4

Empirical Analysis of SLS Algorithms

In this chapter, we discuss methods for empirically analysing the performance and behaviour of stochastic local search algorithms. Most of our general considerations and all empirical methods covered in this chapter apply to the broader class of Las Vegas algorithms, which contains SLS algorithms as a subclass. After motivating the need for a more adequate empirical methodology and providing some general background on Las Vegas algorithms, we introduce the concept of run-time distributions (RTDs), which forms the basis of the empirical methodology presented in the following. Generally, this RTD-based analysis technique facilitates the evaluation, comparison, and improvement of SLS algorithms for decision and optimisation problems; specifically, it can be used for obtaining optimal parameterisations and parallelisations.

4.1 Las Vegas Algorithms

Stochastic Local Search algorithms are typically incomplete - when applied to a given instance of a combinatorial decision or optimisation problem, there is no guarantee that an (optimal) solution will eventually be found. However, in the case of a decision problem, if a solution is returned, it is

guaranteed to be correct. The same holds for the search and decision variants of optimisation problems. Another important property of SLS algorithms is the fact that, given a problem instance, the time required for finding a solution (in case a solution is found eventually) is a random variable. These two properties, correctness of the solution computed, and run-times characterised by a random variable, define the class of *Las Vegas Algorithms (LVAs)*.

Definition 4.1 (Las Vegas Algorithm)

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

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

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

Remark: According to this definition any deterministic search algorithm would also be a Las Vegas algorithm; however, the term is typically used in the context of randomised algorithms.

Obviously, any SLS algorithm for a decision problem is a Las Vegas algorithm, as long as the validity of any solution returned by the algorithm is checked. Typically, checking for the correctness of solutions is very efficient compared to the overall run-time of an SLS algorithm, and most SLS algorithms implement such a check before returning any result. (Note that for problems in $\mathcal{NP}$, the correctness of a solution can always be verified in polynomial time.) Based on this argument, in the following we assume that SLS algorithms for decision problems always check correctness before returning a solution.

As an example, it is easy to see that Uniform Random Picking (as introduced in Section 1.5) is a Las Vegas algorithm: Since generally, a solution is never returned without verifying it first (as explained above), condition (1) of the definition is trivially satisfied, and because of the randomised selection process in each search step, the time required for finding a solution is obviously a random variable.

In the case of SLS algorithms for optimisation problems, at the first glance, the situation seems to be less clear. Intuitively and practically, unless the optimal value of the objective function is known, it is typically impossible to efficiently verify the optimality of a given candidate solution. However, as noted in Section 1.1, many optimisation problems include logical conditions that restrict the set of valid solutions. The validity of a solution can be checked efficiently for combinatorial optimisation problems whose associated decision problems are in $\mathcal{NP}$, and SLS algorithms for solving such optimisation problems generally include such a test before returning a solution. Hence, if only valid solutions are considered correct, SLS algorithms for optimisation problems fit the formal definition of Las Vegas algorithms.

However, SLS algorithms for optimisation problems typically have the additional property that for fixed run-time, the solution quality, *i.e.*, the value of the objective function for the current solution candidate, is also a random variable.

Definition 4.2 (Optimisation Las Vegas Algorithm)

An algorithm A for a class Π' of optimisation problems is an *Optimisation Las Vegas Algorithm (OLVA)* if it is a Las Vegas algorithm, and for each problem instance $\pi' \in \Pi'$ the solution quality after run-time t is a random variable $SQ(t)$. □

Note that for OLVAs, both the solution quality achieved within a bounded run-time is a random variable, but the same holds for the run-time required for achieving or exceeding a given solution quality. The latter follows from the fact that we view LVAs for optimisation problems as LVAs for solving the associated decision variants.

Las Vegas algorithms are prominent in various areas of Computer Science and Operations Research. A significant part of this impact is due to the successful application of SLS algorithms for solving $\mathcal{NP}$-hard combinatorial problems. However, there are other very successful Las Vegas algorithms that are not based on stochastic local search. In particular, a number of systematic search methods, including some fairly recent variants of the Davis Putnam algorithm for satisfiability (SAT) problems (see also

Chapter 6), make use of non-deterministic decisions, such as randomised tie-breaking rules, and fall into the category of Las Vegas Algorithms.

It should also be noted that Las Vegas algorithms can be seen as a special case of the larger, and also very prominent class of *Monte Carlo Algorithms*. Like LVAs, Monte Carlo algorithms are randomised algorithms with randomly distributed run-times. However, a Monte Carlo algorithm can sometimes return an incorrect answer; in other words, it can generate false positive results (incorrect solutions to the given problem) as well as false negative results (missed correct solutions), while for Las Vegas algorithms, only false negatives are allowed.

Empirical *vs* Theoretical Analysis

As a result of their inherently non-deterministic nature, the behaviour of Las Vegas algorithms is usually difficult to analyse. For most practically relevant LVAs, in particular SLS algorithms that perform well in practice, theoretical results are typically hard to obtain, and even in the cases where theoretical results do exist, their practical applicability is often very limited.

The latter situation can arise for different reasons. Firstly, sometimes the theoretical results are obtained under idealised assumptions, which do not hold in practical situations. This is, for example, the case for Simulated Annealing, which is proven to converge towards an optimal solution under certain conditions, one of which is infinitesimally slow cooling in the limit [Hajek, 1988]—which obviously cannot be achieved in practice.

Secondly, most complexity results apply to worst-case behaviour, and in those relatively few cases, were theoretical average-case results are available, these typically are based on instance distributions that are unlikely to be encountered in practice. Finally, theoretical bounds on the run-times of SLS algorithms are typically asymptotic, and do not reflect the actual behaviour accurately enough.

Given this situation, in most cases the analysis of the run-time behaviour of Las Vegas algorithms is based on empirical methodology. In a sense, despite dealing with algorithms which are completely known and easily understood on a step-by-step execution basis, computer scientists are in a sense in the same situation as, for instance, an experimental physicist studying some non-deterministic quantum phenomenon or a microbiologist investigating bacterial growth behaviour. In either case, a complex phenomenon

of interest cannot be easily derived from known underlying principles solely based on theoretical means; instead, the classical scientific cycle of observation, hypothesis, prediction, experiment is employed in order to obtain a model that explains the phenomenon. It should be noted that in all empirical sciences, in particular physics, chemistry, and biology, it is largely a collection of these models which constitutes theoretical frameworks, whereas in computer science, theory is almost exclusively derived from mathematical foundations.

Historical reasons aside, this difference is certainly largely due to the fact that algorithms are completely specified and mathematically defined at the lowest level. However, in the case of SLS algorithms (and many other complex algorithms or systems), this knowledge is often not sufficient to theoretically derive all relevant aspects of their behaviour. In this situation, empirical approaches, based on computational experiments, are often not only the sole way of assessing a given algorithm, but also have the potential to provide insights into practically relevant aspects of algorithmic behaviour that appear well beyond the reach of theoretical analysis.

Norms of LVA Behaviour

By definition, Las Vegas algorithms are always correct, while they are not necessarily complete, *i.e.*, even if a given problem instance has a solution, a Las Vegas algorithm is generally not guaranteed to find it. Completeness is not only an important theoretical concept for the study of algorithms, but is often also very relevant in practical applications. In the following, we distinguish not only between complete and incomplete Las Vegas algorithms, but also introduce a third category, so-called *probabilistically approximately complete* LVAs. Intuitively, an LVA is complete, if it can be guaranteed to solve any soluble problem instance in bounded time; it is probabilistically approximately complete (PAC), if it will solve each soluble problem instance with arbitrarily high probability if it is allowed to run long enough; and it is essentially incomplete, if even arbitrarily long runs cannot be guaranteed to find existing solutions. These concepts can be formalised as follows:

Definition 4.3 (Asymptotic Behaviour of LVAs)

Consider a Las Vegas algorithm A for a problem class Π, and

let $P_s(RT_{A,\pi} \leq t)$ denote the probability that A finds a solution for a soluble instance $\pi \in \Pi$ in time less than or equal to t.

A is called

- *complete*, if and only if for each soluble instance $\pi \in \Pi$ there exists some t_{max} such that $P_s(RT_{A,\pi} \leq t_{max}) = 1$;
- *probabilistically approximately complete (PAC)*, if and only if for each soluble instance $\pi \in \Pi$, $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) = 1$;
- *essentially incomplete*, if it is not PAC, *i.e.* if there exists a soluble instance $\pi \in \Pi$, for which $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) < 1$.

Probabilistic approximate completeness is also refered to as the PAC property, and we will often use the term 'approximately complete' to characterise algorithms that are PAC.

Furthermore, we will use the terms completeness, probabilistic approximate completeness, and essential incompleteness also with respect to single problem instances or subsets of a problem class Π, if the respective properties hold for the corresponding sets of instances instead of Π. □

Examples for complete Las Vegas algorithms are randomised systematic search procedures such as Satz-Rand [Gomes *et al.*, 1998]. Many stochastic local search methods, such as Randomised Iterative Improvement and variants of Simulated Annealing , are PAC, while others, such as basic Iterative Improvement, most variants of Iterated Local Search, and most tabu search algorithms are essentially incomplete (some specific results can be found in subsequent chapters).

Theoretical completeness can be achieved for any SLS algorithm by using a restart mechanism that systematically re-initialises the search such that eventually, the entire search space has been visited. However, the time limits for which solutions are guaranteed to be found using this appproach are typically far too large to be of practical value. A similar situation arises in many practical situations for search algorithms whose completeness is achieved by different means.

Probabilistic approximate completeness is often also referred to as *convergence* in the literature; this property is established for a number of SLS algorithms, such as Simulated Annealing [Kirkpatrick *et al.*, 1983], or Genetic Algorithms [Holland, 1975; Goldberg, 1989]. Another well-known notion of convergence characterises SLS algorithms whose search trajectory ends in a (optimal) solution with probability arbitrarily close to one as run-time approaches infinity. One of the best-known convergence results of this type has been proven for Simulated Annealing [Hajek, 1988]. This notion of convergence implies the PAC property (but not vice versa); for practical purposes, however, it seems to offer no advantage over the PAC property, since for decision problems, the search is typically terminated as soon as a solution is found, and for optimisation problems, the best quality solution encountered so far is memorised and can be accessed at any time throughout the search. In fact, it can be argued that in practice, guaranteed convergence of the search trajectory towards (optimal) solutions often carries the risk of search stagnation due to a lack of diversification in earlier phases of the search process.

Essential incompleteness of an SLS algorithm is usually caused by the algorithm's inability to escape from attractive local minima regions of the search space. Any mechanism that guarantees that a search process can eventually escape from arbitrary regions of the search space, given enough time, can make an SLS algorithm probabilistically approximately complete. Examples for such mechanisms include random restart, random walk, and probabilistic tabu-lists; however, as we will discuss in more detail later (see Section 4.4), not all such mechanisms necessarily lead to performance improvements relevant to practical applications.

For optimisation LVAs, the concepts of completeness, probabilistic approximate completeness, and essential incompleteness can applied to the associated decision problems in a straight-forward way, using the following generalisations:

Definition 4.4 (Asymptotic Behaviour of OLVAs)

Consider an optimisation Las Vegas algorithm A for a problem class Π', and let $P_s(RT_{A,\pi'} \leq t, SQ_{A,\pi'} \leq q')$ denote the probability that A finds a solution of quality $\leq q'$ (with $q' \geq f_{opt}$, where f_{opt} is the optimal solution quality), for a soluble instance $\pi' \in \Pi'$ in time $\leq t$.

A is called

- q'*-complete*, if and only if for each soluble instance $\pi' \in \Pi'$ there exists some t_{max} such that $P_s(RT_{A,\pi'} \leq t_{max}, SQ_{A,\pi'} \leq q') = 1$;
- *probabilistically approximately* q'*-complete (*q'*-PAC)*, if and only if for each soluble instance $\pi' \in \Pi'$, $\lim_{t\to\infty} P_s(RT_{A,\pi'} \leq t, SQ_{A,\pi'} \leq q') = 1$;
- *essentially* q'*-incomplete*, if it is not approximately q'-complete, *i.e.*, if there exists a soluble problem instance $\pi' \in \Pi'$, for which $\lim_{t\to\infty} P_s(RT_{A,\pi'} \leq t, SQ_{A,\pi'} \leq q') < 1$. □

With respect to finding optimal solutions, we use the terms *complete*, *approximately complete*, and *essentially incomplete* synonymously for q'*-complete*, *approximately* q'*-complete*, and *essentially* q'*-incomplete*, where q' is the optimal solution quality for the given problem instance.

Application Scenarios and Evaluation Criteria

For the empirical analysis of any algorithm it is crucial to use appropriate evaluation criteria. In the case of Las Vegas algorithms, depending on the characteristics of the application context, different evaluation criteria are appropriate. Let us start by considering Las Vegas algorithms for decision problems, and classify possible application scenarios in the following way:

Type 1: There are no time limits, *i.e.*, we can afford to run the algorithm as long as it needs to find a solution. Basically, this scenario is given whenever the computations are done offline or in a non-realtime environment where it does not really matter how it takes to find a solution. In this situation we are interested in the expected time required for finding a solution; this can be estimated from easily from a number of test runs.

