of freedom in algorithms which make them theoretically hard to capture
- practical applicability of theoretical results often limited
  - rely on idealised assumptions that do not apply in practical situations (*e.g.*, convergence results for Simulated Annealing)
  - capture only *asymptotic behaviour* and do not reflect *actual behaviour* with sufficient accuracy
  - apply to *worst-case* or *highly idealised average-case*

Therefore:

Analyse the behaviour of SLS algorithms using sound *empirical methodologies*

Example: follow a scientific procedure:

- 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

Goals of empirical analysis:

Obtain insights into algorithmic performance

- help assessing suitability for applications
- provide basis for comparing algorithms
- characterise algorithm behavior
- facilitate improvements of algorithms

Examine aspects of stochastic search performance

- variability due to randomization
- robustness w.r.t. parameter settings
- robustness across different instances / instance types
- scaling with instance size

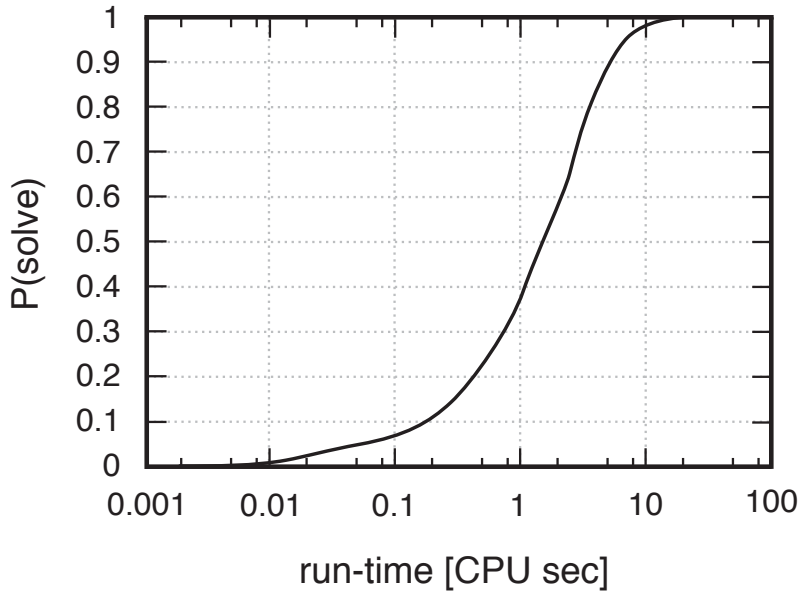
Run-Time Distributions

Empirical analysis of SLS algorithms

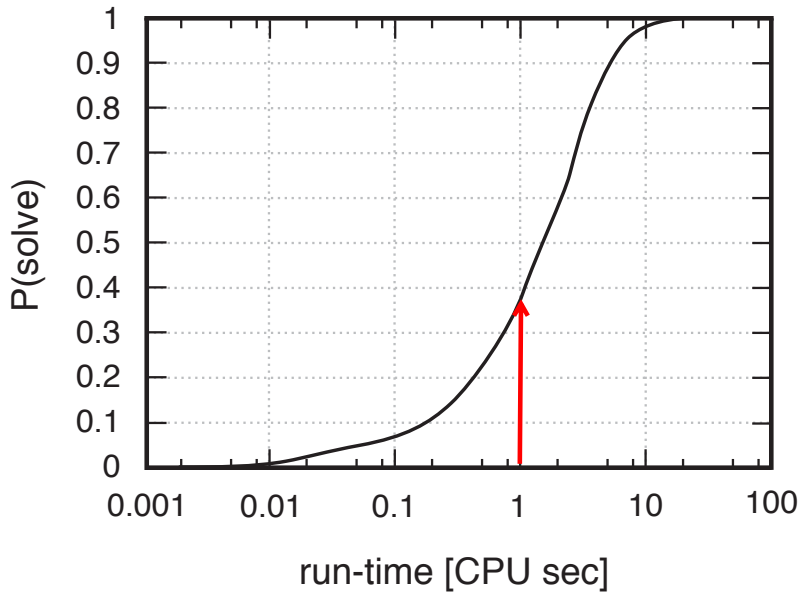
- traditional approach
  - run SLS algorithm n times with time limit t_{max}
  - compute averages, standard deviations
- more detailed point of view
  - for decision SLS algorithms the run time to reach a solution is a random variable
    $\rightsquigarrow$ (univariate) run-time distributions (RTDs)
  - for optimisation SLS algorithms the run time and the solution quality returned are random variables
    $\rightsquigarrow$ (bivariate) run-time distributions (RTDs)

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

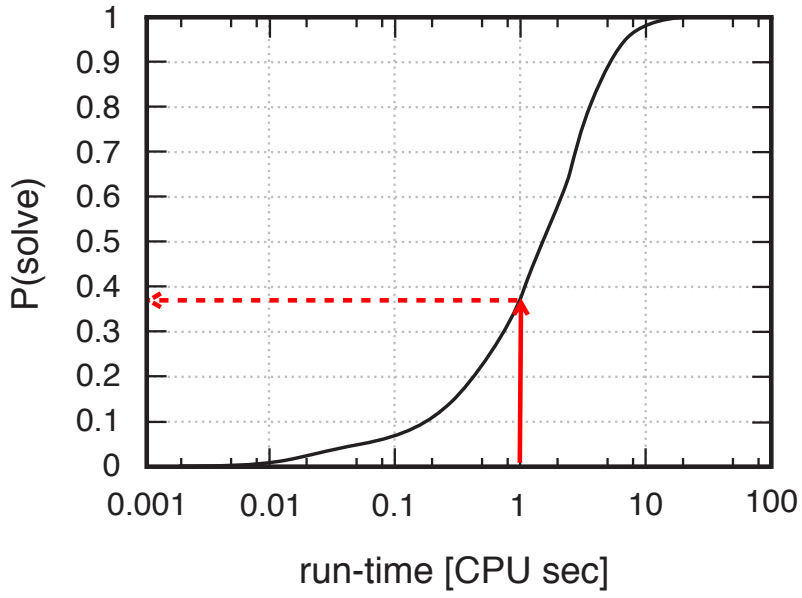
Example of run-time distribution for SLS algorithm applied to hard instance of combinatorial decision problem:



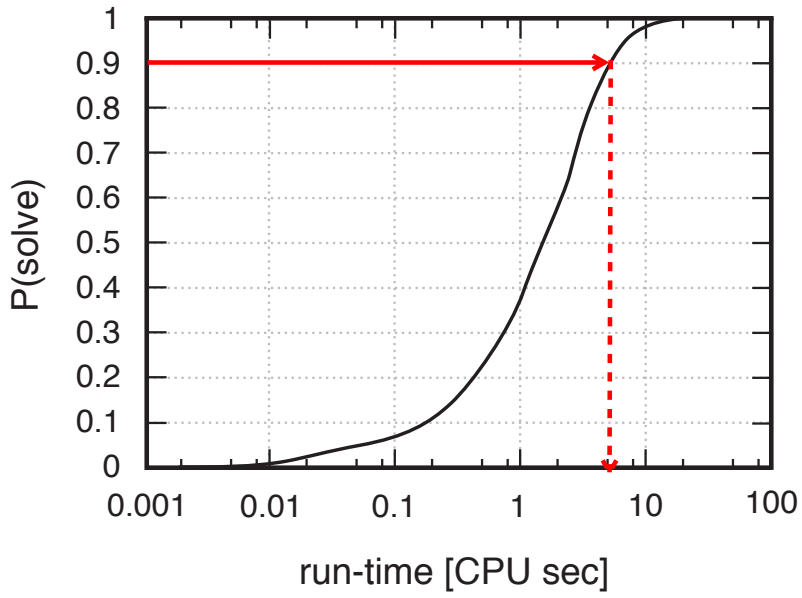
Example of run-time distribution for SLS algorithm applied to hard instance of combinatorial decision problem:



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



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



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

Given SLS algorithm A for decision problem Π:

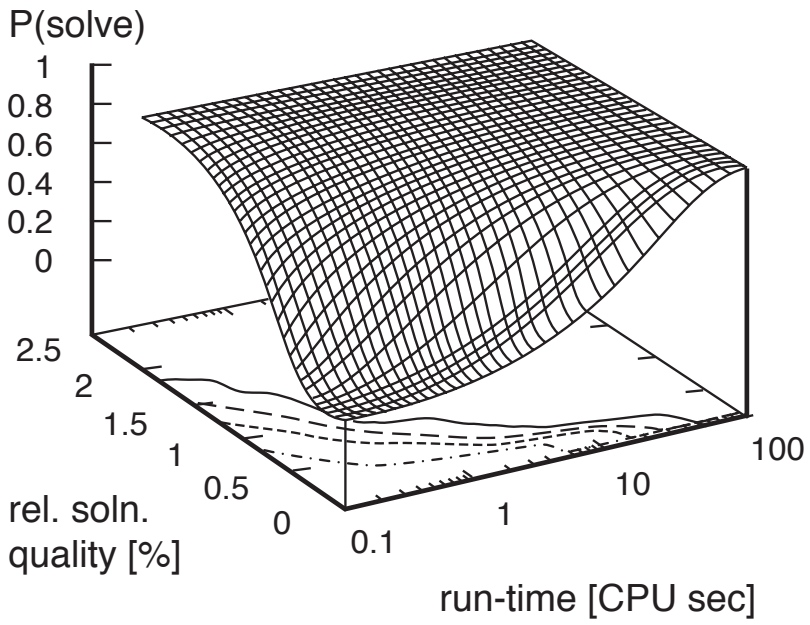
- The *success probability* $P_s(RT_{A,\pi} \leq t)$ is the probability that A finds a solution for a soluble instance $\pi \in \Pi$ in time $\leq t$.
- The *run-time distribution (RTD) of A on π* is the probability distribution of the random variable $RT_{A,\pi}$.
- The *run-time distribution function*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

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

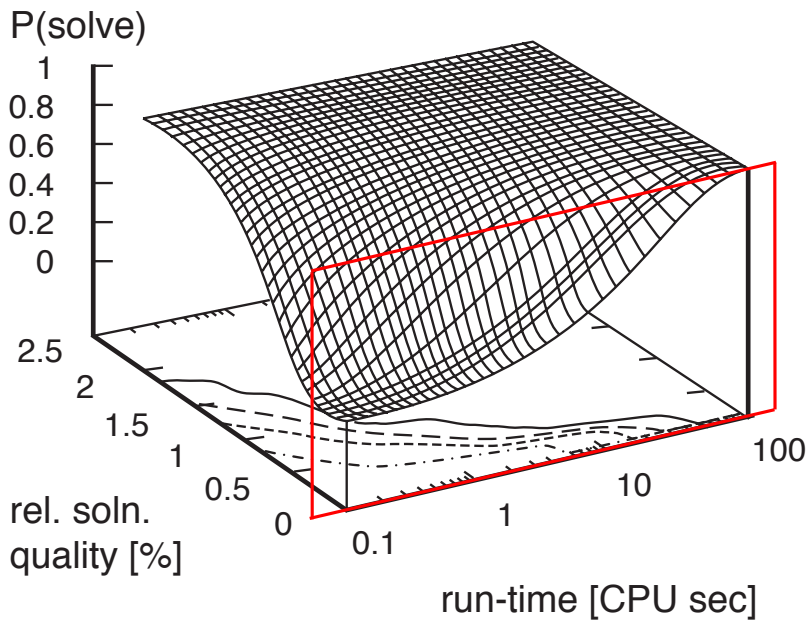
Given SLS algorithm A' for optimisation problem Π':

- 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'.