Type 2: There is a hard time limit for finding the solution such that the algorithm has to provide a solution after a time of t_{max}; solutions that are found later are of no use. In real-time applications, such as robotic control or dynamic task scheduling, t_{max} can be very small. In this situation we are not so much interested in the expected time for finding a solution but in the probability that after the hard deadline t_{max} a solution has been found.

Type 3: The usefulness or utility of a solution depends on the time that was needed to find it. Formally, if utilities are represented as values in $[0, 1]$, we can characterise these scenarios by specifying a utility function $U : \mathbb{R}^+ \mapsto [0, 1]$, where $U(t)$ is the utility of finding a solution at time t. As can be easily seen, application types 1 and 2 are special cases of type 3 which can be characterised by utility functions that are either constant (type 1) or step functions $U(t) = 1$ for $t \leq t_{max}$ and $U(t) = 0$ for $t > t_{max}$ (type 2). However, as we will explain in more detail shortly, more complex utility functions are encountered in many real-world applications.

While in the case of no time limits being given (type 1), the mean run-time of a Las Vegas algorithm might suffice to roughly characterise its run-time behaviour, in real-time situations (type 2) it is basically meaningless. Type 3 is not only the most general class of application scenarios, but these scenarios are also the most realistic. The reason for this is the fact that real-world problem solving usually involves time-constraints which are less strict than the hard deadline given in type 2 scenarios. Instead, at least within a certain interval, the value of a solution gradually decreases over time. In particular, this situation is given when taking into account the costs (like CPU time) of finding a solution.

As an example, consider a situation where hard combinatorial problems have to be solved online, using expensive hardware in a time-sharing mode. Even if the immediate benefit of finding a solution is invariant over time, the costs for performing the computations will diminish the final payoff. Two common ways of modelling this effect are constant or proportional discounting, *i.e.*, to use utility functions of the form $U(t) = \max\{u_0 - ct, 0\}$ or $U(t) = e^{-\lambda t}$, respectively [Poole *et al.*, 1998]. Based on the utility function, the weighted solution probability $U(t) \cdot P(RT \leq t)$ can be used as a performance criterion. If $U(t)$ and $P_s(RT \leq t)$ are known, optimal cutoff times t^* that maximise the weighted solution probability can be determined as well as the expected utility for a given time t'. These evaluations and calculations require detailed knowledge of the solution probabilities $P_s(RT \leq t)$, potentially for arbitrary run-times t.

In the case of optimisation Las Vegas algorithms, solution quality has to be considered as an additional factor. One might imagine application contexts in which the run-time is basically unconstrained, such as in the type 1 scenarios discussed above, but a certain solution quality needs to be obtained, or situations in which a hard time-limit is given, during which

the best possible solution is to be found. Typically, however, one can expect to find more complex tradeoffs between run-time and solution quality. Therefore, the most realistic application scenario for optimisation Las Vegas algorithms is a generalisation of type 3, where the utility of a solution depends on its quality as well as on the time needed to find it. This is modelled by utility functions $U(t,q) : \mathbb{R}^+ \times \mathbb{R}^+ \mapsto [0,1]$, where $U(t,q)$ is the utility of a solution of quality q found at time t. Analogous to the case of decision LVAs, the probability $P_s(RT \leq t, SQ \leq q)$ for obtaining a certain solution quality q within a given time t, weigthed by the utility $U(t,q)$ can be used a performance criterion.
[hh: add some figures illustrating utility functions, run-time distributions, and utility–weighted solution probability?]

4.2 Run-time Distributions

As we have argued in the previous section, it is generally not sufficient to evaluate LVAs based on the expected time for solving a problem instance or achieving a given solution quality, or the probability of solving a given instance within a given time. Instead, application scenarios are often characterised by complex utility functions, or SLS algorithms or other LVAs are evaluated without a priori knowledge of the application scenario, such that a utility function is unknown but cannot be assumed to correspond to one of the special cases characterising type 1 or 2 application scenarios. Therefore, LVA evaluations should be based on a detailed knowledge and analysis of the solution probabilities $P_s(RT \leq t)$ for decision problems, and $P_s(RT \leq t, SQ \leq q)$ for optimisation problems, respectively. Obviously, these probabilities can be determined from the probability distributions of the random variables characterising the run-time and solution quality of a given LVA.

Definition 4.5 (Run-time Distribution)

Consider a Las Vegas algorithm A for a class Π of decision problems and let $P_s(RT_{A,\pi} \leq t)$ denote 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 $RT_{A,\pi}$, which is characterised by

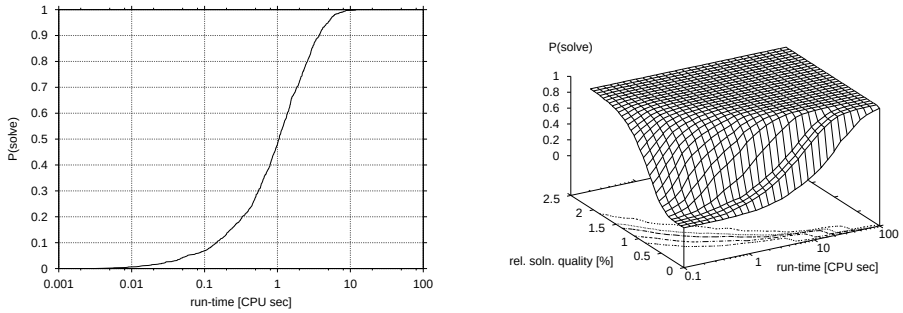


Figure 4.1: Typical run-time distributions for SLS algorithms applied to hard combinatorial decision (left) and optimisation problems (right); for details, see text.

the *run-time distribution function* $rtd : \mathbb{R} \mapsto [0,1]$ defined as $rtd(t) = P_s(RT_{A,\pi} \leq t)$.

Similarly, given an optimisation Las Vegas algorithm A' for a class Π' of optimisation problems, and a soluble problem instance $\pi' \in \Pi'$, let $P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$ denote the probability that A' applied to π' finds a solution 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 $(RT_{A',\pi'}, SQ_{A',\pi'})$, which is characterised by the *run-time distribution function* $rtd : \mathbb{R} \times \mathbb{R}^+ \mapsto [0,1]$ defined as $rtd(t,q) = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$. □

Remark: Since RTDs are completely and uniquely characterised by their distribution functions, we will often use the term 'run-time distribution' or 'RTD' to refer to the corresponding run-time distribution functions.

Example 4.1: RTDs for decision and optimisation LVAs

Figure 4.1 (left) shows the typical run-time distribution for a SLS algorithm

applied to an instance of a hard combinatorial decision problem. The RTD is represented by a cumulative probability distribution curve $(t, \widehat{P}_s(RT \leq t))$ which has been empirically determined from 1,000 runs of WalkSAT, one of the most prominent SLS algorithms for SAT, on a hard Random-3-SAT instance with 100 variables and 430 clauses (for details on the algorithm and problem class, see Chapter 6).

Figure 4.1 (right) shows the bivariate RTD for an SLS optimisation algorithm applied to an instance of a hard combinatorial optimization problem. The plotted surface corresponds to the cumulative probability distribution of an empirically measured RTD, in this case determined from 1,000 runs of an Iterated Local Search (ILS) algorithm applied to instance `pcb442` with 442 cities from TSPLIB, a benchmark library for the TSP (details on SLS algorithms and benchmark problems for the TSP will be discussed in Chapter 8). Note how the contours of the three dimensional qualified RLD surface projected into the run-time / solution quality plane reflect the trade-off between run-time and solution quality: for a given probability level, better solution qualities require longer runs, while vice versa, shorter runs yield lower quality solutions.

The behaviour of a Las Vegas algorithm applied to a given problem instance is completely and uniquely characterised by the corresponding RTD. Given an RTD, other performance measures or evaluation criteria can be easily computed. For decision LVAs, measures such as the mean run-time for finding a solution, its standard deviation, median, percentiles , or success-probabilities for arbitrary time limits, are often used in empirical studies. For optimisation LVAs, popular evaluation criteria include the mean or standard deviation of the solution quality for a given run-time (cutoff time) as well as basic descriptive statistics of the run-time required for obtaining a given solution quality.

Different from these measures, however, knowledge of the RTD allows the evaluation of Las Vegas algorithms for problems and application scenarios which involve more complex trade-offs. Some of these can be directly represented by a utility function, while others might concern preferences on properties of the RTDs. As an example for the latter, consider a situation where for a given time-limit t', one SLS algorithm gives a high mean

solution quality but a relatively large standard deviation, while another algorithm produces slightly inferior solutions in a more consistent way. RTDs provide a basis for addressing such trade-offs quantitatively and in detail.

Qualified RTDs and SQDs

Multivariate probability distributions, such as the RTDs for optimisation LVAs, are often more difficult to handle than univariate distributions. Therefore, when analysing and characterising the behaviour of optimisation LVAs, instead of working directly with bivariate RTDs, it is often easier and more appropriate, to focus on the (univariate) distributions of the run-time required for reaching a given solution quality threshold.

Definition 4.6 (Qualified Run-time Distribution)

Let A' be an optimisation Las Vegas algorithm for a class Π' of optimisation problems, and let $\pi' \in \Pi'$ be a soluble problem instance. If $rtd(t, q)$ is the RTD of A' on π', then for any solution quality q', the *qualified run-time distribution of A' on π' for q'* is defined by the distribution function $qrtd_{q'}(t) = rtd(t, q') = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q')$. □

Obviously, the qualified RTDs thus defined are marginal distributions of the bivariate RTD; intuitively, they correspond to cross-sections of the two-dimensional RTD graph for fixed solution quality values. Qualified RTDs are useful for characterising the ability of a SLS algorithm for an optimisation problem to solve the associated decision problems (*cf.* Chapter 1). In practice, they are commonly used for studying an algorithm's ability to find optimal or close-to-optimal solutions (if the optimal solution quality is known), or feasible solutions (in cases where hard constraints are given). Analysing series of qualified RTDs with solution quality thresholds which are increasingly more restrictive can give a detailed picture of the behaviour of an optimisation LVA.

An important question arises with respect to the solution quality bounds used when measuring or analysing qualified RTDs. For some problems, benchmark instances with known optimal solutions are available. In this

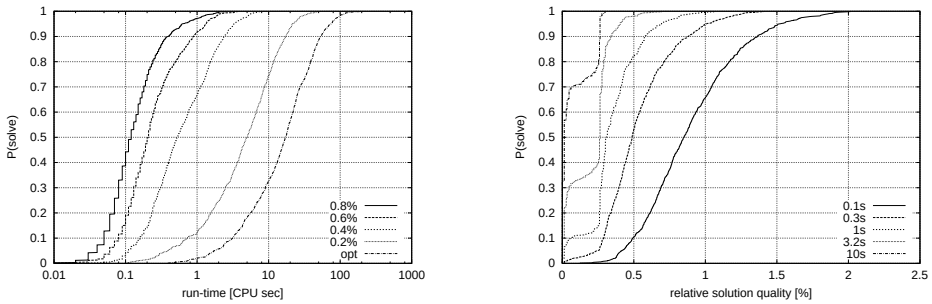


Figure 4.2: *Left:* qualified RTDs for the bivariate RTD from Figure 4.1; *right:* SQDs for the same RTD.

case, bounds expressed as relative deviations from the optimal solution quality are often used; the relative deviation of solution quality q from optimal solution quality f_{opt} is calculated as $q/f_{opt} - 1$ and is often expressed in percent. (In cases where $f_{opt} = 0$, sometimes the solution quality is normalised by dividing it by the maximal (*i.e.*, worst) possible solution quality.) If optimal solutions are not known, one possibility is to evaluate the SLS algorithms w.r.t. the best known solutions. This method, however, has the potential disadvantage that best known solutions can become obsolete. Therefore, it is sometimes preferable to use lower bounds of the optimal solution quality, especially if these are known to be close to the optimum, as is the case for the TSP [Held and Karp, 1970; Johnson and McGeoch, 1997]. Alternately, there are statistical methods for estimating optimal solution qualities in cases where tight lower bounds are not available [**?**; **?**].

Example 4.2: Qualified run-time distributions

Figure 4.2 (left) shows a set of qualified RTDs which correspond to marginal distributions of the bivariate empirical RTD from Example **??**. Note that when tightening the solution quality bound, the qualified RTDs get shifted to the right and appear somewhat steeper in the semilog plot. This indicates that not only the run-time required for finding higher quality solutions is higher, but also the relative variability of the run-time (as reflected, *e.g.*, in

the variation coefficient, *i.e.*, the standard deviation of the RTD devided by its mean). The latter observation reflects a rather typical property of SLS algorithms for hard optimisation problems.

An orthogonal view of an optimisation LVAs behaviour is given by the distribution of the solution quality for fixed run-time limits.