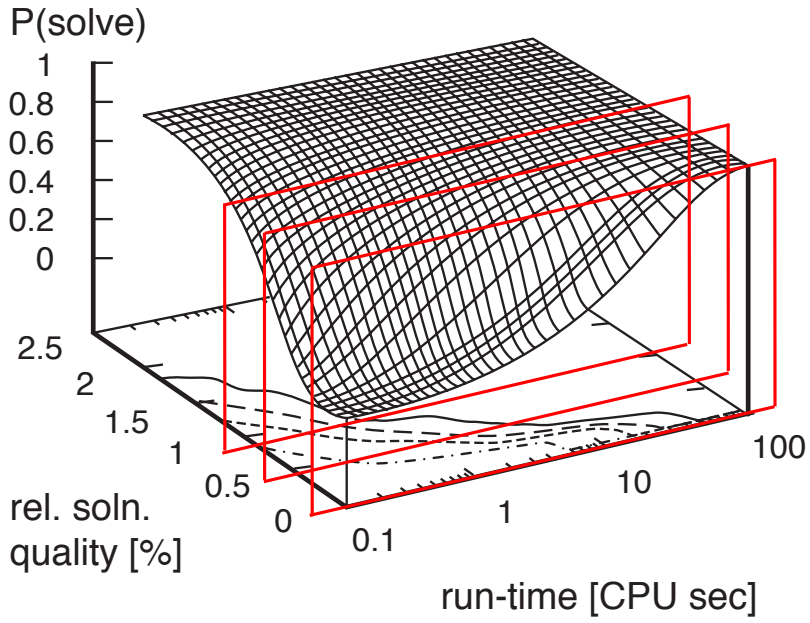
Example of run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



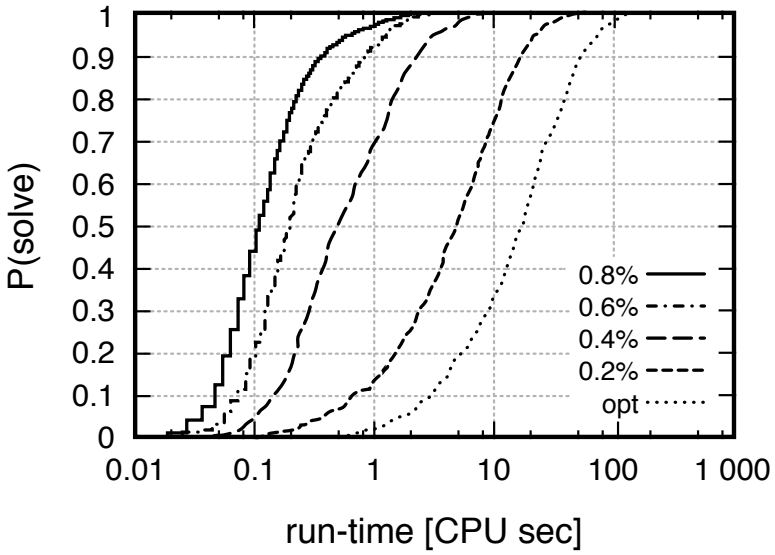
Example of run-time distribution for SLS algorithm applied to hard instance of combinatorial optimisation problem:



Example of 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 SLS algorithm 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:

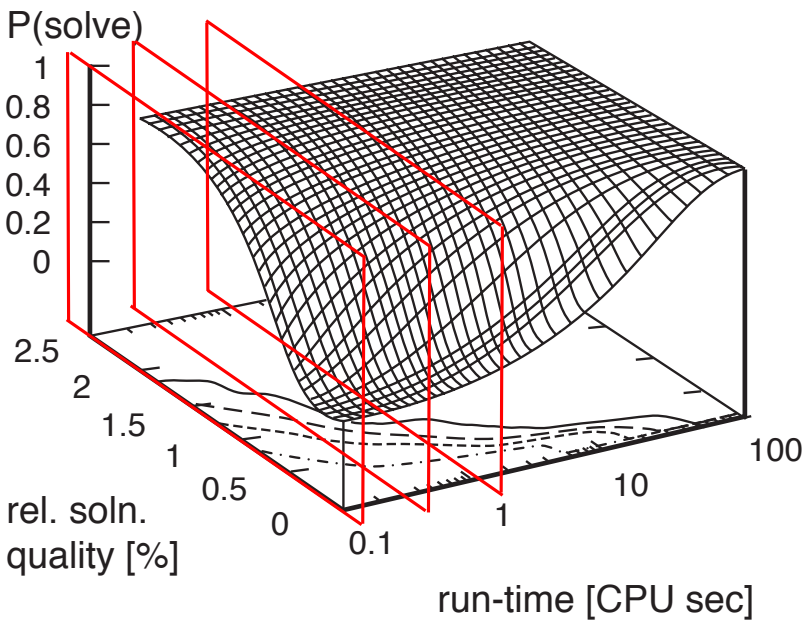
  $$qrtd_{q'}(t) := rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q').$$

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

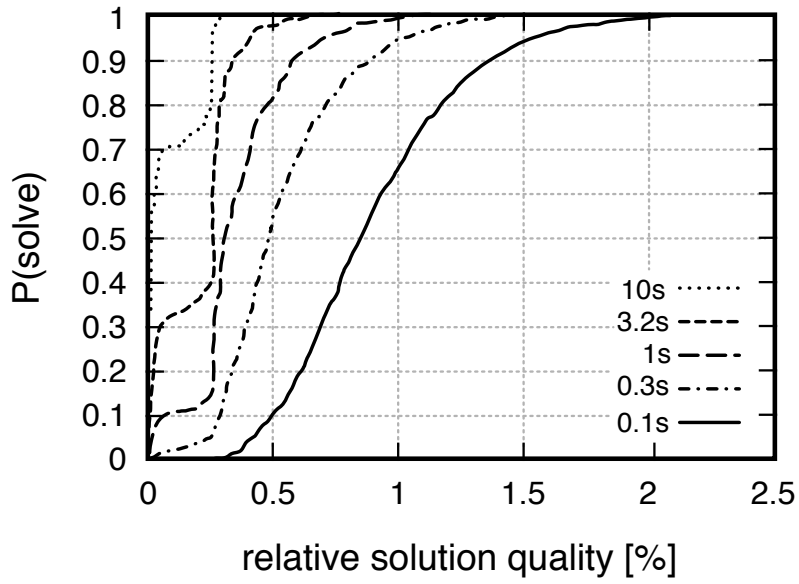
- QRTDs characterise the ability of a given SLS algorithm for a combinatorial optimisation problem to solve the associated decision problems.

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 SLS algorithm 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

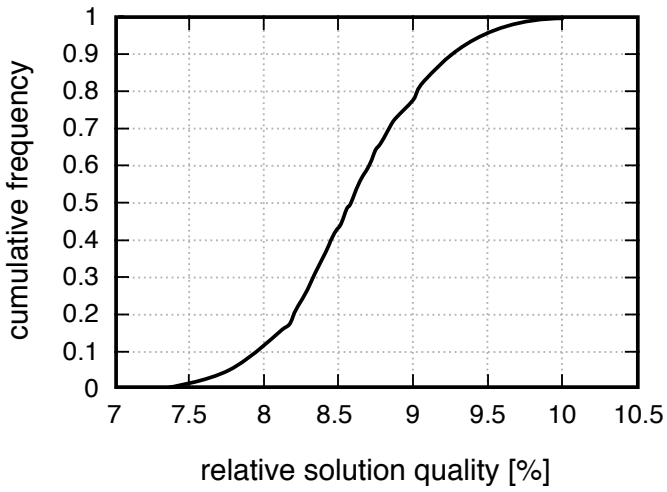
Note:

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

- For convergent SLS algorithms, the SQDs for very large time-limits t' approach degenerate distributions that concentrate all probability on the optimal solution quality.

- For any incomplete SLS 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'*.

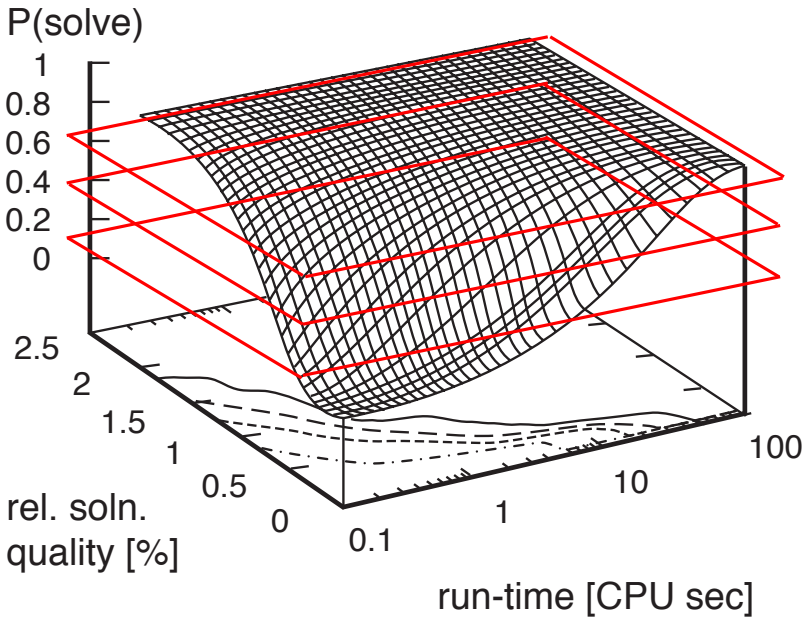
example: asymptotic SQD of Random-order first improvement for TSP instance pcb3038



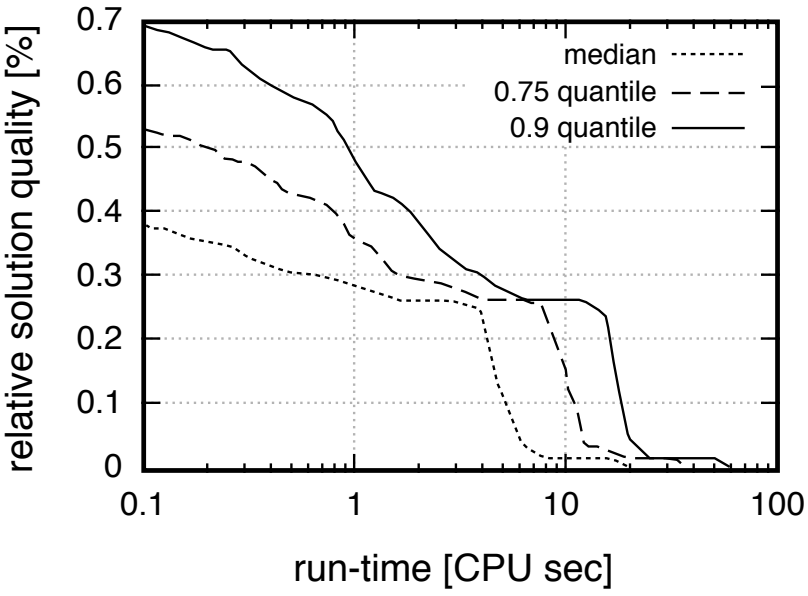
Solution quality statistics over time (SQTs)

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



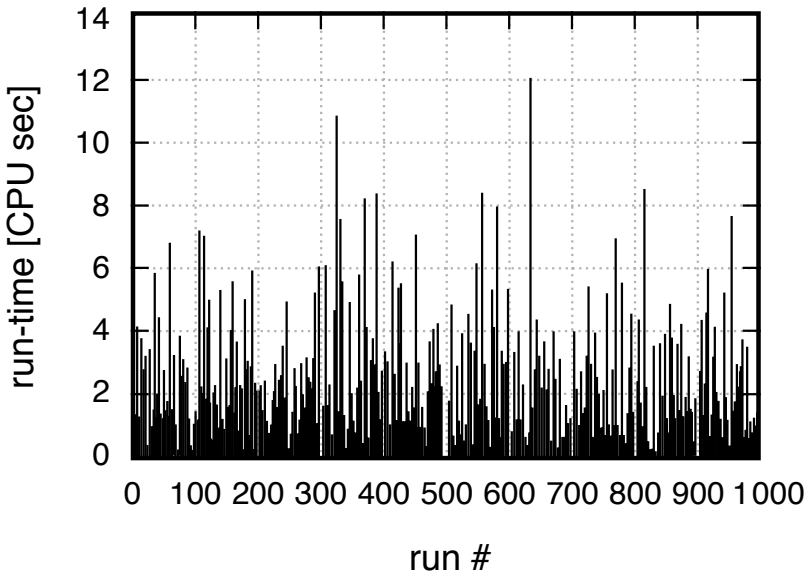
Typical SQT curves for SLS optimisation algorithms applied to instance of hard combinatorial optimisation problem:



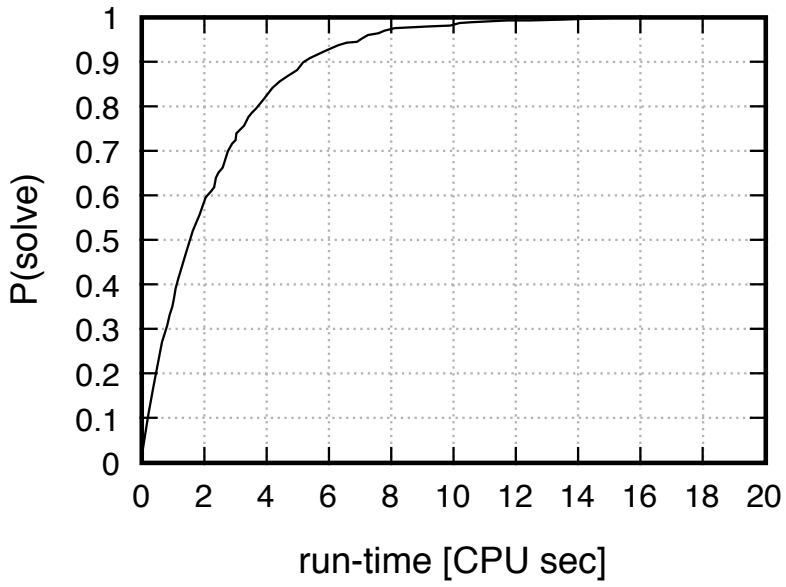
Empirically measuring RTDs

- Except for very simple algorithms, where they can be derived analytically, RTDs are measured empirically.
- 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 true 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 SLS algorithm 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* $p_s^* := lim_{t\to\infty} P_s(RT_{a,\pi} \leq t)$.
- 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 SLS algorithm 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 $\widehat{P}_s(RT \leq t', SQ \leq q') := \#\{j \mid sq(t', j) \leq q'\}/k$.

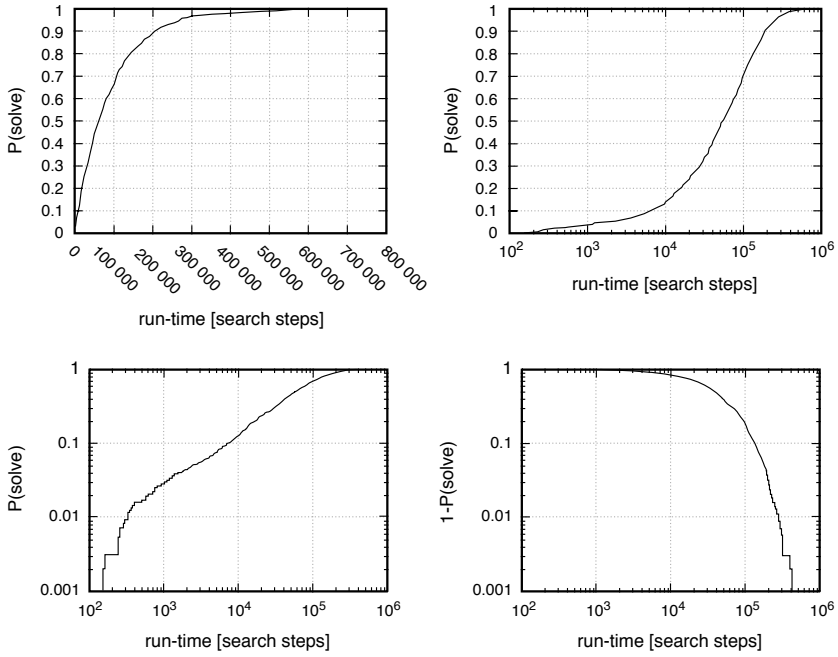
Note: Qualified RTDs, SQDs and SQT curves can be easily derived from the same solution quality traces.

```
Parameter-settings:
max-tries 5
time 10.000000
num-ants 25
num-neigh 20
alpha 1.000000
beta 2.000000
rho 0.500000
q_0 0.000000
ls_flag 3
nn_ls 20
dlb_flag 1
mmas_flag 1
begin problem /Users/stuetzle/Benchmarks/TSP/lin318.tsp
seed 1335270635
begin try 0
best 42814 iteration 1 tours 27 time 0.020
best 42775 iteration 3 tours 77 time 0.050
best 42711 iteration 4 tours 102 time 0.060
best 42582 iteration 7 tours 177 time 0.110
best 42362 iteration 8 tours 202 time 0.120
best 42253 iteration 10 tours 252 time 0.140
best 42199 iteration 11 tours 277 time 0.150
best 42181 iteration 26 tours 652 time 0.280
best 42159 iteration 30 tours 752 time 0.320
best 42143 iteration 34 tours 852 time 0.360
best 42101 iteration 1021 tours 25527 time 9.120
best 42080 iteration 1024 tours 25602 time 9.160
best 42050 iteration 1025 tours 25627 time 9.170
best 42029 iteration 1027 tours 25677 time 9.190
end try 0
seed 1528632406
begin try 1
best 42747 iteration 1 tours 27 time 0.020
best 42681 iteration 4 tours 102 time 0.060
```


Different views of RTD plots are useful for the *qualitative analysis* of SLS 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:



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, 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*);
- 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)*.
- **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.

Some Usages of RTDs

Uses of empirical analysis through RTDs 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 SLS, *e.g.*, by providing the basis for

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

Asymptotic run-time behaviour of SLS algorithms

- *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 $\to 1$ as run-time $\to \infty$.

  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 $\to \infty$.

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 Best 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.

  $\rightsquigarrow$ 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.

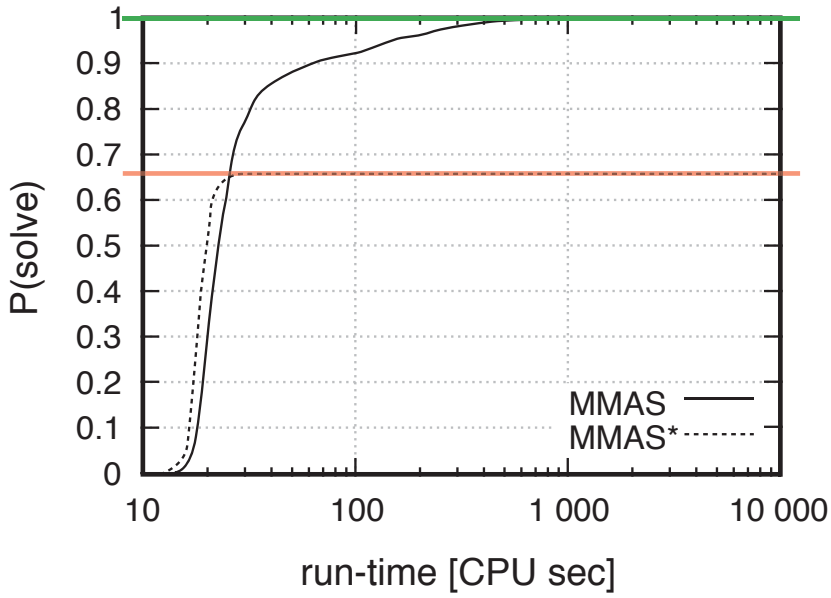
Asymptotic behaviour and stagnation

The three norms of SLS algorithm 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:

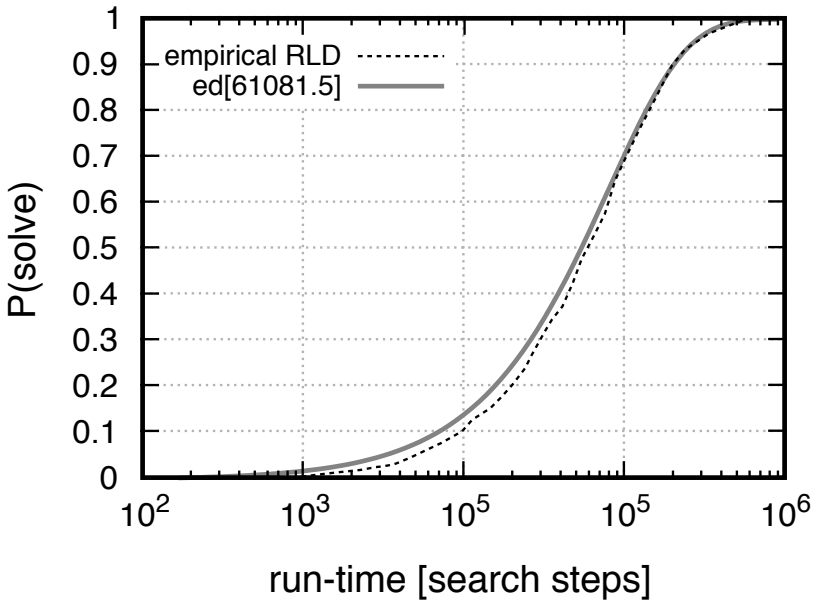


Note: $\mathcal{MM}$AS is provably PAC, $\mathcal{MM}$AS* is essentially incomplete.