Definition 4.7 (Solution Quality Distribution)

Let A be an optimisation Las Vegas algorithm A for a class Π' of optimisation problems, and let $\pi' \in \Pi'$ be a soluble problem instance. If $rtd(t,q)$ is the RTD of A on π', then for any run-time t', the *solution quality distribution (SQD) of A on π' for t'* is defined by the distribution function $sqd_{t'}(q) = rtd(t',q) = P_s(RT_{A',\pi'} \leq t', SQ_{A',\pi'} \leq q)$. □

Like qualified RTDs, solution quality distributions are marginal distributions of a bivariate RTD. Intuitively, they correspond to cross-sections of the two-dimensional RTD graph for fixed run-times; in this sense they are orthogonal to qualified RTDs. SQDs are particularly useful in situations where fixed cutoff times are given (such as in type 2 application scenarios). Furthermore, they facilitate quantitative and detailed analyses of the trade-offs between the chance of finding a good solution fast and the risk of obtaining only low-quality solutions.

Different from run-time, solution quality is inherently bounded from below by the quality of the optimal solution of the given problem instance. This constrains the SQDs of typical SLS algorithms, such that for sufficiently long run-times, an increase in mean solution quality is often accompanied by a decrease solution quality variability. In particular, for an probabilistically approximately complete algorithm, the SQDs for increasingly large time-limits t' approach a degenerate probability distribution that has all probability mass concentrated on the optimal solution quality.

Example 4.3: Solution quality distributions

Figure 4.2 (right) shows a set of SQDs, marginal distributions of the bivariate empirical RTD from Example **??**, which offer an orthogonal view to the qualified RTDs from Example 4.2. The SQDs show clearly that for increasing run-time, the entire probability mass is shifted towards higher-quality solutions, while the variability in solution quality decreases. It is also interesting to note that the SQDs for large run-times are multimodal, as can be seen from the fact that they have multiple steep segments which corrrespond to the peaks in probability density (modes).

Time-dependent Summary Statistics

Instead of dealing with a set of SQDs for a series of time-limits, researchers (and practitioners) often just look at the development of certain solution quality statistics over time (SQTs). A common example of such an SQT is the function $\overline{SQ}(t)$, which characterises the time-dependent development of the mean solution quality achieved by a given algorithm. We generally prefer SQTs reflecting the development of percentiles (*e.g.*, median) of the SQDs over time, since percentiles are typically statistically more stable than means. Furthermore, SQTs based on SQD percentiles offer the advantage that they can be seen as horizontal sections or contour lines of the underlying bivariate RTD surfaces. Combinations of such SQTs can be very useful for summarising certain aspects of a full SQD series, and hence a complete bivariate RTD; they are particularly suited for explicitly illustrating trade-offs between run-time and solution quality. Especially individual SQTs, however, offer a fairly limited view of an optimisation Las Vegas algorithm's run-time behaviour in which important details can be easily missed.

Example 4.4: Solution quality statistics over time

Figure 4.3 (left) shows the development of solution quality and standard deviation over time, obtained from the same empirical data underlying the bivariate RTD from Example **??**. From this type of evaluation, which is often used in the literature, we can easily see that in the given example the algorithm behaves in a very desirable way: with increasing run-time, the median solution quality as well as the higher SQD percentiles improve

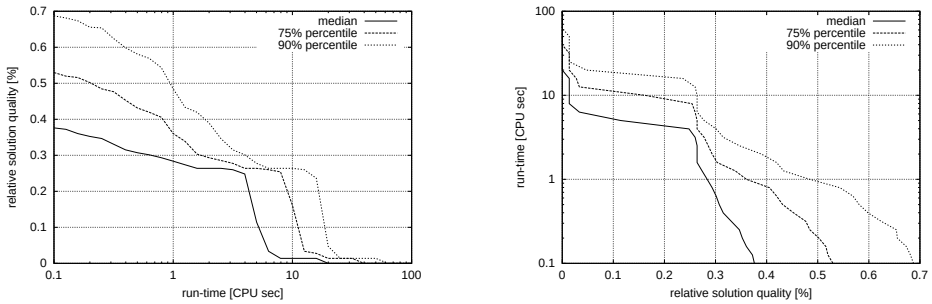


Figure 4.3: *Left:* development of median solution quality, 75% and 90% SQD percentiles over time for same TSP algorithm and problem instance as used in Figure 4.1 (left). *right:* RTQ for the same algorithm and problem instance.

monotonically; in this particular example, we can also see that there is a big and rapid improvement in solution quality after 4–20 CPU seconds. The gradual decrease in solution quality variability in during the first and final phase of the search is rather typical for the behaviour of high-performance SLS algorithms for hard combinatorial optimisation problems; this indicates that for longer runs the algorithm tends to find better solutions in a more consistent way. Note, however, that interesting properties, such as the fact that in our example the SQDs for large run-times are multimodal, or that the variation in run-time increases when higher-quality solutions need to be obtained, cannot be observed from the SQT data shown here.

It is interesting to note that while SQTs are commonly used in the literature for evaluating and analysing the behaviour or SLS algorithms for optimisation problems, the orthogonal concept of qualified RTD statistics dependent on solution quality (RTQs) does not appear to be used at all. Possibly the reason for this lies in the fact that SQTs appear to be intuitively more closely related to the run-time behaviour of an optimisation LVA, and that empirical SQTs can be obtained more easily experimentally (the latter issue will be discussed in more detail in the next section). Nevertheless, RTQs can be useful, for instance in cases where trade-offs between the mean and the

standard deviation of the time required for reaching a certain solution quality q' have to be examined in dependence of q', but where the details offered by a series of qualified RTDs (or the full bivariate RTD) are not of interest.

Example 4.5: Run-time statistics depending on solution quality

Figure 4.3 (right) illustrates several percentiles of the qualified RTDs from Figure **??** for relative solution quality q in dependence of q. Note the difference to the SQT plots in Figure 4.3 (left), which show SQD statistics as a function of run-time.

Empirically Measuring RTDs

Except for very simple algorithms, such as random picking, it is typically not possible to determine RTDs theoretically from properties of a given Las Vegas algorithm. Hence, the true RTDs characterising a Las Vegas algorithm's behaviour are typically approximated by empirical RTDs. For a given instance π of a decision problem, the empirical RTD of an LVA A can be easily determined by performing k (independent) runs of A on π and recording for each successful run the time required to find a solution. The empirical run-time distribution is the cumulative distribution associated with these observations. Each run corresponds to drawing a sample from the true RTD of A on π, and clearly, the more runs are performed, the better will the empirical RTD obtained from these samples approximate the true underlying RTD. For algorithms which are known to be either complete or probabilistically approximately complete (PAC), it is often desirable (athough not always practical) to terminate each run only after a solution has been found; this way, a complete empirical approximation of A's RTD on π can be obtained. In cases where, because of time limits, not all runs are successful, either because the algorithm is essentially incomplete, or because the search was terminated before a solution could be found (with reasonably high probability), a truncated approximation of the true RTD can be obtained from the successful runs. Practically, nearly always a cut-off time is used as a criterion for terminating unsuccessful runs.

More formally, let k be the total number of runs performed with a cutoff-time t', and let $k' \leq k$ be the number of successful runs, *i.e.*, runs during which a solution was found. Furthermore, let $RT(j)$ denote the run-time for the jth entry in a list of all successful runs, ordered according to increasing run-times. The cumulative empirical RTD is then defined by $\widehat{P}(RT \leq t) = \#\{j \mid RT(j) \leq t\}/k$. The ratio $sr = k'/k$ is called the success-rate of A on π with cutoff t'. For algorithms that are known or suspected to be essentially incomplete, the success-rate converges to the asymptotic maximal success probability of A on the given problem instance π, which is formally defined as $p_s^* := \lim_{t\to\infty} P_s(RT_{A,\pi} \leq t)$. For sufficiently high cutoff-time, the empirically determined success-rate can give useful approximations of p_s^*. Unfortunately, in the absence of theoretical knowledge on the success probability or the speed of convergence of the success-rate, the decision whether a given cutoff-time is high enough to get a reasonable estimate of the success probability needs to be based on educated guessing. In practice, the following criterion is often useful in situations, where a reasonably high number of runs (typically between 100 and 10,000) can be performed: When increasing a given cutoff t' by a factor of τ (where τ is typically between 10 and 100) does not result in an increased success-rate, it is assumed that the asymptotic behaviour of the algorithm is observed and that the observed success-rate is a reasonably good approximation of the asymptotic success probability.

Note that in these situations, as well as in cases where success-rates of one cannot be achieved for practical reasons (*e.g.*, due to limited computing resources), certain RTD statistics, in particular all percentiles lower than sr, are still available. Other RTD statistics, particularly the mean time for finding a solution, can be estimated using the following approach: When for cutoff time t', k' out of k runs were successful, the probability for any individual run with cutoff t' to succeed equals the success-rate $sr = k'/k$. Consequently, for n successive (or parallel) independent runs with cutoff t', the probability that at least one of these runs is successful is $1 - (1 - sr)^n$. Using this result, percentiles higher than sr can be estimated for the variant of the respective algorithm that re-initialises the search after each time interval of length t' (static restart). Furthermore, the expected time for finding a solution can be estimated from the mean time over the successful runs by taking into account the expected number of runs required to find a

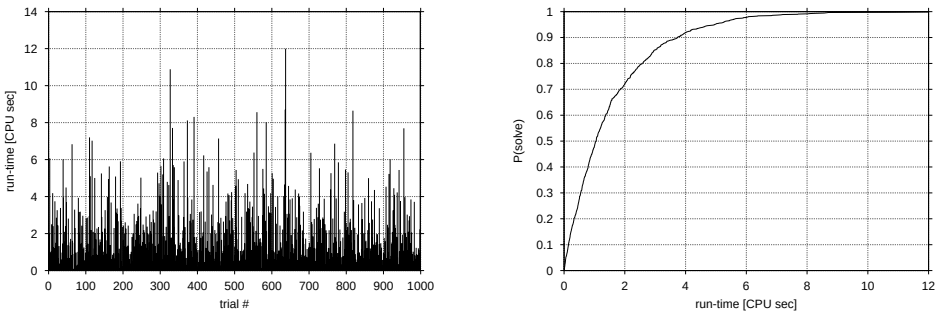


Figure 4.4: Run-time data for WalkSAT/SKC, applied to a hard Random-3-SAT instance for approx. optimal noise setting, 1,000 tries. *Left:* bar diagramm of $rt(j)$; *right:* corresponding RTD.

solution: [1]

$$\widehat{E}(RT') = \frac{1}{k'} \sum_{j=1}^{k'} RT(j) + (1 - sr) \cdot t' \tag{4.1}$$

Note that this estimate applies to the algorithm with static restart and cutoff t'; the mean run-time will generally be different when no restart is used as well as for cutoffs different from t'. In the latter case, RTD information can actually be used for determining performance optimising cutoff settings (*cf.* Section 4.4).

Example 4.6: Raw run-time data *vs* Empirical RTDs

Figure 4.4 (left) shows the raw data from running WalkSAT/SKC, a prominent SLS algorithm for SAT applied to a hard problem instance with 100 variables and 430 clauses; each vertical line represents one run of the algorithm and the height of the lines indicates the CPU time needed for finding a solution. The right side of the same figure shows the corresponding RTD as a cumulative probability distribution curve $(t, \widehat{P}_c(RT \leq t))$. Note that the run-time is extremely variable, which is typical for SLS algorithms for hard combinatorial problems. Clearly, the RTD representation gives a much

[1] This method is equivalent to the one used in [Parkes and Walser, 1996], where a more detailed derivation of this estimate can be found.

more informative picture of the run-time behaviour of the algorithm than simple descriptive statistics summarising the data shown on the left side of Figure 4.4, and, as we will see later in this chapter, it also provides the basis for more sophisticated analyses of algorithmic behaviour. (The graphs shown in Figure 4.4 are based on the same data that was used in Example 4.1.)

For empirically approximating the bivariate RTDs of optimisation LVA A' on a given problem instance π', a slightly different approach is used: During each run of A', whenever the incumbent solution (*i.e.*, the best solution found during this run) is improved, the quality of the improved solution and the time the improvement was achieved is recorded in a solution quality trace. The empirical RTD is derived from the solution quality traces obtained over multiple (independent) runs of A' on π'. Formally, let k be the number of runs performed and let $sq(t, j)$ denote the quality of the best solution found in run j until time t. Then the cumulative empirical run-time distribution of A' on π' is defined by $\widehat{P}(RT \leq t', SQ \leq q') = \#\{j \mid sq(t', j) \leq q'\}/k$. Qualified RTDs and SQDs, as well as SQT and RTQ data, can also be easily derived from the solution traces. With regard to the use of cutoff-times and their impact on the completeness of the empirical RTDs, considerations very similar to the ones discussed for the case of decision problems apply.