Functional characterisation of SLS 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 SLS algorithm behaviour.
- Approximations with parameterised families of continuous distribution functions known from statistics, such as exponential or normal distributions, are particularly useful.

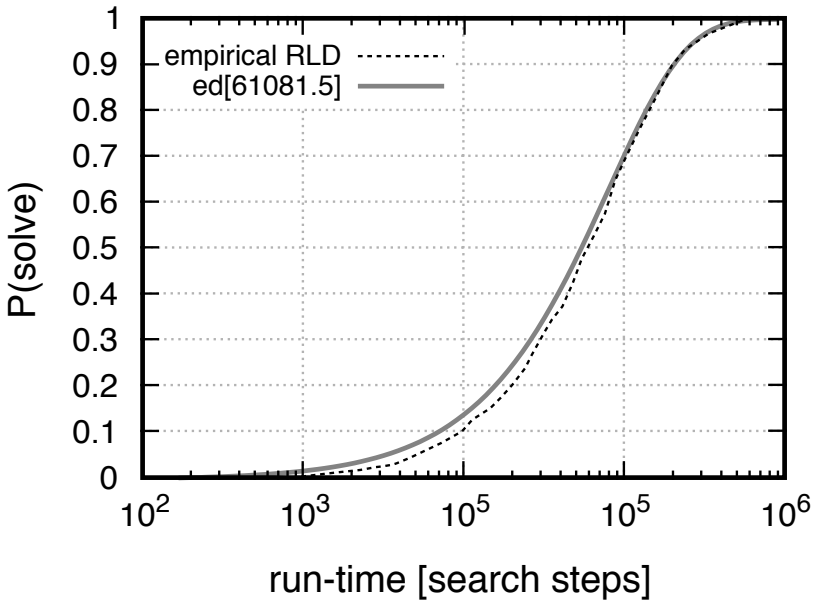
Approximation of an empirical RTD with an exponential distribution $\mathrm{ed}[m](x) := 1 - 2^{-x/m}$:



Functional characterisation of SLS behaviour (2)

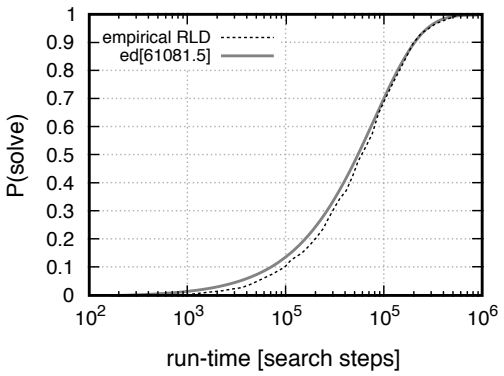
- *Model fitting techniques*, such as the *Marquardt-Levenberg* or *Expectation Maximisation algorithms*, can be used to find good approximations of empirical RTDs with parameterised cumulative distribution functions.
- The quality of approximations can be assessed using *statistical goodness-of-fit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

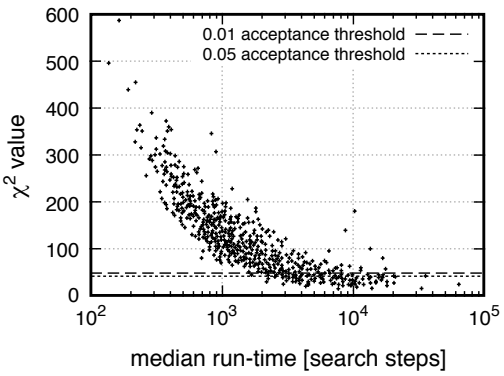
Approximation of an empirical RTD with an exponential distribution $\mathrm{ed}[m](x) := 1 - 2^{-x/m}$:



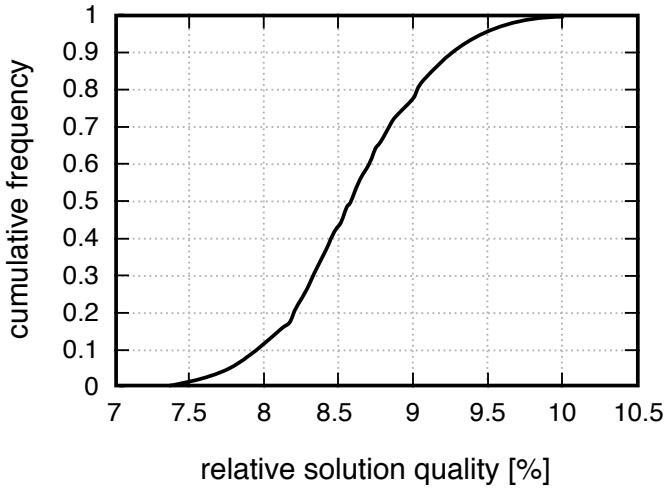
The optimal fit exponential distribution obtained from the Marquardt-Levenberg algorithm passes the χ^2 goodness-of-fit test at $\alpha = 0.05$.

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





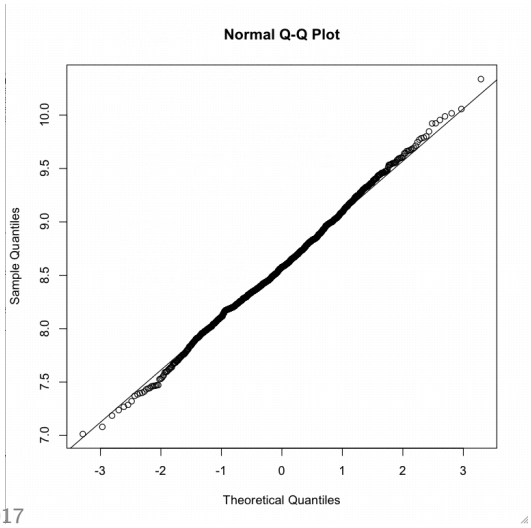
example: asymptotic SQD of random-order first improvement for TSP instance pcb3038



hypothesis: solution quality data follow a normal distribution
The hypothesis can be tested using the Shapiro-Wilk test; test does not reject hypothesis that data follow a normal distribution with mean 8,6 and std.deviation 0.51 (p-value 0.2836.

Q-Q plot

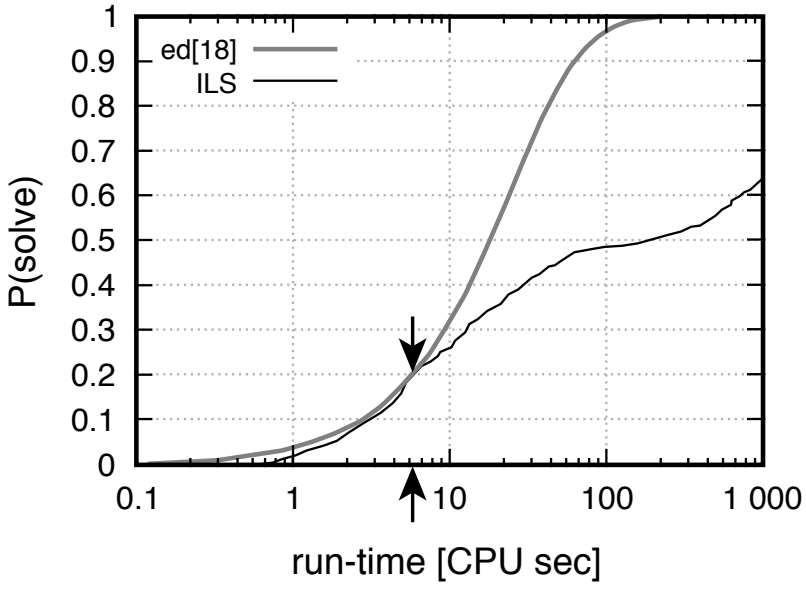
- graphical method for comparing two probability distributions by plotting their quantiles against each other
- if two distributions compared are similar, the points are roughly on a line



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 $t < t''$.

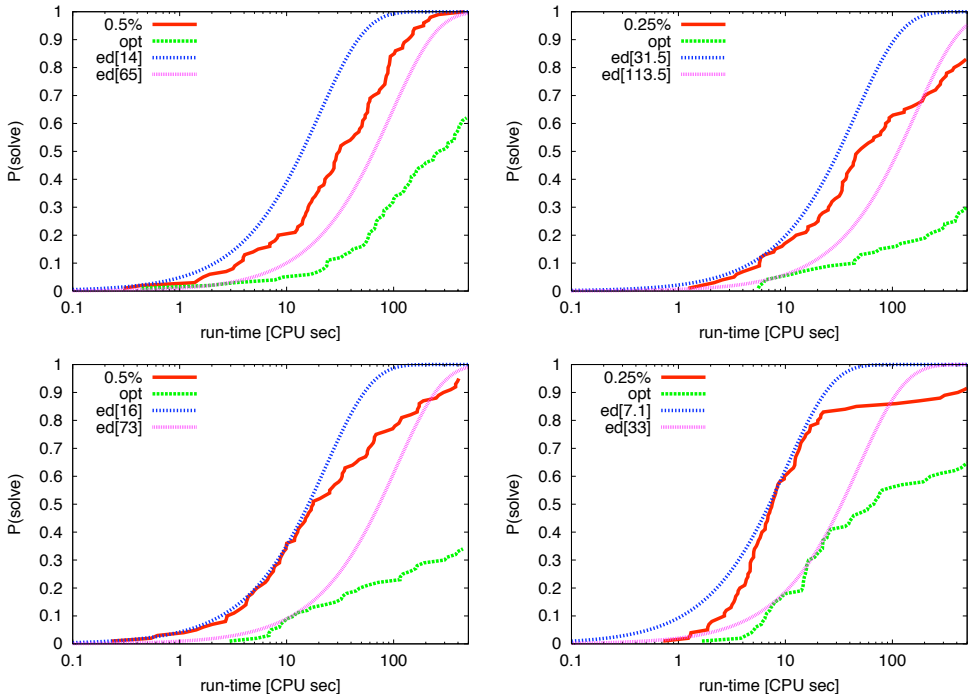
Example of an empirical RTD of an SLS algorithm on a problem instance for which static restarting is effective:



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

Improvements – check on several instances

Basic ILS for QAP (better, VNS perturbation, 2-exchange LS)



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.

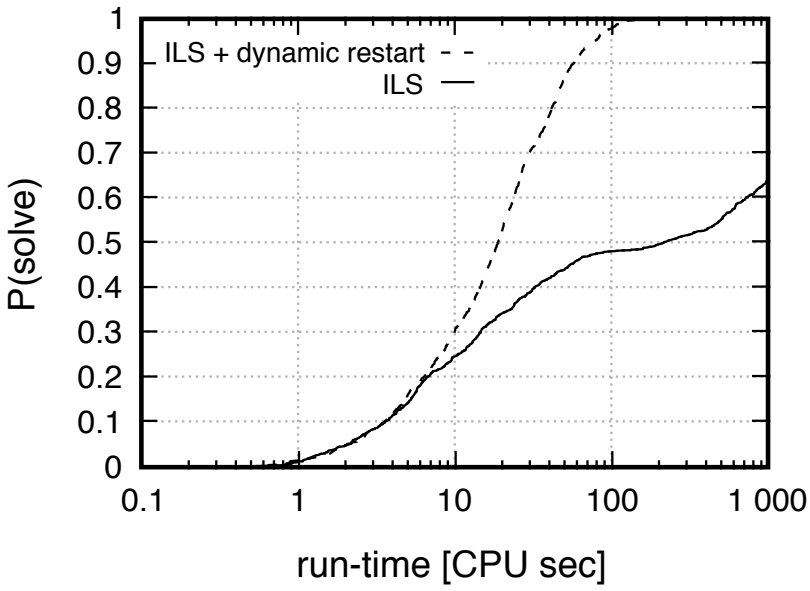
- Optimal cutoff-times for static restarting typically vary considerably between problem instances; for optimisation algorithms, they also depend on the desired solution quality.

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.
- *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

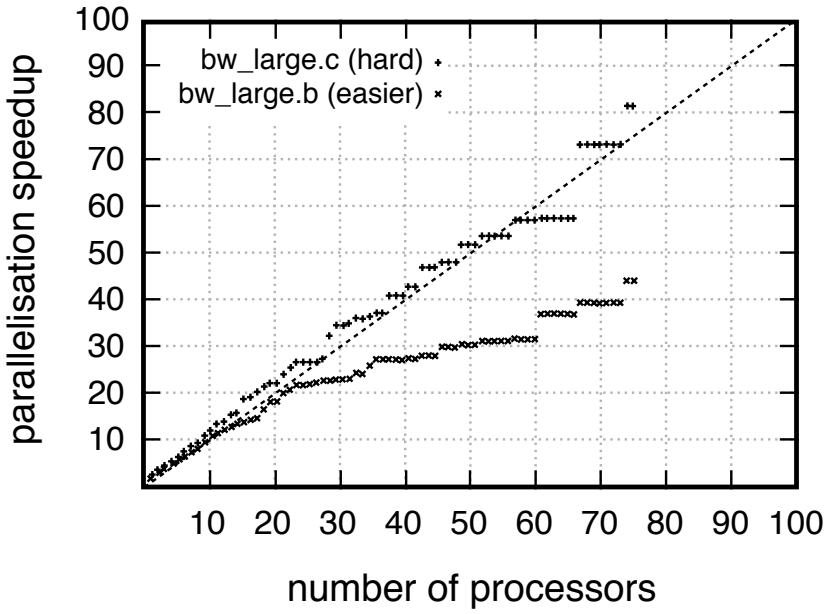
- Restart strategies often suffer from the fact that search initialisation can be relatively time-consuming (setup time, time required for reaching promising regions of given search space).
- This problem can be avoided by using other diversification mechanisms for overcoming search stagnation, such 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 SLS algorithm 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:



Summary descriptive statistics

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: Due to stochasticity of SLS algorithms, reporting measures of variability is important

Example:

For a given SLS algorithm for SAT applied to a specific SAT instance we observe the following basic descriptive statistics for the RLD

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.
- Unlike the standard deviation (or variance), the variation coefficient and quantile ratios are invariant under multiplication by constants.
- Descriptive statistics can be easily calculated from empirical RTDs.
- Obtaining sufficiently stable descriptive statistics requires a similar number of runs as measuring reasonably accurate RTDs.
- QRTDs and SQDs can be handled analogously to RTDs.

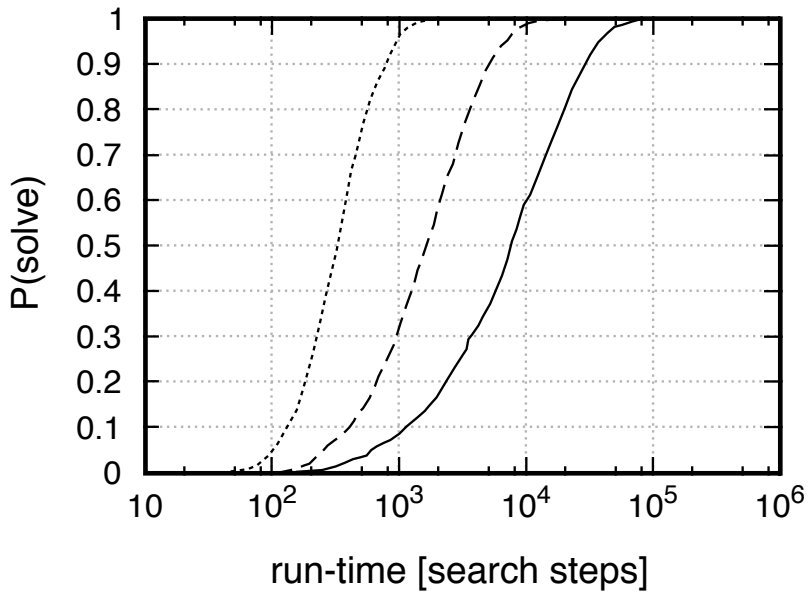
Analyses on instance ensembles and comparison of algorithms

Basic quantitative analysis for ensembles of instances (1)

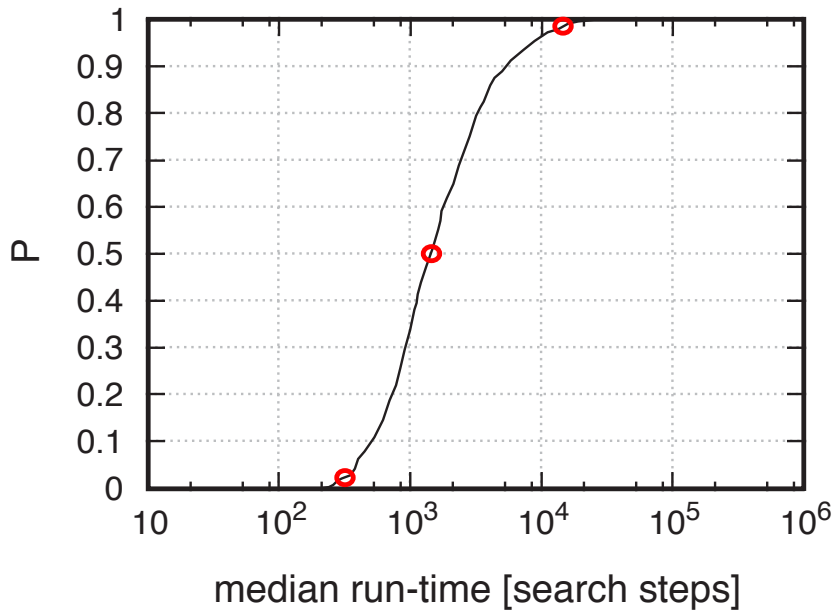
- In principle, the same approach as for individual instances is applicable: Measure empirical RTD for each instance, analyse using RTD plots or descriptive statistics.
- In many cases, the RTDs for set of instances have similar shapes or share important features (*e.g.*, being uni- or bi-modal, or having 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:



Distribution of median search cost for WalkSAT/SKC over set of 1000 randomly generated, 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*.

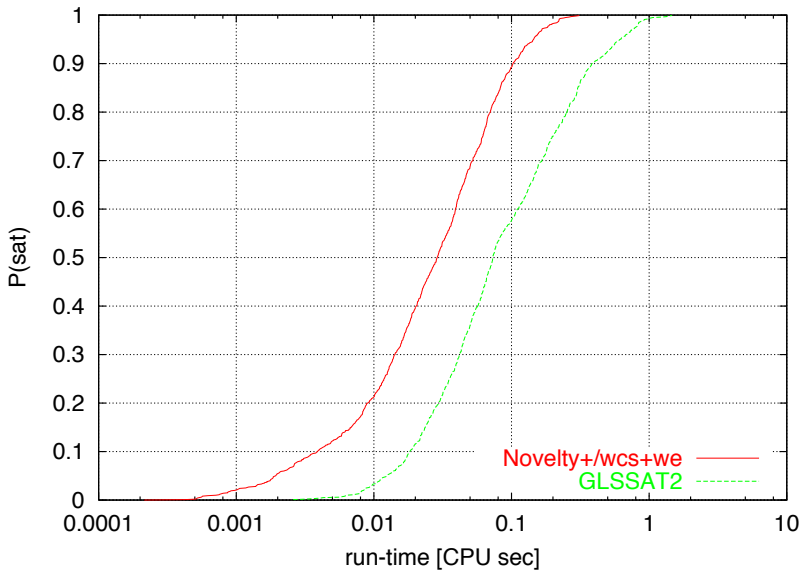
  $\rightsquigarrow$ Report and analyse run-time distributions on representative instance(s) as well as *search cost distribution (SCD)*, *i.e.*, distribution of basic RTD statistics (*e.g.*, median or mean) across given instance ensemble.

- 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.

Comparing algorithms based on RTDs (1)

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

Example of probabilistic domination

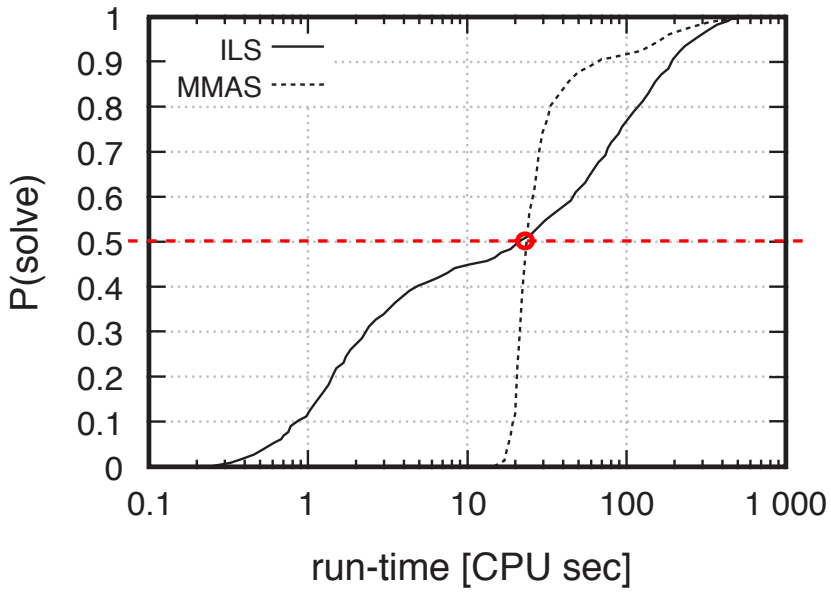


Comparing algorithms based on RTDs (2)

- A probabilistic domination relation holds between two SLS 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.