CPU Time *vs* Operation Counts

Up to this point, and consistent with a large part of the empirical analyses of algorithmic performance in the literature, we have used CPU-time for measuring and reporting the run-time of algorithms. Obviously, a CPU-time measurement is always based on a concrete implementation and run-time environment (machine and operating system). However, it is often more appropriate, especially in the context of comparative studies of algorithmic performance, to measure run-time in a way that allows to abstract from these factors and that facilitates comparing empirical results across various platforms. This can be done using operation counts, which reflect the number of operations which are considered to contribute significantly towards an algorithm's performance, and cost models, which relate the cost (typically in

terms of run-time per execution) of these operations relative to each other or absolute in terms of CPU-time for a given implementation and run-time environment [Ahuja and Orlin, 1996].

Generally, using operation counts and an associated cost model rather than CPU-time measurements as the basis for empirical studies often gives a clearer and more detailed picture of algorithmic performance. This approach is especially useful for comparative studies involving various algorithms or different variants of one algorithm. Furthermore, it allows to explicitly address trade-offs in the design of SLS algorithms, such as complexity *vs* efficacy of different types of local search steps. To make a clear distinction between run-time measurements corresponding to actual CPU-times and abstract run-times measured in operation counts, we refer to the latter as *run-lengths*. Similarly, we refer to RTDs which are obtained from run-times measured in terms of operation counts as *run-length distributions* or RLDs.

For SLS algorithms, a commonly used operation count is the number of local search steps. In the case of pure SLS methods, such as iterative improvement, there is only one type of local search step, and while the cost or time complexity of such a step typically depends on the size and other properties of the given problem instance, in many cases it is constant or close to constant within and between runs of the algorithm on the same instance. In this situation, measuring run-time in terms of local search steps as elementary operations is often the method of choice; furthermore, run-times measured in terms of CPU-time and run-lengths based on local search steps as basic operations are related to each other by scaling with a constant factor.

Example 4.7: RTDs *vs* RLDs

Figure 4.5 shows RTD and RLD data for the same experiments (solving three Uniform Random-3-SAT SAT instances with 100 variables and 430 clauses each using WalkSAT/SKC, a prominent SLS algorithm for SAT). The operations counted for obtaining RLDs are local search steps; in the case of WalkSAT/SKC, a each local search step corresponds to flipping the truth value assigned to a propositional variable. Note that, when comparing the RTDs and the corresponding RLDs in a semi-log plot, both distributions always have the same shape. This reflects the fact that the CPU-time per step

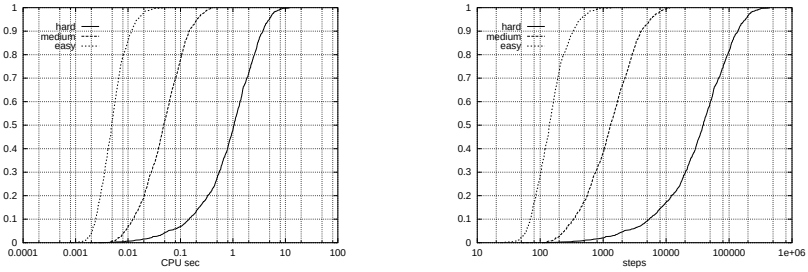


Figure 4.5: RTDs *(left)* and RLDs *(right)* for WalkSAT/SKC, applied to three Uniform Random-3-SAT instances of varying difficulty, based on 1,000 runs and using using an approx. optimal noise parameter setting. **[hh: label y axes; need eps + plt files – TODO(hh)]**

is roughly constant. However, closer examination of the RTD and RLD data reveals that the CPU-time per step is not constant for the three instances; the reason for this is the fact that the hard problem was solved on a faster machine than the medium and easy instances. In this example, the CPU-time per search step is 0.027ms for the hard instance and 0.035ms for the medium and easy instances; the time required for search initialisation is 0.8ms for the hard instance and 1ms for the medium and easy instances. These differences result solely from the difference in CPU speed between the two machines used for running the respective experiments.

In case of hybrid SLS algorithms characterised by GLSM models with multiple frequently used states, such as Iterated Local Search (*cf.* Sections 2.3 and 3.3), the search steps for each state of the GLSM model may have significantly different execution costs (*i.e.* run-time per step) associated with them, and consequently, they should be counted separately. By weigthing these different operation counts relative to each other, using an appropriate cost model, it is typically possible to aggregate them into run-lengths or RLDs. Alternatively, or in situations where the cost of local search steps can vary significantly within a run of the algorithm or between runs on the same instance, it can be useful to use finer-grained elementary operation, such as the number of evaluations of the underlying objective function, or

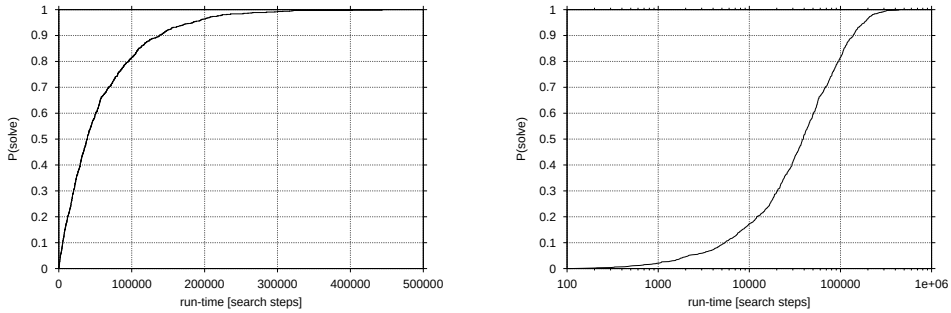


Figure 4.6: *Left:* RLD for WalkSAT/SKC on a hard Random-3-SAT instance for approx. optimal noise parameter setting; *right:* semi-log plot of the same RLD.

the number of updates of internal data structures used for implementing the algorithm's step function.

4.3 RTD-based Analysis of LVA Behaviour

After having introduced RTDs (and related concepts) in the previous section, we now show how these can be used for analysing and characterising the behaviour and relative performance of Las Vegas algorithms. We will start with the quantitative analysis of LVA behaviour based on single RTDs; next, we will show how this technique can be generalised to cover sets and distributions of problem instances. We will then explain how RTDs can be used for the comparative analysis of algorithms before returning to individual algorithms, for which we discuss advanced analysis techniques, including the empirical analysis of asymptotic behaviour and stagnation.

Basic Quantitative Analysis based on Single RTDs

When analysing or comparing the behaviour of Las Vegas Algorithms, the empirical RTD (or RLD) data can be used in different ways. In our experience, graphic representations of empirical RTDs provide often a good starting point. As an example, Figures 4.6 and 4.7 show the RLD for the

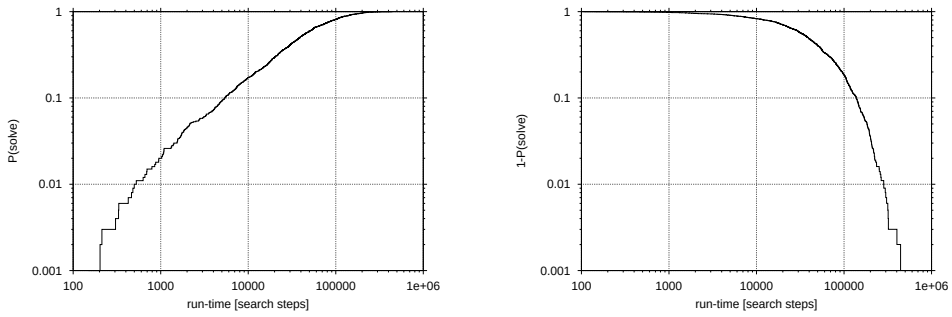


Figure 4.7: Log-log plot of the same RLD as in Figure 4.6 *(left)* and log-log plot of the corresponding failure probability over time *(right)*.

hard problem instance from Figure 4.5 in three different views. Compared to standard representations, semi-log plots (as shown in Figure 4.7, right), give a better view of the distribution over its whole range; this is especially important, since SLS algorithms show often RLDs with extreme variability. Also, when using semi-log plots to compare RLDs, uniform performance differences characterised by a constant factor can be easily detected, as they correspond to simple shifts along the horizontal axis (for an example, see Figure **??**, page **??**). On the other hand, log-log plots of an RLD or its associated failure rate decay function, $1 - rtd(t)$, are often very useful for examining the behaviour of a given Las Vegas algorithm for extremely short or extremely long run-times (*cf.* Figure 4.7).

While graphical representations of RTDs are well suited for investigating and describing the qualitative behaviour of Las Vegas Algorithms, quantitative analyses are usually based on the basic descriptive statistics of the RTD data. For our example, some of the most common standard descriptive statistics, such as the empirical mean, standard deviation, minimum, maximum, and some percentiles, are reported in Table 4.1. Note again the huge variability of the data, as indicated by the large standard deviation and percentile ratios. The latter, like the *variation coefficient*, $vc = stddev/mean$, have the advantage of being invariant to multiplication of the data by a constant, which – as we will see later – is often advantageous when comparing RTDs.

In the case of optimisation LVAs, analogous considerations apply to

mean	stddev	vc	min	max	median
57,606.23	58,953.60	1.02	107	443,496	38,911

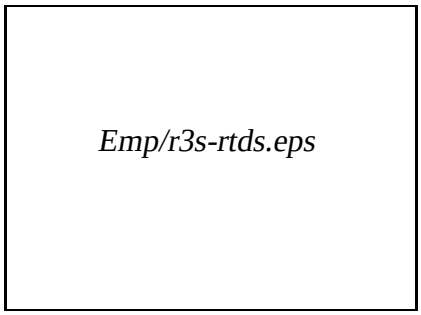
$Q_{0.25}$	$Q_{0.75}$	$Q_{0.1}$	$Q_{0.9}$	$Q_{0.75}/Q_{0.25}$	$Q_{0.9}/Q_{0.1}$
16,762	80,709	5,332	137,863	4.81	25.86

Table 4.1: Basic descriptive statistics for the RLD given in Figures **??** and 4.7; Q_x denotes the x-percentile; the variation coefficient $vc = stddev/mean$ and the percentile ratios Q_x/Q_{1-x} are measures for the relative variability of the run-length data.

graphical representations and standard descriptive statistics of qualified RTDs for various solution quality bounds. Similarly, different graphical representations and summary statistics can be used for analysing and characterising empirical SQDs for various run-time bounds or time-dependent statistics of solution quality; this approach is more commonly followed in the literature, but not always preferable over studying qualified RTDs.

Generally, it should be noted that for directly obtaining sufficiently stable estimates for summary statistics, the same number of test-runs have to be performed as for measuring reasonable empirical RTDs. Thus, measuring RTDs does not cause a computational overhead in data acquisition when compared to measuring only a few simple summary statistics, such as averages and empirical standard deviations. At the same time, arbitrary percentiles and other descriptive statistics can be easily calculated from the RTD data. Furthermore, for optimisation LVAs, bivariate RTDs, qualified RTDs, SQDs, and SQTs can all be easily determined from the same solution quality traces without significant overhead in computation time. Because qualified RTDs, SQDs, and SQTs merely present different views on the same underlying bivariate RTD and since similar considerations apply to all of these, in the following discussion of empirical methodology we will often just explicitly mention RTDs.

Because of the high variability in run-time over multiple runs on the same problem instance that is typical for many SLS algorithms, empirical estimates of mean run-time can be rather unstable, even when obtained



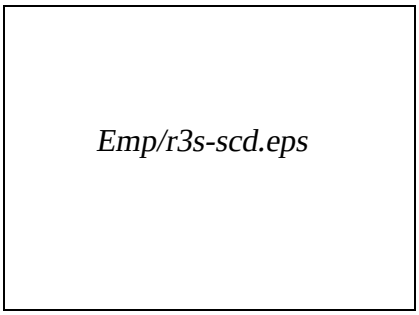


Figure 4.8: *Left:* RLDs for WalkSAT/SKC (approx. optimal noise parameter setting), a prominent SLS algorithm for SAT, applied to three hard Random-3-SAT instances. *Right:* Distribution of median local search cost for the same algorithm across a set of 1,000 Uniform Random 3-SAT instances.

from relatively large numbers of successful runs. This potential problem can be alleviated by using percentiles and percentile ratios instead of means and standard deviations for summarising RTD data with simple descriptive statistics [**?**].

Basic Quantitative Analysis for Ensembles of Instances

In many applications, the behaviour of a given algorithm needs to be tested on a set of problem instances. In principle, the same method as described above for single instances can be applied — RTDs are measured for each instance, and the corresponding sets of graphs and/or associated descriptive statistics are reported.

[hh: need to generate these (data should be available) – TODO(hh)]

Often, LVA behaviour is analysed for a set of fairly similar instances (such as instances of the same type, but different size, or instances from the same random instance distribution). In this case, the RTDs will often have similar shapes (particularly as seen in a semilog-plot) or share prominent qualitative properties, such as being uni- or bi-modal, or having a very prominent right tail. A simple example can be seen in Figure 4.8 (left side), where very similarly shaped RTDs are obtained when applying the same SLS algorithm for SAT (WalkSAT/SKC) to three randomly generated in-

stances from the same instance distribution (Uniform Random-3-SAT with 100 variables and 430 clauses). In such cases, a representative or typical instance can be selected for presentation or further analysis, while the similar data for the other instances is only briefly summarised. It is very important, however, to not assume naively properties of or similarities between RTDs based on a few selected examples only, but to carefully test such assumptions by manual or automated analysis of all or sufficiently many RTDs. In the next section, we will demonstrate how in certain cases, the latter can be done in an elegant and informative way by using functional approximations of RTDs and statistical goodness-of-fit tests.