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



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 Wilcoxon rank-sum 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 is the probability of rejecting a false 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 how likely are the sampled data under the assumption that the null hypothesis is correct.

  The null hypothesis is rejected iff this p-value is smaller than the previously chosen significance level α.
- 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 *Wilcoxon rank sum test* (aka *Mann Whitney U-test*) is used to test whether the medians of two samples (*e.g.*, empirical RTDs) are significantly different.
- Unlike the t-test, the Wilcoxon 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 and SQDs of high performing 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*.

Comparative analysis for instance ensembles (1)

Goal: Compare performance of SLS 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)

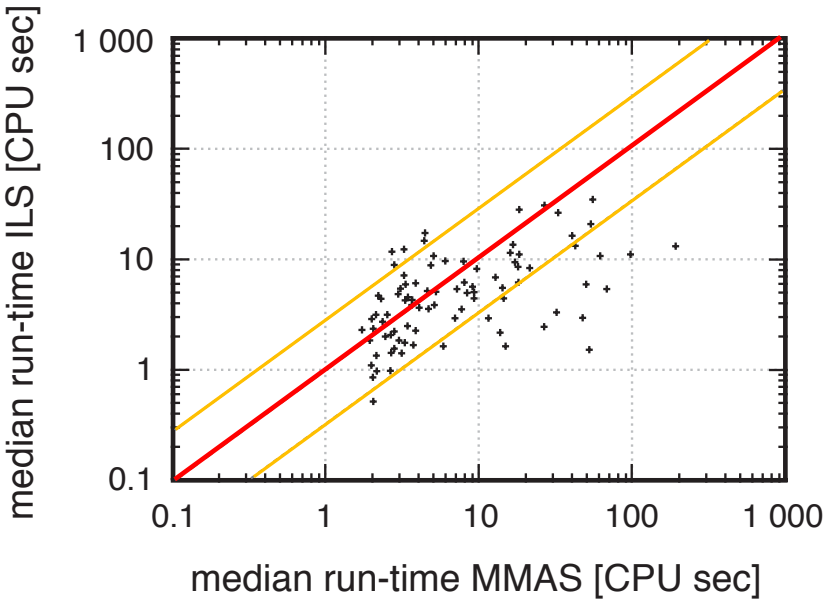
- Use statistical tests to assess significance of performance differences across given instance ensemble.
- The *binomial sign test* measures whether there are statistically significant deviations from the theoretically expected distribution of observations into two categories. (*e.g.*, percentage of instances on which median solution quality of an algorithm A is lower than that of algorithm B.
- 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.

  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:

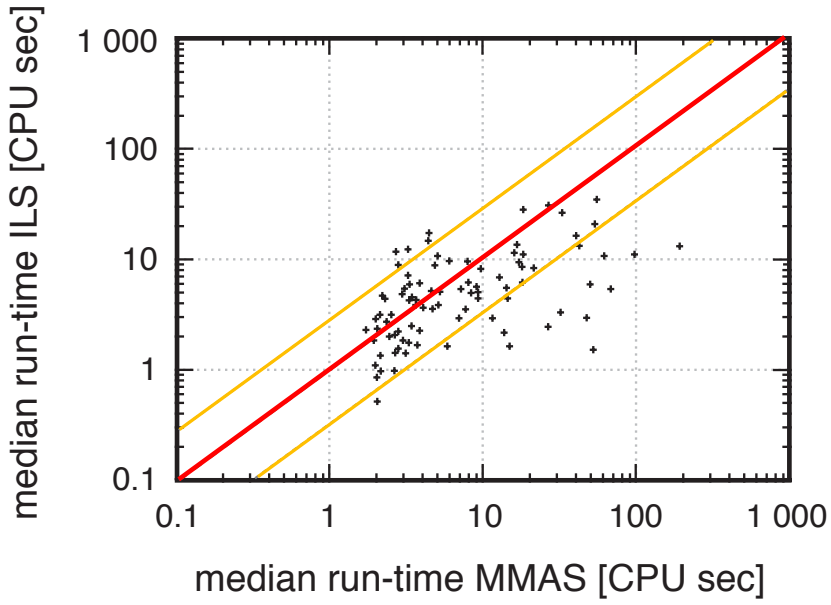


10 runs per instance.

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.
- 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.

  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.

  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 SLS algorithm 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 SLS performance to instance size).

Benchmark sets

Some criteria for constructing/selecting benchmark sets:

- instance hardness (focus on challenging 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, QAPLIB, 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!

Other issues

Environment specification

- specify the execution environment
  - machine, operating system, compiler, etc.
- specify implementation
  - report special details that impact performance (e.g. special data structures)
  - programming language, compiler
  - ideally, make source code available

More issues concerning SLS algorithms

- performance, avg. vs. peak vs. robustness
- correctness
- reproducibility (environment, implementation)
- ease of implementation
- configurability

Case study: SLS algorithms for QAP

- exemplify possible experimental comparison of SLS algorithms for the QAP
  - experiments on a single problem instance (not using RTDs, this time ..)
  - aggregation of results across instances
- SLS algorithms include ACO (2), ILS (2), TS (2), SA, EA
- Question: which metaheuristic for which problem instances
- algorithms tested on a total of
  - 34 instances from QAPLIB
  - 97 randomly generated instances

Acknowledgements: Thanks to Michael Sampels for the data and plots.

Example results: instance `tai50a`

- characteristics
  - unstructured, randomly generated instance
  - best known solution: 4 941 410
- all algorithms are given a same computation time limit, corresponding to $10\,000n$ iterations of a reference tabu search algorithm
- time limit 111.39 sec; 95% confidence interval is $[110.37, 112.41]$ secs
- each algorithm is run 25 times
- computational environment: AMD Athlon 1100 MHz CPU, 256 MB RAM, RedHat Linux 7.0, Kerlen 2.2.16-22, compiler gcc 2.95.3, -O3 flags

Best results of each trial, ordered ..

rank	name	value
1	aco.be	4961194
2	ils.be	4962298
3	ils.be	4963926
⋮	⋮	⋮
120	ils.d	5004444
121	ec.d	5005364
122	aco.ch	5007138
⋮	⋮	⋮
199	ts.nl	5109440
200	ts.nl	5117944

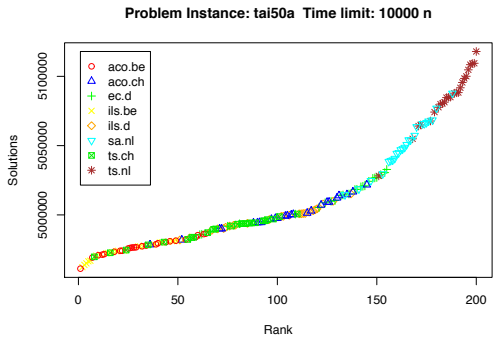


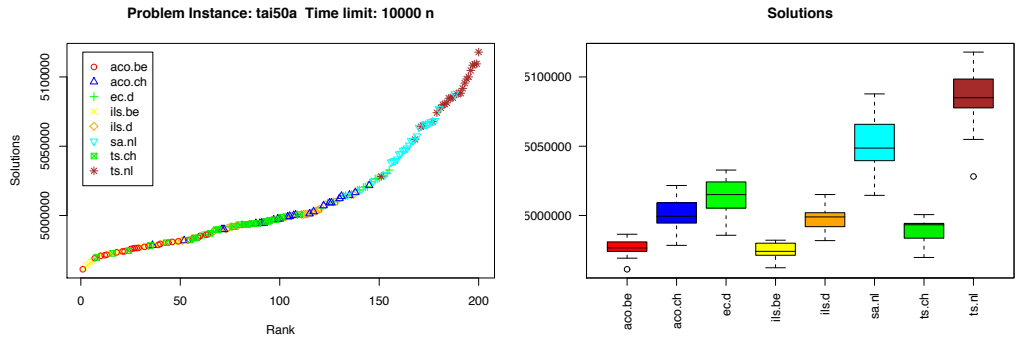
Figure: Results of 25 trials of the developed implementations of metaheuristics on tai50a

Compute mean values and sort list

name	mean value	sd value	mean rank	sd rank
ils.be	4974510	6012.32	25.72	16.84
aco.be	4977123	5836.67	32.92	18.36
ts.ch	4989029	8548.36	70.28	27.41
ils.d	4997612	7972.34	97.08	24.12
aco.ch	5000704	10537.01	103.84	26.66
ec.d	5012760	14523.08	124.72	30.65
sa.nl	5050280	17866.44	163.84	12.23
ts.nl	5085123	19689.38	185.6	11.48

Are these differences statistically significant?

Using boxplots for visualization



- median presented by a line
- box gives data from q_{25} to q_{75} quartiles
- whiskers go to extremes of the data
- outliers shown individually

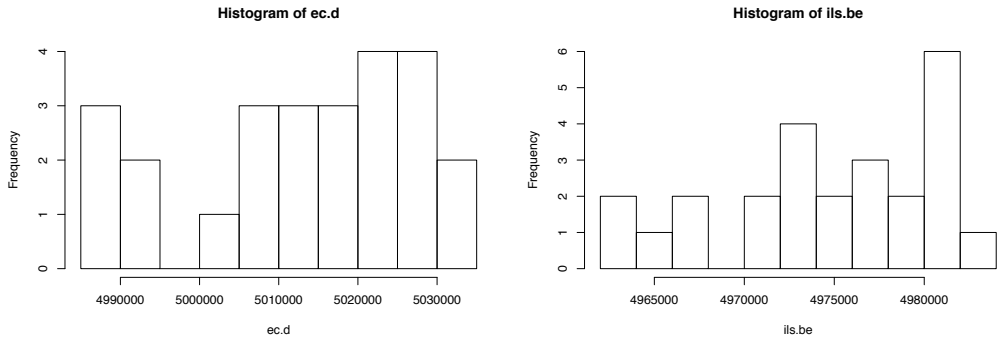
Pairwise Student's t-test

- H_0: two populations are identical
- adjustement for pairwise testing

	aco.be	aco.ch	ec.d	ils.be	ils.d	sa.nl	ts.ch
aco.ch	2.4e–09	-	-	-	-	-	-
ec.d	< 2e–16	0.00444	-	-	-	-	-
ils.be	0.76037	4.0e–11	< 2e–16	-	-	-	-
ils.d	2.1e–07	0.76037	0.00021	4.6e–09	-	-	-
sa.nl	< 2e–16	< 2e–16	< 2e–16	< 2e–16	< 2e–16	-	-
ts.ch	0.00444	0.00444	2.0e–09	0.00038	0.04658	< 2e–16	-
ts.nl	< 2e–16	< 2e–16	< 2e–16	< 2e–16	< 2e–16	< 2e–16	< 2e-16

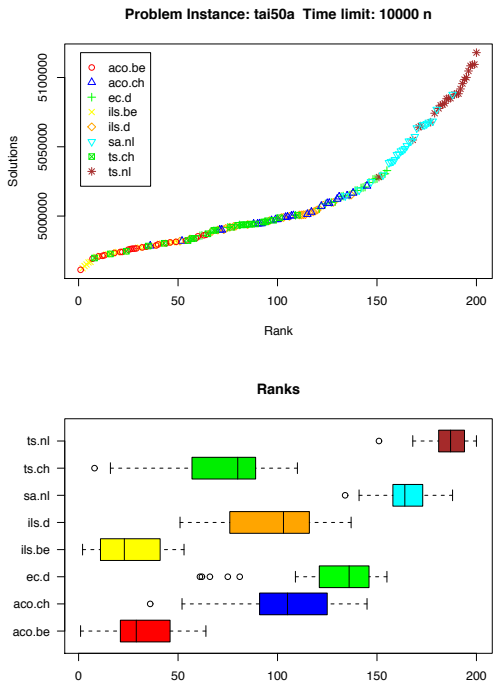
Student's t-test is valid if normal distribution can be assumed for populations

Check of distribution assumption



- distribution of data points is clearly not normal
- although t-test robust to different distributions, it seems not really be justified
- more appropriate are non-parametric tests

Ranking of data



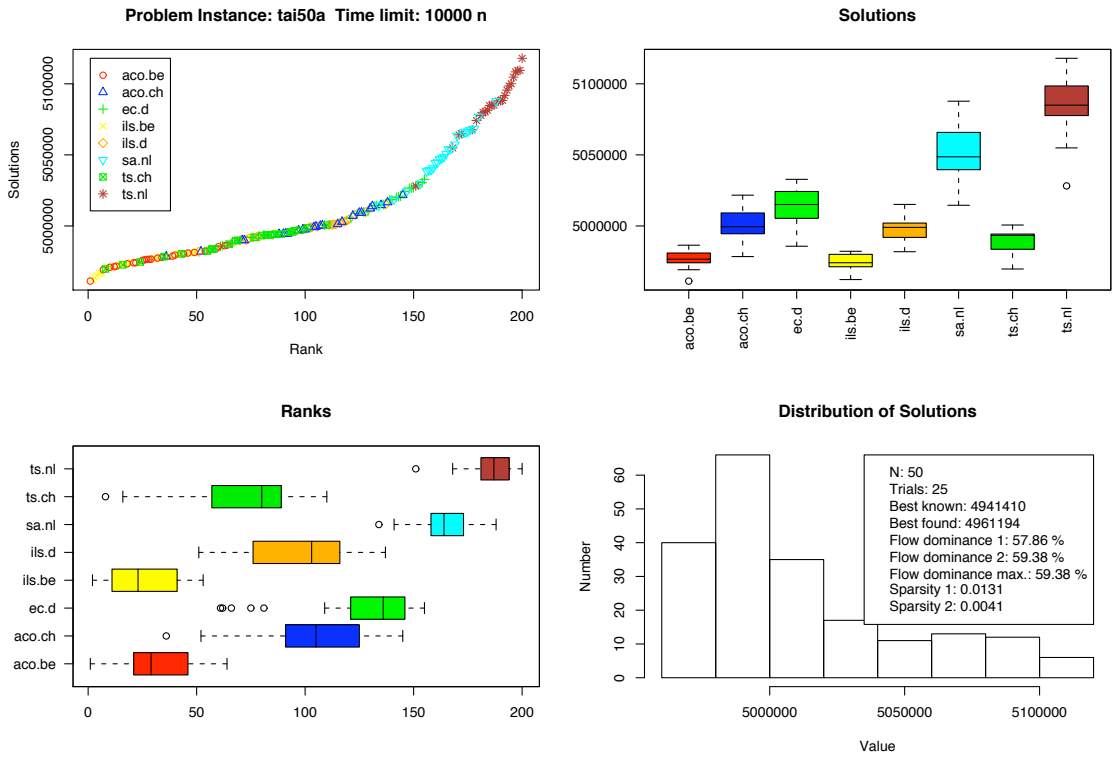
- adequate, non-parametric tests rely on the ranks of the results

Wilcoxon test

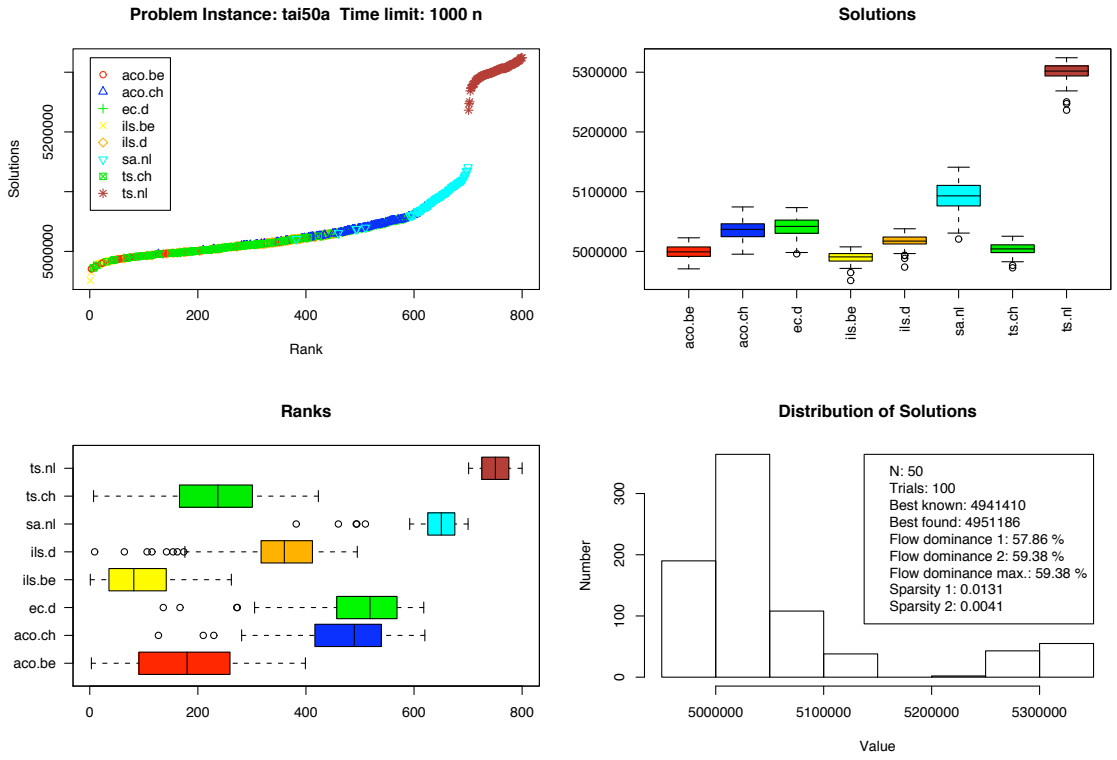
- H_0: two populations are identical
- adjustement for pairwise testing

	aco.be	aco.ch	ec.d	ils.be	ils.d	sa.nl	ts.ch
aco.ch	1.3e–10	-	-	-	-	-	-
ec.d	2.9e–12	0.00741	-	-	-	-	-
ils.be	0.39082	2.9e–11	3.8e–13	-	-	-	-
ils.d	2.1e–11	0.39082	0.00093	5.9e–13	-	-	-
sa.nl	3.8e–13	2.9e–12	6.2e–09	3.8e–13	5.9e–13	-	-
ts.ch	3.4e–05	0.00015	8.0e–06	1.1e–06	0.00600	3.8e–13	-
ts.nl	3.8e–13	3.8e–13	1.9e–12	3.8e–13	3.8e–13	6.0e–07	3.8e–13