For bigger sets of instances, such as the sets obtained from sampling random distributions of problem instances, it becomes important to characterise the performance of a given algorithm on individual instances as well as across the whole ensemble. Often (but not always!) when analysing the behaviour of reasonably optimised, probabilistically approximately complete SLS algorithms in such situations, there is a fairly simple scaling relationship between the RTDs for individual problem instances: Given two instances and a desired probability of finding a solution, the ratio of the runtimes required for achieving this solution probability for the two instances is roughly constant. This is equivalent to the observation that in a semilog plot, the two corresponding RTDs essentially differ only by a shift along the time axis. If this is the case, the performance of the given algorithm across the ensemble can be summarised by one RTD for an arbitrarily chosen instance from the ensemble, and the distribution of the mean (or any percentile) of the individual RTDs across the ensemble. The latter type of distribution intuitively captures the cost for solving instances across the set; in the past it has often been referred to as "hardness distribution" – however, it should be noted that without further knowledge, the underlying notion of hardness is entirely relative to the algorithm used rather than intrinsic to the problem instance, and hence this type of distribution is technically more appropriately termed a (local search) cost distribution. An example for such a cost distribution, here for an SLS algorithm for SAT (WalkSAT/SKC) applied to a set of 1,000 Uniform Random-3-SAT instances with 100 variables and 430 clauses each, can be seen in Figure 4.8 (right side).

In reality, the simple multiplicative scaling relationship between any two instances of a given ensemble will hardly ever hold exactly. Hence, depending on the degree and nature of variation between the RTDs for the given

ensemble, it is often reasonable and appropriate to report cost distributions along with a small set of RTDs that have been carefully selected from the ensemble such that they representatively illustrate the variation of the RTDs across the sets. Sometimes, distributions (or statistics) of other basic descriptive RTD statistics across the ensemble of instance, *e.g.*, a distribution of variation coefficients or percentile ratios, can be useful for getting a more detailed picture of the algorithm's behaviour on the given ensemble. It can also be very informative to investigate the correlation between various features of the RTD across the ensemble; specifically, the correlation between the median (or mean) and some measure of variation can be very interesting for understanding LVA behaviour.

Finally, it should be mentioned that when dealing with sets of instances that have been obtained by systematically varying some parameter, such as problem size, it is natural and obvious to study characteristics and properties of the corresponding RTDs (or the cost distributions) in dependence of this parameter. Otherwise, similar considerations as discussed above for ensembles of instances, apply. Again, choosing an appropriate graphical representation, such as a semilogarithmic plot for the functional dependence of mean cost on problem size, is often the key for easily detecting interesting behaviour (*e.g.*, exponential scaling).

In Depth: Benchmark Sets

The selection of benchmark instances is an important factor in the empirical analysis of an algorithm's behaviour, and the use of inadequate benchmark sets can lead to questionable results and misleading conclusions. The criteria for benchmark selection depend significantly on the problem domain under consideration, on the hypotheses and goals of the empirical study, and on the algorithms being analysed. There are, however, some general issues and principles which will be discussed in the following.

Typically, benchmark sets should mainly consist of problem instances that are intrinsically hard or difficult to solve for a broad range of algorithms. While easy instances can be sometimes useful for illustrating or investigating properties of specific algorithms (for example polynomially solvable instances that are hard for certain, otherwise high-performing algorithms), they should not be used as general benchmark problems, as this can easily lead to heavily biased evaluations and assessments of the usefulness of specific algorithms. Similar considerations apply to problem size; small problem instances can sometimes lead to atypical SLS behaviour that does not generalise to larger problem sizes. To avoid such problems

and to facilitate studies on the scaling of SLS performance it is generally advisable to include problem instances of different sizes into benchmark sets.

Furthermore, benchmark sets should contain a diverse collection of problem instances. An algorithm's behaviour can substantially depend on specific features of problem instances, and in many cases at least some of these features are not known *a priori*. Using a benchmark set comprising a diverse range of problem instances reduces the risk of incorrectly generalising from behaviour or performance results that only apply to a very limited class of problem instances.

We distinguish three types of benchmark instances: instances obtained from real-world applications, artificially crafted problem instances, and randomly generated instances. Some combinatorial problems have no real-word applications; where real-world problem instances are available, however, they often provide the most realistic test-bed for algorithms of potential practical interest. Artificially crafted problem instances can be very useful for studying specific properties or features of an algorithm; they are also often used in situations where real-world instances are not available or unsuitable for a specific study (*e.g.*, because they are too large, too difficult to solve, or only very few real-world instances are available). Random problem instance generators have been developed and widely used in many domains, including SAT and TSP. These generators effectively sample from distributions of problem instances with controlled syntactic properties, such as instance size or expected number of solutions. They offer the advantage that large test-sets can be generated easily, which facilitates the application of statistical tests. However, basing the evaluation of an algorithm on randomly generated problem instances only carries the risk of obtaining results that are misleading or meaningless w.r.t. to practical applications.

Ideally, benchmark sets used for empirical studies should comprise instances of all three types. In some cases, it can also be beneficial to additionally use suitable encoded problem instances from other domains. The performance of SAT algorithms, for example, is often evaluated on SAT-encoded instances from domains such as graph colouring, planning, or circuit verification [**?**]. In these cases it is often important to ensure that the encoding schemes used do not produce undesirable features that, for instance, may render the resulting instances abnormally difficult for the algorithm(s) under consideration.

In principle, artificially crafted and randomly generated problem instances can offer the advantage of carefully controlled properties; in reality, however, the behaviour of SLS algorithms is often affected by problem features that are not well understood or difficult to control. (This issue will be further discussed in Chapter 5.) Randomly generated instance sets often show a large variation w.r.t. their non-controlled features, leading to the kind of diversity in the benchmark sets that we have advocated above. On the other hand, this variation often also causes extreme differences in difficulty for instances within the same sample of problem instances (see, *e.g.*, [Hoos, 1998a; Hoos and Stützle, 1999]). This can easily lead to substantial differences in difficulty (as well as other properties) between test-sets sampled from the same instance distribution. As a consequence, comparative

analyses should always evaluate all algorithms on identical test-sets.

To facilitate the reproducibility of empirical analyses and the comparability of results between studies, it is important to use established benchmark sets and to make newly created test-sets available to other researchers. In this context, public benchmark libraries play an important role. Such libraries exist for many domains; widely known examples include TSPLIB (containing a variety of TSP and TSP-related instances), SATLIB (which includes a collection of benchmark instances for SAT) ORLIB (comprising test instances for a variety of problems from Operations Research), TPTP (a collection of problem instances for theorem provers), and CSPLIB (a benchmark library for constraints). Good benchmark libraries are regularly updated with new, challenging problems. Using severely outdated or static benchmark libraries for empirical studies gives rise to various, well-known pitfalls [Hooker, 1994; 1996] and should therefore be avoided as much as possible. Furthermore, good benchmark libraries will provide descriptions and explanations of all problem instances offered, ideally accompanied by references to the relevant literature. Generally, a good understanding of all benchmark instances to be used in the context of an empirical study, regardless of their source, is often crucial for interpreting the results correctly and conclusively.

Comparing Algorithms based on RTDs

Empirical investigations of algorithmic behaviour are frequently performed in the context of comparative studies, often with the explicit or implicit goal to establish the superiority of a new algorithm over existing techniques. In this situation, given two Las Vegas algorithms for a decision problem, one would empirically show that one of them consistently gives a higher solution probability than the other. Likewise, for an optimisation problem, the same applies for a specific (*e.g.*, the optimal) solution quality, or a range of solution qualities. Formally, this can be captured by the concept of probabilistic domination, defined in the following way:

Definition 4.8 (Probabilistic Domination)

Let $\pi \in \Pi$ an instance of a decision problem Π, and A and B be two Las Vegas algorithms for Π. A probabilistically dominates B on π if $\forall t : P_s(RT_{A,\pi} \leq t) \geq P_s(RT_{B,\pi} \leq t)$ and $\exists t : P_s(RT_{A,\pi} \leq t) > P_s(RT_{B,\pi} \leq t)$.

Similarly, for an instance $\pi' \in \Pi'$ of an optimisation problem Π', and optimisation LVAs A' and B' for Π', A' probabilistically

dominates B' on π' for solution quality less than or equal to q, if $\forall t, q : P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'}) \leq q) \geq P_s(RT_{B',\pi'} \leq t, SQ_{B',\pi'}) \leq q)$ and $\exists t : P_s(RT_{B',\pi'} \leq t, SQ_{B',\pi'}) \leq q) > P_s(RT_{B',\pi'} \leq t, SQ_{B',\pi'}) \leq q)$.

A' is said to probabilistically dominate B' on π', if A' probabilistically dominates B' on π' for arbitrary solution quality bound q. □

Remark: A probabilistic domination relation holds between two Las Vegas algorithms on a given problem instance if their respective (qualified) RTDs do not cross over. This provides a simple method for graphically checking probabilistic domination between two LVAs on individual problem instances.

In practice, performance comparisons between Las Vegas algorithms are complicated by the fact that even for a single problem instance, a probabilistic domination does not always hold. This situation is characterised by the occurrence of cross-overs between the corresponding RTDs, indicating, that which of the two algorithms gives better performance, *i.e.*, higher solution probabilities (for a given solution quality bound), depends on the time the algorithm is allowed to run.

Statistical can be used to assess the significance of performance differences. In the simplest case, the Mann-Whitney U-test (or, equivalently, the Wilcoxon rank sum test) can be applied [Sheskin, 2000]; this test determines whether the medians of two samples are equal, hence a rejection indicates significant performance differences. This test can also be used to determine whether the median solution quality achieved by two SLS optimisation algorithms are identical.[2] The more specific hypothesis whether the theoretical RTDs (or SQDs) of two algorithms are identical can be tested using the Kolmogorov-Smirnov test for two independent samples [Sheskin, 2000].

[2]The widely used t-test generally fulfills a similar purpose, but requires the assumption that the given samples are normally distributed with identical variance; since this assumption is often violated in the context of the empirical analysis of SLS behaviour, the t-test should be avoided.

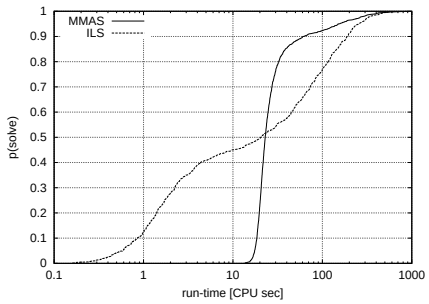


Figure 4.9: Qualified RTDs for two SLS algorithms for the TSP that, applied to a standard benchmark instance, are required to find a solution of optimal quality. The two RTDs cross over between 20 and 30 CPU seconds.

Example 4.8: Comparative RTD Analysis

Figure 4.9 shosw the qualified RTDs for two SLS algorithms for the TSP, Max-Min Ant System ($\mathcal{MM}$AS) and Iterated Local Search (ILS) under the requirement of finding a solution of optimal quality for TSPLIB instance `lin318`. Clearly, there is no probabilistic domination between the two algorithms. The qualified RTD curves cross over at one specific point between 20 and 30 CPU seconds and ILS gives a higher solution probability than MMAS for shorter runs, whereas MMAS is more effective for longer runs. However, for long run-times LVA B again recovers and eventually finds optimal solutions in every single run. Both algorithms eventually find optimal solutions in all runs and hence do not show any evidence for essentially incomplete behavior on this problem instance. Interestingly, it appears that MMAS has practically no chance of finding an optimal solution in less than 10 CPU seconds, while ILS finds optimal solutions with small probability after only 0.2 CPU seconds. (This salient difference in performance is partly explained by the fact that population-based algorithms like MMAS typically incur a certain overhead from maintaining multiple candidate solutions.)

Comparative Analysis for Ensembles of Instances

As mentioned before, empirical analyses of LVA behaviour are mostly performed on ensembles of problem instances. For comparative analyses, in principle this can done by comparing the respective RTDs on each individual problem instance. Ideally, when dealing with two algorithms A and B, one would hope to observe probabilistic domination of A by B (or vice versa) on every instance of the ensemble. In practice, probabilistic domination does not always hold for all instances, and even where it holds, it may not be consistent across a given set of instances. Hence, an instance-based analysis of probabilistic domination (based on RTDs) can partition a given problem ensemble into three subsets: Those on which A probabilistically dominates B, those on which B probabilistically dominates A, and those for which probabilistic domination is not observed, *i.e.*, where A's and B's RTDs cross over. The relative sizes of these partitions give a rather realistic and detailed picture of the algorithms' relative performance on the given set of instances.

Statistical tests can be used to assess the significance of performance differences between two algorithms applied to the same ensemble of instances. These tests are applied to performance measures, such as mean run-time or an RTD percentile, for each algorithm on any problem instance in the given ensemble; hence, they do not capture qualitative differences in performance, particularly as given in cases where there is no probabilistic domination of one algorithm over the other. The binomial sign test as well as the Wilcoxon matched pairs signed-rank test measure whether the median of the paired differences is statistically significantly different from zero, indicating that one algorithm performs better than the other [Sheskin, 2000]. The Wilcoxon test is more sensitive, but requires the assumption that the distribution of the paired differences is symmetric. It may be noted that the widely used t-test for two dependent samples requires assumptions on the normality and homogeneity of variance of the underlying distributions of search cost over the given test-set; this test should not be used for comparing the performance of SLS algorithms, where these assumptions are typically not satisfied.

Particularly for large instance ensembles, it is often useful to refine this analysis by looking at particular performance measures, such as the median run-time, and studying the correlation between A and B w.r.t. these. For

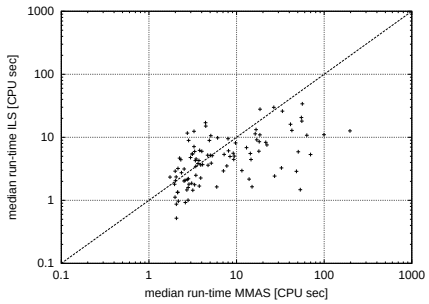


Figure 4.10: Correlation between median run-time required by $\mathcal{MM}$AS *vs* ILS for finding the optimal solutions to instances of a set comprising 100 TSP instances with 300 cities each; each median was measured from 10 runs per algorithm.

qualitative analyses of such correlations, scatter plots can be used in which each instance is represented by one point in the plot, whose coordinates correspond to the performance measure for A and B applied to that instance. Quantitatively, the correlation can be summarised using the empirical correlation coefficient. When the nature of an observed performance correlation seems to be regular (*e.g.*, a roughly linear trend in the scatter plot), a regression analysis can be used to model the corresponding relationship in the algorithms' performance.

To test whether the correlation between the performance of two algorithms are significant, non-parametric tests like Spearman's rank order test or Kendall's tau test can be employed [Sheskin, 2000]. These tests determine whether there is a significant *monotonic* relationship. They are preferable over tests based on Pearson product-moment correlation coefficient, which requires the assumption that the two random variables underlying the performance data stem from a bivariate normal distribution.

Example 4.9: Comparative Analysis on Instance Ensembles

Figure 4.10 shows the correlation between the performance of an ILS algorithm and an ACO algorithm for TSP applied to a set of randomly generated Euclidean TSP instances (the algorithms and problem class are described in

Chapter 8). Clearly, for the ILS algorithm has a lower median run-time than the ACO algorithm for the majority of the problem instances. The median run-times required for finding optimal solutions show a significant correlation (correlation coefficient equal to ???), which indicates that instances that are difficult for one algorithm tend to also be difficult for the other. This suggests that similar features are responsible for rendering instances from this class of TSP instances difficult for both SLS algorithms, a hypothesis that can be investigated further through additional empirical analysis (*cf.* Chapter 5). **[hh: report sizes of three partitions (see above) and spearman's rank order corr coeff, check stat significance – TODO(ts)]**

Peak Performance *vs* Robustness

Most state-of-the-art SLS algorithms have parameters (such as the noise parameter in Randomised Iterative Improvement, or the mutation and crossover rates in Evolutionary Algorithms) that need to be set manually; often, these parameter settings have a very significant impact on the respective algorithms' performance. The existence of such parameters complicates the empirical investigation of LVA behaviour significantly. This is particularly the case for comparative studies, where "unfair parameter tuning", *i.e.*, the use of unevenly optimised parameter settings can give extremely misleading results. Many comparative empirical studies of algorithms in the literature use peak performance w.r.t. parameter settings as the measure for comparing parameterised algorithms. This can be justified by viewing peak performance as a measure of potential performance; more formally, it can be seen as a tight upper bound on performance over algorithm parameterisations.

For peak performance analyses, it is important to determine optimal or close to optimal parameterisations of the respective algorithms. Since differently parameterised versions of the same algorithm can be viewed as distinct algorithms, the RTD-based approach described above can be applied. For continuous parameters, such as the noise parameter mentioned before, a series of such experiments can be used to obtain approximations of optimal values. Peak performance analysis can be very complex, especially when multiple parameters are involved which are typically not independent from each other, or when dealing with complex parameters, such as the temper-

ature schedule for Simulated Annealing, for which the domain of possible settings are extremely large and complex. In such cases, it can be infeasible to get reasonable approximations of optimal parameter settings; in the context of comparative studies, this situation should then be clearly acknowledged and about the same effort should be spent in tuning the parameter settings for every algorithm participating in a direct comparison. An alternative to hand-tuning is the use of automised parameter tuning approaches that are based on techniques from experimental design [**?**; Coy *et al.*, 2000; **?**].

In practice, optimal parameter settings are often not known *a priori*; furthermore, for the same algorithm, optimal parameter settings can differ considerably between problem instances or instance classes. Therefore, robustness of an SLS algorithm w.r.t. suboptimal parameter settings is an important issue. This notion of robustness can be defined as the variation in an algorithm's RTD (or some of its basic descriptive statistics) caused by specific deviations from an optimal parameter setting. It should be noted that typically, such robustness measures can be easily derived from the same data that has been collected for determining optimal parameter settings.

[hh: add example for robustness? (could use RTDs for WalkSAT with different noise settings.)]

A more general notion of robustness of an LVA's behaviour additionally covers other types of performance variation, such as the variation in runtime for a fixed problem instance and a given algorithm (which is captured in the corresponding RTD) as well as performance variations over different problem instances or domains. In all these cases, using RTDs rather than just basic descriptive statistics often gives a much clearer picture of more complex dependencies and effects, such as qualitative changes in algorithmic behaviour which are reflected in the shape of the RTDs (a prominent example for this can be found in Chapter 6). More advanced empirical studies should attempt to relate variation in LVA behaviour over different problem instances or domains to specific features of these instances or domains; such features can be of entirely syntactic nature (*e.g.*, instance size), or they can reflect deeper, semantic properties. In this context, for SLS algorithms, features of the corresponding search spaces, such as density and distribution of solutions, are particularly relevant and often studied; this approach will be further discussed in Chapter 5.

4.4 Characterising and Improving LVA Behaviour

Up to this point our discussion of the RTD-based empirical methodology was focused on analysing specific quantitative and qualitative aspects of algorithmic behaviour as reflected in RTDs. In this section, we first discuss more advanced aspects of empirical RTD analysis. This includes the analysis of asymptotic and stagnation behaviour as well as the use of functional approximations for mathematically characterising entire RTDs. Then, we discuss how a more detailed and sophisticated analysis of RTDs can facilitate improvements in the performance and run-time behaviour of a given Las Vegas algorithm.

Asymptotic Behaviour and Stagnation

In Section 4.1, we defined various norms of LVA behaviour. It is easy to see that all three norms of behaviour, completeness, probabilistic approximate completeness (PAC property), and essential incompleteness, correspond to properties of the given algorithm's theoretical RTDs. For complete algorithms, the theoretical cumulative RTDs will reach one after a bounded time (where the bound depends on instance size). Empirically, for a given time bound, this property can be falsified by finding a problem instance on which at least one run of the algorithm did not produce a solution within the time respective bound. However, it should be clear that a completeness hypothesis can never be verified experimentally, since the instances for which a given bound does not hold might be very rare, and the probability for producing longer runs might be extremely small.

SLS algorithms for combinatorial problems are often incomplete, or in the case of complete SLS algorithms, the time bounds are typically too high to be of any practical relevance. There are, however, in many cases empirically observable and practically significant differences between essentially incomplete and PAC algorithms [**?**]. Interestingly, neither property can be empirically verified or falsified: For an essentially incomplete algorithm, there exists a problem instance for which the probability of not finding a solution in an arbitrarily long run is greater than zero. Since only finite runs can be observed in practice, arbitrarily long unsuccessful runs could hypothetically always become successful after the horizon of observation. On the other hand, even if unsuccessful runs are never observed, there is always

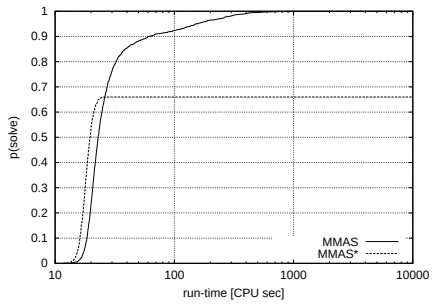


Figure 4.11: Qualified RTDs for two SLS algorithms for the TSP that are required to find an optimal solution of a well-known benchmark instance; $\mathcal{MM}$AS is provably PAC, whereas $\mathcal{MM}$AS* is an essentially incomplete variant of the same algorithm (see text for details). Each RTD is based on 1,000 independent runs of the respective algorithm.

a possibility that the failure probability is just too small compared to the number of runs performed, or the instances on which true failure can occur are not represented in the ensemble of instances tested. However, empirical run-time distributions can provide evidence for (rather than proof of) essential incompleteness or PAC behaviour and hence provide the basis for hypotheses which, in some cases, can then be proven by theoretical analyses. Such evidence primarily takes the form of an apparent limiting success probability that is asymptotically approached by a given empirical RTD.

Example 4.10: Asymptotic Behaviour in Empirical RTDs

Figure **??** shows the qualified RTDs for two variants of an ACO algorithm required to find an optimal solution for TSPLIB instance `lin318`. The RTD for $\mathcal{MM}$AS* shows severe stagnation behaviour; after 26 CPU seconds, the probability for finding a solution does not increase any further, and up to 10,000 CPU seconds not a single additional solution is found. This provides strong evidence (but no proof) that $\mathcal{MM}$AS* is essentially incomplete. Conversely, all 1,000 runs of $\mathcal{MM}$AS were successful and the underlying RTD appears to asymptotically approach one, suggesting that $\mathcal{MM}$AS is probabilistically approximately complete. In fact, $\mathcal{MM}$AS, a

slight extension of $\mathcal{MM}$AS*, is provably PAC, while $\mathcal{MM}$AS* is essentially incomplete. The two algorithms differ only in the key feature that renders $\mathcal{MM}$AS PAC [Stützle and Dorigo, 2002] (details on $\mathcal{MM}$AS can be found in Chapter 8).

In practice, true asymptotic behaviour (such as probabilistic approximate completeness) is less relevant then the rate at which the failure probability of a given LVA decreases over time. Intuitively, a drop in this rate indicates a stagnation in the algorithm's progress towards finding solutions of the given problem instance. Here, we adopt a slightly different view of stagnation, which turns out to be consistent with the intuition described before. This view is based on the fact that in many cases, the probability of obtaining a solution of a given problem instance by using a particular Las Vegas algorithm can be increased by restarting the algorithm after a fixed amount of time (the so-called cutoff time) rather than letting it run longer and longer. Whether or not such a restart strategy yields the desired improvement depends entirely on the respective RTD, and it is easy to see that only for RTDs identical to exponential distributions, restart does not result in any performance loss or improvement [Hoos and Stützle, 1999].

Exponential RTDs are characterised by a constant rate of decay in their right tail, which corresponds to the failure probability, a measure of the probability of not finding an existing solution of a given problem instance within a given amount of time. When augmenting any LVA with a fixed restart policy, the resulting algorithms will show RTDs with exponentially decaying right tails. We let $\lambda(t)$ denote the decay rate obtained for fixed cutoff time t ; then the ratio $\lambda(t^*)/\lambda(t)$, where t^* is the cutoff time leading to maximal decay of the failure probability, is a measure of the efficiency of the LVA, and its reciprocal can be used for quantifying stagnation.

Definition 4.9 (LVA Efficiency and Stagnation)

Let A be a Las Vegas algorithm for a given combinatorial problem Π and $RT_{A,\pi}$ the cumulative run-time distribution function of A applied to a problem instance $\pi \in \Pi$.

Then we define $\lambda_{A,\pi}(t) = d/dt[\log(RT_{A,\pi}](t) = 1/RT_{A,\pi}(t) \cdot d/dt[RT_{A,\pi}](t)$, where $d/df[f]$ denotes the first derivative of a

function f. Furthermore, we define $\lambda^*_{A,\pi} = \inf\{\lambda_{A,\pi}(t) \mid t > 0\}$.

The *efficiency of A on π at time t* is then defined as $\mathit{eff}_{A,\pi}(t) = \lambda^*_{A,\pi}/\lambda_{A,\pi}(t)$. Similarly, the *stagnation ratio of A on π at time t* is defined as $\mathit{stagr}_{A,\pi}(t) = 1/\mathit{eff}_{A,\pi}(t)$, and, the *stagnation of of A on π at time t* is given by $\mathit{stag}_{A,\pi}(t) = \mathit{stagr}_{A,\pi}(t) - 1$.