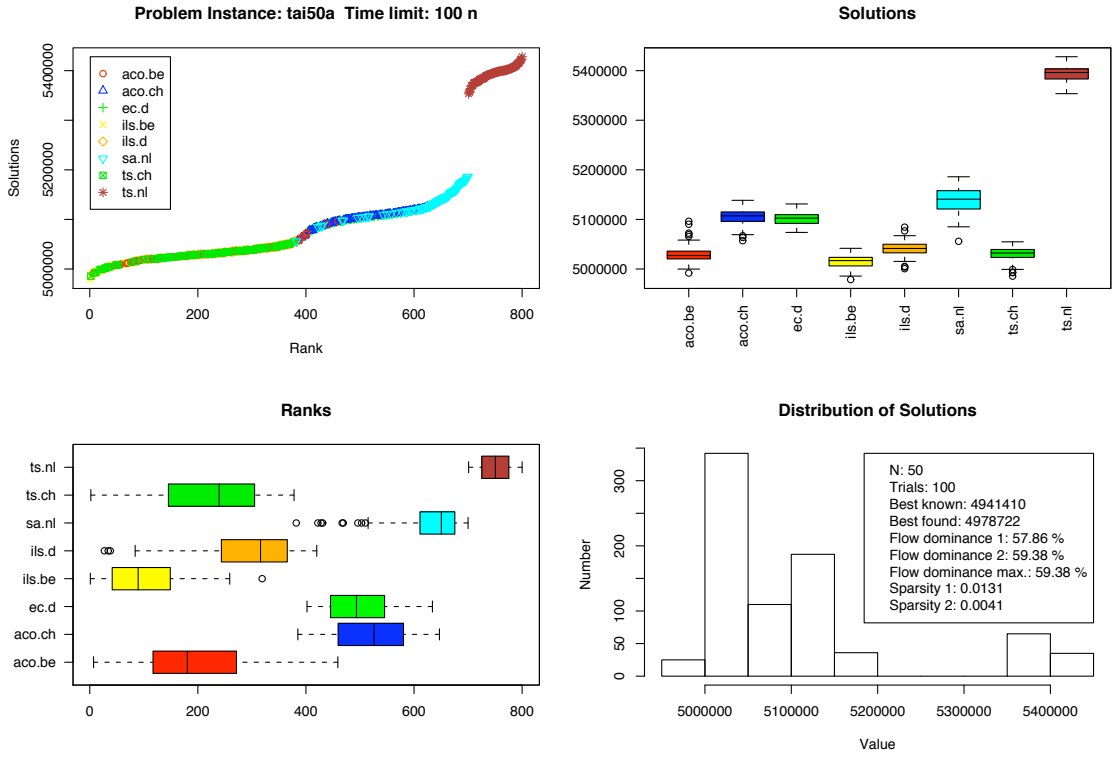
Plots summarizing the experimental data



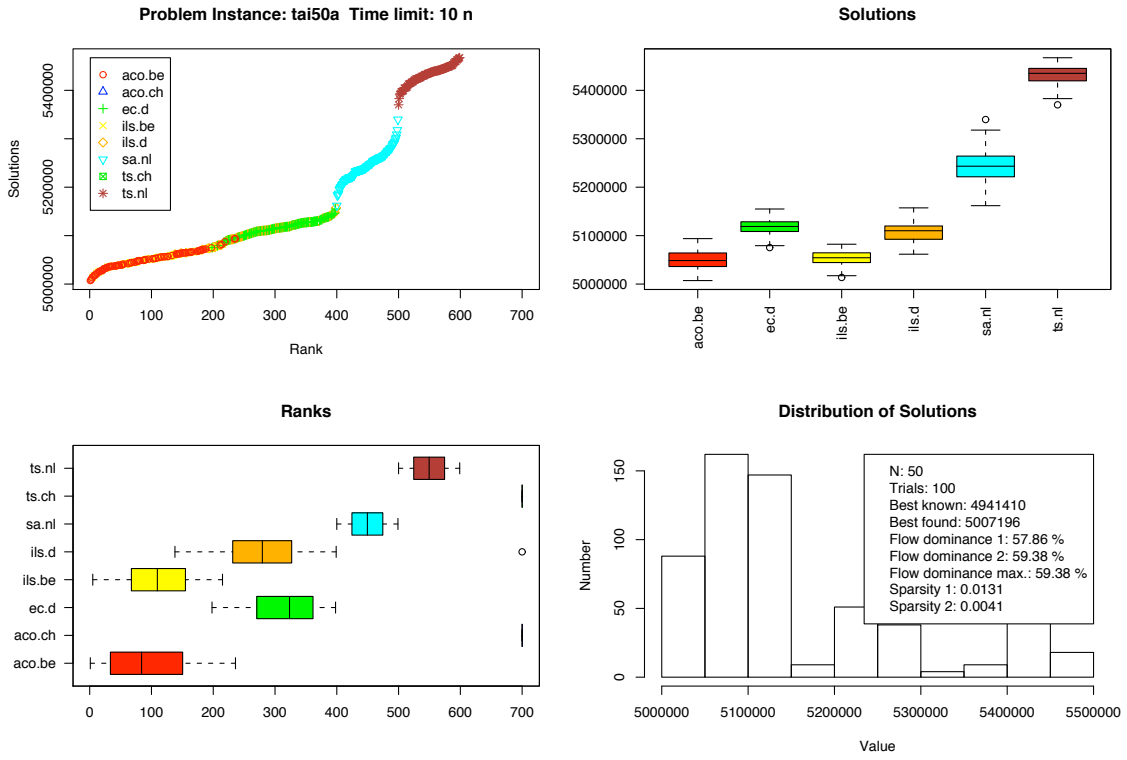
Plots summarizing the experimental data (1000n)



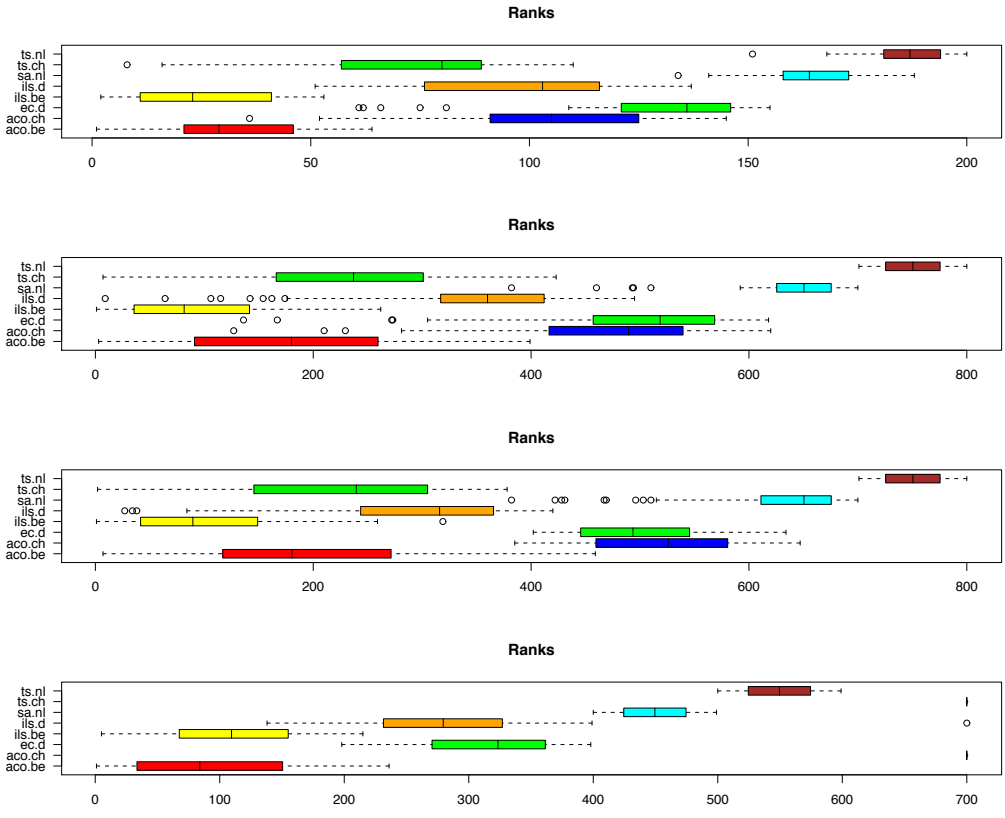
Plots summarizing the experimental data (100n)



Plots summarizing the experimental data (10n)



Effects of various time limits



Final result for `tai50a`

- Ranking is from left to right
- $10000n$
  aco.be $\sim$ ils.be ts.ch aco.ch $\sim$ ils.d ea.d sa.nl ts.nl
- $1000n$
  ils.be aco.be ts.ch ils.d aco.ch ea.d sa.nl ts.nl
- $100n$
  ils.be aco.be ts.ch ils.d ea.d aco.ch sa.nl ts.nl
- $10n$
  aco.be ils.be ils.d aco.ch ea.d sa.nl ts.nl aco.ch $\sim$ ts.ch

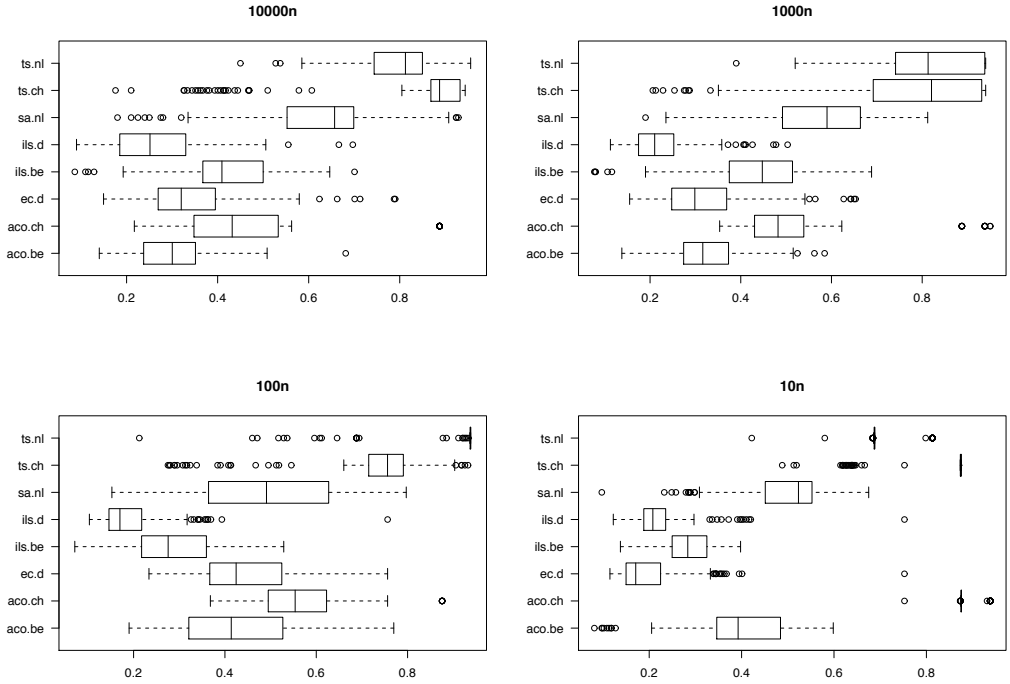
General result for QAP

- How to calculate the general result
  - one or two implementations for a metaheuristic
  - combine the results?
- approach: compute normalized mean rank for each ALG for each instance *inst* for each time limit t

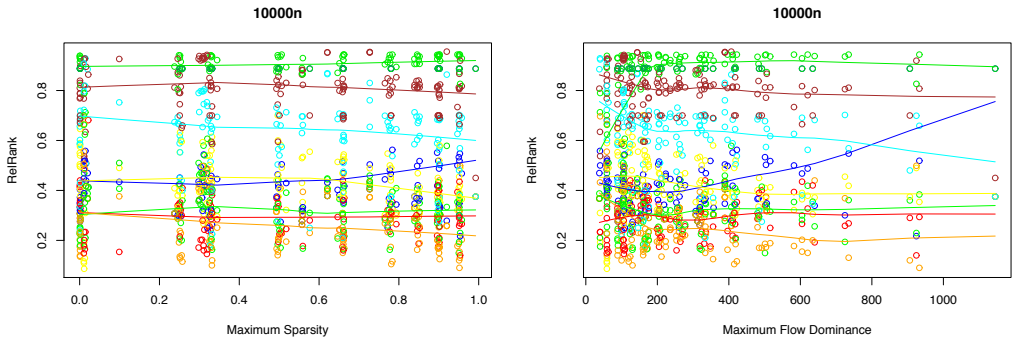
$$m(ALG, inst, t) = \frac{avg\_rank(ALG, inst.t)}{\#experiments(\text{inst,t})} \quad \in (0, 1)$$

- summarize results across instances; compare algorithms for each time limit

General result for QAP

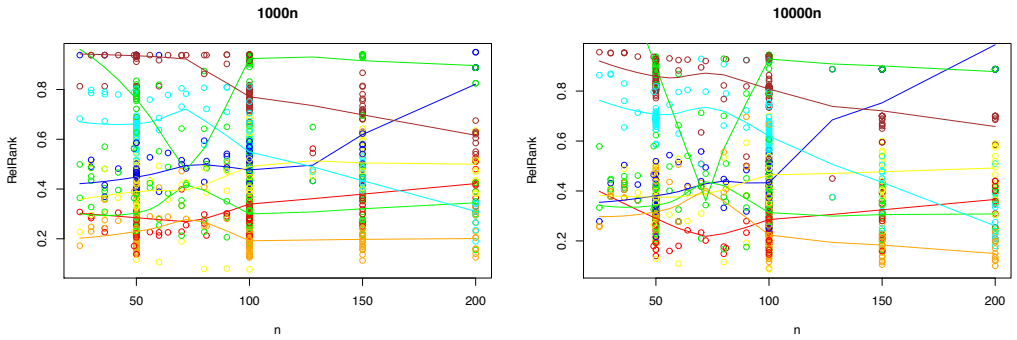


Correlation with sparsity, flow dominance



- for increasing sparsity and flow dominance
  - relative performance of aco.ch decreases
  - relative performance of sa.nl increases

Correlation with instance size



- for increasing instance size
  - relative performance of aco.ch decreases
  - relative performance of sa.nl increases
  - ts.ch good for $n \in [50, 100]$