Finally, we define *minimal efficiency of A on π* as $\mathit{eff}_{A,\pi} = \inf\{\mathit{eff}_{A,\pi}(t) \mid t > 0\}$ and the *minimal efficiency of A on a problem class* Π as $\mathit{eff}_{A,\Pi} = \inf\{\mathit{eff}_{A,\pi} \mid \pi \in \Pi\}$. The maximum stagnation ratio and maximum stagnation on problem instances or problem classes are defined analogously. □

Remark: For empirical RTDs $\widehat{RT}_{A,\pi}$, where the derivative of the RTD is not well-defined, the following estimate for the decay rate can be used: $\lambda_{A,\pi}(t) = -\log(1 - \widehat{RT}_{A,\pi}(t))/t$. It is easy to see that for arbitrarily precisely sampled RTDs, this approaches the theoretical value of $\lambda_{A,\pi}(t)$ defined above.

Under this definition, the stagnation rate is a measure of performance loss compared to the case where an optimal restart strategy is used. It is easy to see that according to the definition, for essentially incomplete algorithms there are problem instances for which the minimum efficiency approaches zero as run-times get arbitrarily long. Constant minimum efficiency of one is observed if and only if the corresponding RTD is an exponential distribution. LVA efficiency greater than one indicates that restarting the algorithm rather then letting it run longer would result in a performance loss; this situation is often encountered for SLS algorithms during the initial search phase.

It should be clear that our measure of LVA efficiency is a relative measure; hence the fact that a given algorithm has high minimal efficiency does *not* imply that this algorithm cannot be further improved. As a simple example, consider Random Picking as introduced in Section 1.5; this primitive search algorithm has efficiency one for arbitrary problem instances and run-times, yet there are many other SLS algorithms which perform significantly better than Random Picking, some of which have a smaller minimal

efficiency. Hence, LVA efficiency as defined above cannot be used to determine the optimality of a given Las Vegas algorithm's behaviour in an absolute way.[3] Instead, it provides a quantitative measure for relative changes in efficiency of a given LVA over the course of its run-time.

Functional Characterisation of LVA Behaviour

Obviously, any empirical RTD, as obtained by running a Las Vegas algorithm on a given problem instance, can be completely characterised by a function — a step function which can be derived from the empirical RTD data in a straightforward way. Typically, if an empirical RTD is a reasonably precise approximation of the true RTD (*i.e.* the number of runs underlying the empirical RTD is sufficiently high), this step function is rather regular and can be approximated well using much simpler mathematical functions.

Such approximations are useful for summarising the observed algorithmic behaviour as reflected in the raw empirical RTD data. But more importantly, they can provide the basis for modelling the observed behaviour mathematically, which is often a key step in gaining deeper insights into an algorithm's behaviour. It should be noted that this general approach is commonly used in other empirical disciplines and can be considered one of the fundamental techniques in science.

In the case of empirical RTDs, approximations with parameterised families of continuous probability functions known from statistics, such as exponential or normal distributions, are particularly useful. Given an empirical RTD and a parameterised family of cumulative probability functions, good approximations can be found using standard model fitting techniques, such as the Marquart-Levenberg algorithm [**?**] or the expectation maximisation (EM) algorithm [**?**]. The quality of the approximation thus obtained can be assessed using standard statistical goodness-of-fit tests, such as the well-known χ^2-test or the Kolmogorov-Smirnov test [Sheskin, 2000]. Both of these tests are used to decide if a sample comes from a population with a specific distribution. While the Kolmogorov-Smirnov test is restricted to continuous distributions, the $\chi 2$ goodness-of-fit test can also be applied to

[3] However, the definition can easily be extended such that an absolute performance measure is obtained; this is done by using the optimal decay rate λ^* over a set of algorithms instead of $\lambda^*_{A,\pi}$ in the definition of LVA efficiency.

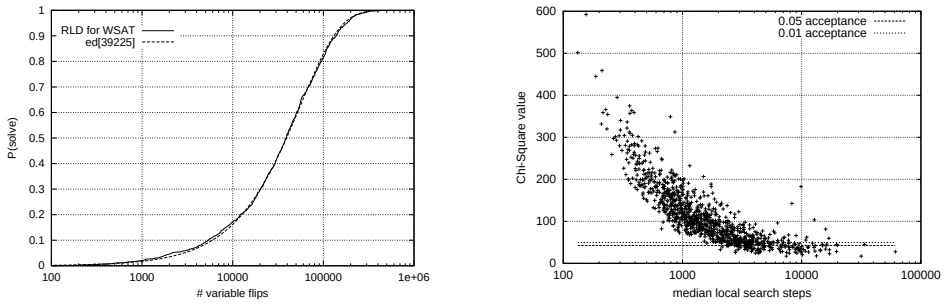


Figure 4.12: *Left:* Best-fit approximation of the RLD from Figure 4.6 by an exponential distribution; this approximation passes the χ^2 goodness-of-fit test at significance level $\alpha = 0.05$. *Right:* Correlation between median run-length and χ^2 values testing RLDs of individual instances versus a best-fit exponential distribution for a test-set of 1,000 hard Random-3-SAT instancesl; the horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance levels of the χ^2-test.

discrete distributions.

Example 4.11: Functional Approximation of Empirical RTDs

Looking at the empirical RLD of WalkSAT/SKC applied to a hard Uniform Random-3-SAT instance from Figure 4.6, one might notice that RLD graph resembles that of an exponential distribution. This leads to the hypothesis that on the given problem instance, the algorithm's behaviour can be characterised by an exponential RLD. To test this hypothesis, we first fit the RLD data with a cumulative exponential distribution function of the form $ed[m](x) = 1 - exp(x/m)$, using the Marquart-Levenberg algorithm (as realised in C. Gramme's Gnufit software) to determine the optimal value for the parameter m. This approximation is shown in Figure 4.12 (left side). Then, we apply the χ^2 goodness-of-fit test to test the hypothesis whether the resulting exponential distribution is identical to the theoretical RTD underlying the empirically observed run-lenghts. In the given example, we resulting χ^2 value of 26.24 indicates that our distribution hypothesis passed

the test at a standard significance level $\alpha = 0.05$.

It is worth noting that, since Las Vegas algorithms (like all algorithms according to the standard definition, see *e.g.* 1.5) are of an inherently discrete nature, their true (theoretical) RTDs are always step functions. However, there are good reasons for the use of continuous probability functions for approximation: For increasing problem sizes these step functions will become arbitrarily fine — an effect which, especially for computationally hard problems, such as SAT or TSP, becomes relevant even for relatively modest and certainly realistically solvable problem sizes. Furthermore, abstracting from the discrete nature of RTDs, often facilitates a more uniform characterisation that is mathematically easier to handle. However, for "very easy" problem instances, *i.e.*, instances which can be solved by a given algorithm in tens or hundreds of basic operations or CPU cycles, the discrete nature of the respective true RTDs can manifest itself — an effect which needs to be taken into account when fitting parameterised functions to such data and testing the statistical significance of the resulting approximations.

Functional Characterisation for Instance Ensembles

Like the previous RTD-based analytical approaches, the functional characterisation of LVA behaviour can be extended from single problem instances to ensembles of instances in a rather straightforward way. For small instance sets, it is generally feasible to perform the approximation and goodness-of-fit test for each instance as described above; for larger ensembles this procedure needs to be automated and its results analysed and summarised in an appropriate way. Overall, similar considerations apply as described in the previous section.

Using this approach, hypotheses on the behaviour of a given LVA on classes or distributions of problem instances can be tested. Hypotheses on an LVA's behaviour on infinite or extremely large sets of instances, such as the set of all SAT instances with a given number of clauses and variables, cannot be proven by this method; however, it allows to falsify such hypotheses or to collect arbitrary amounts of evidence for their validity.

Example 4.12: Functional RTD Approx. for Instance Ensembles

A simple generalisation from the result presented in the previous example results in the hypothesis that for an entire class of SAT instances WalkSAT/SKC's behaviour can be characterised by exponential run-time distributions. Here, we test this hypothesis for a set of $1,000$ Uniform Random-3-SAT instances with 100 variables and 430 clauses. By fitting the RLD data for the individual instances with exponential distributions and calculating the χ^2 as outlined above, we get the result shown in Figure 4.12 (right side), where we plot the median values of the RLDs against the corresponding χ^2 values: although, for most instances, the distribution hypothesis is rejected, we observe a clear correlation between the solution cost of the instances and the χ^2 values; and for almost all of the hardest instances, the distribution hypothesis passes the test. Thus, although our original generalised hypothesis could not be confirmed, the results suggest an interesting modification of this hypothesis. (Further analysis of the easier instances, for which the RLDs could not be well approximated by exponential distributions, shows that there is a systematic deviation in the left tail of the RLDs, while the right tail matches that of an exponential distribution; details on this result can be found in [Hoos and Stützle, 1999; **?**].)

This functional characterisation approach can also be used for analysing and modelling the dependency of LVA behaviour on algorithmic parameters or properties of problem instances (particularly problem size). Furthermore, it facilitates comparative studies of the behaviour of two or more LVA algorithms. In all these cases, reasonably simple, parameterised models of the algorithms' run-time behaviour provide a better basis for the respective analysis than the basic properties and statistics of RTDs discussed before. For example, when studying scaling of an algorithm's run-time behaviour with problem size, having good parameterised functional approximations of the RTDs reduces the investigation to an analysis of the impact of problem size on the model parameters (*e.g.*, the median of an exponential distribution).

As we will see later, such characterisations can also have direct consequences for important issues such as parallelisation or optimal parameterisation of Las Vegas algorithms. At the same time, they can suggest

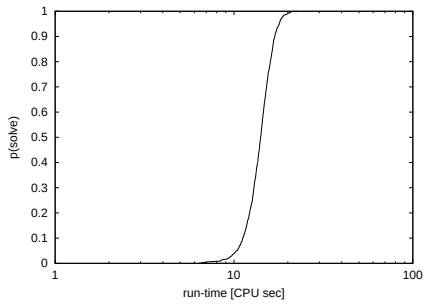


Figure 4.13: Qualified RTD of an ACO algorithm for TSP ($\mathcal{MM}$AS) on TSPLIB instance (`lin318`), based on 1,000 independent runs. The fact that this RTD is steeper than an exponential indicates that restart with any fixed cutoff time will lead to performance loss.

novel interpretations of LVA behaviour and thus facilitate an improved understanding of these algorithms.

Optimal Cutoff Times for Static Restarts

A detailed analysis of an algorithm's RTDs, particularly with respect to asymptotic behaviour and stagnation, can often suggest ways of improving the performance of the algorithm. Arguably the simplest way to overcome stagnation of an SLS algorithm is to restart the search after a fixed amount of time (cutoff time). Generally, based on our definition of search efficiency and stagnation, it is easy to decide whether such a *static restart strategy* can improve the performance of a Las Vegas algorithm A for a given problem instance π. If for all run-times t, the efficiency of A on π at time t, $\mathit{eff}_{A,\pi}(t)$, is larger than one, restart with any cutoff-time t will lead to performance loss. Intuitively, this is the case, when with increasing t, the probability of finding a solution within a given time interval increases, which is reflected in an cumulative RTD graph that is steeper than the exponential distribution $ed[m]$ for which $ed[m](t) = rtd_{A,\pi}(t)$ (an example for such an RTD is shown in Figure **??**). Furthermore, if and only if $\mathit{eff}_{A,\pi}(t) = 1$ for all t, restart at any time t will not change the success probability for any time t'; as mentioned in Section 4.3, this condition is satisfied only if the RTD of A

on π is an exponential distribution. Finally, if and only if $eff_{A,\pi}(t) < 1$ for some run-time t, restarting the algorithm at time t will lead to an increased solution probability for some run-time $t' > t$. This condition is equivalent to the fact that the cumulative RTD graph of A on π is less steep at t' than the exponential distribution $ed[m]$ for which $ed[m](t') = rtd_{A,\pi}(t')$.

In the case where random restart is effective for some cutoff-time t', an optimal cutoff time t_{opt} can intuitively be identified by finding the "left-most" exponential distribution, $ed[m^*]$, that touches the RTD graph of A on π, and the minimal t for which $ed[m^*](t) = rtd_{A,\pi}(t)$ for some t. Formally, this is achieved using the following equations:

$$m^* = \min\{m \mid \exists t > 0 : ed[m](t) - rtd_{A,\pi}(t) = 0\} \quad (4.2)$$

$$t_{opt} = \min\{t \mid t > 0 \wedge ed[m^*](t) - rtd_{A,\pi}(t) = 0\} \quad (4.3)$$

where $rtd_{A,\pi}(t)$ is the theoretical run-time distribution of A on π, and A is incomplete, *i.e.*, $P_s(RT \leq t) < t$ for all (finite) t (note that A may still be probabilistically approximately complete).

Generally, there are two special cases to be considered when solving these two equations. Firstly, we might not be able to determine m^* because the set over which we minimise in the first equation has no minimum. In this case, if the infimum of the set is zero, it can be shown that the optimal cutoff time is either equal to zero or it is equal to $+\infty$ (depending on the behaviour of t_{opt} as m^* approaches zero. Secondly, if m^* as defined by the first equation exists, it might still not be possible to determine t_{opt}, because the set in the second equation does not have a minimum. In this case, there are arbitrarily small times t for which $ed[m^*](t) = rtd_{A,\pi}(t)$, *i.e.*, the two curves are identical on some interval $[0, t')$, and the optimal cutoff time is equal to zero. In practice, optimal cutoff times of zero will hardly occur, since it would require that A can solve π with probability larger than zero for infinitesimally small run-times.

Equations 4.2 and 4.3 apply to theoretical as well as empirical RTDs. In the latter case, however, it is sufficient to consider only run-times t in Equations 4.2 and 4.3 that have been observed in one of the runs underlying the empirical RTD. There is one caveat with this method: Cases in which the optimal cutoff time determined from Equation 4.3 is equal to one of the longest run-times underlying the given empirical RTD should be treated with caution. The reason for this lies in the fact that the high quantiles

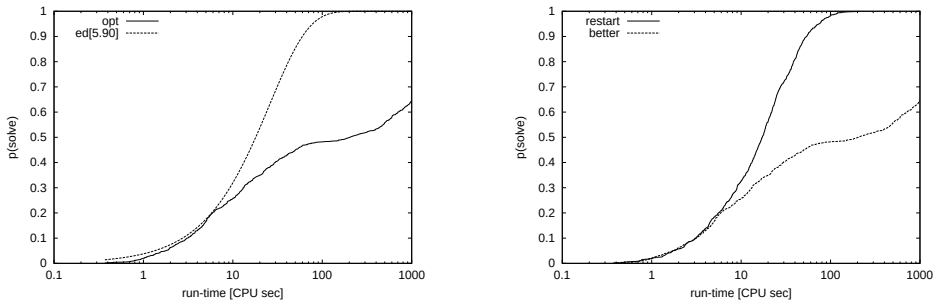


Figure 4.14: Qualified RTD for an ILS algorithm required to find optimal solutions for TSPLIB instance pcb442; note the stagnation behaviour apparent from the RTD graph. *Left:* Optimal cutoff time for static restarts, t_{opt}, and corresponding exponential distribution $ed[m^*]$; *right:* effect of dynamical restart strategy. (Details are given in the text.) **[hh: mark optimal cutoff in left graph; ideally: mark actual restart points in right graph; disp grid – TODO(ts/hh)]**

of empirical RTDs, which correspond to the longest runs, are often rather statistically unstable. Still, using cutoffs based on such extreme run-times may be justified, if there is evidence that the algorithm shows stagnation behaviour.

In the case of SLS algorithms for optimisation problems, optimal cutoff times are determined from qualified RTDs. Clearly, such optimal cutoff times depend on the solution quality bound. In many cases, tighter solution quality bounds (*i.e.*, bounds that are closer to the optimal solution quality) lead to higher optimal cutoff times; yet, for weak solution quality bounds restart with any cutoff time typically leads to performance loss.

Example 4.13: Determining Optimal Cutoff Times for Static Restarts _

Figure 4.14 shows the empirical qualified RTD of a simple ILS algorithm for the TSP for finding optimal solutions to TSPLIB instance pcb442 with $n = 442$ cities. The algorithm was run 1,000 times on a Pentium 700MHz machine with 512MB RAM and unsuccessful runs were terminated after 1,000 CPU seconds. This qualified RTD shows strong stagnation behaviour;

note that this behaviour could not have been observerd when limiting the maximal run-time of the algorithm to less than 5 CPU seconds. Figure 4.14 shows the optimal cutoff time for static restarts, t_{opt}, and the corresponding exponential distribution $ed[m^*]$ determined according to Equations 4.2 and 4.3. The same exponential distribution characterises the rough shape of the RTD for the algorithm using static restarts with cutoff time t_{opt}.

Dynamic Restarts and Other Diversification Strategies

One drawback of using a static restart strategy lies in the fact that optimal cutoff times typically vary considerably between problem instances. Therefore, it would be preferable to re-initialises the search process not after a fixed cutoff time, but depending on search progress. A simple example of such a *dynamic restart strategy* is based on the time that has passed since the currently best solution (w.r.t. evaluation function value) was found; if this time interval exceeds a threshold θ, a restart is performed. (In this scheme, best solutions are not carried over restarts of the search.) The time threshold θ is typically measure in search steps; θ corresponds to the minimal time interval between restarts and is often defined depending on syntactic properties of the given problem instance, in particular, instance size.

Example 4.14: Improving SLS Behaviour Using Dynamic Restarts

Figure 4.14 (right) shows the effect of the simple dynamic restart strategy described above on the ILS algorithm and TSP instance from Example 4.13. Here, for a TSP instance with n vertices, $\theta = n$ is used as the minimal time-interval between restarts . Interestingly, the RTD of ILS with this dynamic restart mechanism is basically identical to the RTD of ILS with static restart for the optimal cutoff-time determined in the previous example. This indicates that the particular dynamic restart mechanism used here is very effective in overcoming the stagnation behaviour of the ILS algorithm without restart.

Restarting an SLS algorithm from a new initial solutions is typically a rather time-consuming operation. Firstly, a certain *setup time* is required for generating a new candidate solution from which the search is started and for initialising the data structures used by the search process accordingly. This setup time is often substantially higher than the time required for performing a search step. Secondly, after initialising the search process, SLS algorithms almost always require a certain number of search steps to reach regions of the underlying search space in which there is a non-negligible chance of finding a solution. These effects are reflected in extremely low success probabilities in the extreme left tail of the respective RTDs. Furthermore, they typically increase strongly with instance size, rendering search restarts a costly operation.

These disadvantages can be avoided by using diversification techniques that are less drastic than restarts in order to overcome stagnation behaviour. One such technique called *fitness-distance diversification* has been used to enhance the ILS algorithm for the TSP mentioned in Example 4.13; the resulting ILS variant shows substantially better performance than the variant using dynamic restarts from Example 4.14. (Details on this enhanced ILS algorithm can be found in Chapter 8.) Another diversification technique that also has the theoretical advantage of rendering the respective SLS algorithm probabilistically approximately complete (PAC), is the so-called *random walk extension* [**?**]. In terms of the GLSM model of the respective SLS algorithms, the random walk extension consists of adding a random walk state in such a way that throughout the search, arbitrarily long sequences of random walk steps can be performed with some (small) probability. This technique was used to obtain state-of-the-art SLS algorithms for SAT, such as Novelty$^+$ (for details, see Chapter 6). Generally, effective techniques for overcoming search stagnation are an important component of advanced SLS strategies and improvements in these techniques can be expected to play a major role in designing future generations of SLS algorithms.

Multiple Independent Runs Parallelisation

Las Vegas algorithms lend themselves to a straightforward parallelisation approach by performing independent runs of the same algorithm in parallel. From the discussion in the previous sections we know that if an SLS algorithms has an exponentially distributed RTD, such a strategy is particularly

effective. Based on a well-known result from the statistical literature [Rohatgi, 1976], if for a given algorithm the probability of finding a solution in t time units is exponentially distributed with parameter m, then the probability of finding a solution in at least one of p independent runs of time t each is exponentially distributed with parameter m/p. Consequently, if we run such an algorithm once for time t, we get exactly the same success probability as when running the algorithm p times for time t/p. By executing these p independent runs in parallel on p processors, an optimal *parallelisation speedup* $S_p = RT_1/RT_p = p$ is achieved, where $RT_1 = t$ is the sequential run-time and $RT_p = t/p$ is the parallel computation time, using p processors. This theoretical result holds for arbitrary numbers of processors.

In practice, SLS algorithms do not have perfectly exponential RTDs; as explained above, there are typical deviations in the left tail which reflect the setup time and initial search phase. Therefore, when the number of processors is high enough that each of the parallel runs becomes very short, the parallelisation speedup will generally be less than optimal. Given an empirical RTD, the parallelisation speedup S_p for reaching a certain success probability p_s can be calculated as follows. RT_1, the sequential run-time required for reaching a solution propability p_s, can be directly determined from the given RTD; technically, $RT_1 = \min\{t' \mid \widehat{P_s}(RT \leq t') \geq p_s\}$. Then the parallel time required for reaching the same solution probability by performing multiple independent runs on p processores is given by

$$RT_p = \min\{t' \mid \widehat{P_s}(RT \leq t') \geq 1 - (1 - p_s)^{1/p}\} \tag{4.4}$$

Using this equation, the minimal number of processors required for achieving the desired success probability within a maximal parallel run-time t_{max} can be easily determined. It is interesting to note that for tighter success guarantees, *i.e.*, for higher p_s, the maximal number of processors for which optimal parallelisation can be achieved, is generally also higher.

Example 4.15: Speedup Through Independent Parallel Runs

Figure 4.15 shows the parallelisation speedup S_p as a function of the number of processors (computed based on Equation **??**) for a high-performing SLS algorithm for SAT (Novelty) applied to two well-known benchmark instances for SAT, the SAT-encoded planning problems `bw_large.b` and `bw_large.c`. The underlying empirical RTDs (determined using instance

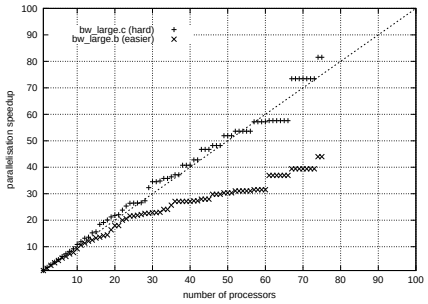


Figure 4.15: Speedup achieved by multiple independent runs parallelisation of a high–performing SLS algorithm for SAT applied to two SAT-encoded instances of a hard planning problem. The diagonal line indicates optimal parallelisation speedup. Note that for the easier instance, the parallelisation speedup becomes suboptimal for smaller number of processors than for the harder instance. (For details, see text.)

specific optimal noise parameter settings of Novelty) are based on 250 successful runs each, and all points of the speedup curves are based on no less than then runs. A desired success probability of $p_s = 0.95$ was used for determining the sequential and parallel runt-times.

`bw_large.c` is much harder than `bw_large.b`, and allows approx. optimal speedup for more than 70 processors; the underlying RTD is almost perfectly approximated by an exponential distribution. For the easier instance, the parallelisation speedup becomes suboptimal for ≥ 10 processors; this is due to the larger relative impact of the setup time and initial search phase on overall run-time.

Generally, using multiple independent runs is an attractive model of parallel processing, since it involves basically no communication overhead and can be easily implemented for almost any parallel hardware and programming environment, from networks of standard workstations to specialised MIMD machines with thousands of processors. The resulting parallel SLS algorithms are precisely captured by the homogeneous cooperative GLSM

model without cooperation introduced in Chapter 3. They are of interest for SLS applications to time-critical tasks (like robot control or online scheduling) as well as for the distributed solving of very large and hard problem instances.

4.5 Further Readings and Related Work

The term *Las Vegas algorithm* was originally introduced by Laszlo Babai [Babai, 1979]. Although the concept is widely known, the literature on Las Vegas Algorithms is relatively sparse. Luby, Sinclair, and Zuckerman have studied optimal strategies for selecting cutoff times [Luby *et al.*, 1993]; closely related theoretical work on the parallelisation of Las Vegas algorithms has been done by Luby and Ertel [Ertel and Luby, 1994]. The application scenarios for Las Vegas algorithms and norms of LVA behaviour covered here have been introduced by Hoos and Stützle [Hoos, 1998a; Hoos and Stützle, 1998].

Run-time distributions have been occasionally observed in the literature for a number of years [Taillard, 1991; Battiti and Tecchiolli, 1992; Taillard, 1994; ten Eikelder *et al.*, 1996]. Their use, however, has been typically restricted to purely descriptive purposes or to obtaining hints on the speedup achievable by performing independent parallel runs of a given sequential algorithm [Battiti and Tecchiolli, 1992; Taillard, 1991]. Taillard specifies general conditions under which super-optimal speedups can be achieved through multiple independent tries parallelisation [Taillard, 1994]. The use of RTDs at the core of an empirical methodology for SLS algorithms was first presented by Hoos and Stützle [Hoos and Stützle, 1998]. Since then, RTD-based methods have been used for the empirical study of a broad range of SLS algorithms for numerous combinatorial problems [**?**; Hoos and Stützle, 2000a; **?**; **?**].

There is some related work on the use of search cost distributions over instance ensembles for the empirical analysis of complete search algorithms. Kwan shows that for different types of random CSP instances, the search cost distributions for recent complete algorithms cannot be characterised by normal distributions [Kwan, 1996]. Frost, Rish, and Vila use continuous probability distributions for approximating the run-time behaviour of complete algorithms applied to randomly generated Random-3-SAT and binary

CSPs from the phase transition region [Frost *et al.*, 1997]. In [Rish and Frost, 1997], this approach is extended to search cost distributions for unsolvable problems from the over-constrained region.

Gomes and Selman studied run-time distributions of backtracking algorithms based on the Brelaz heuristic for solving Quasigroup problems, a special type of CSP, in the context of algorithm portfolios design [**?**]. Interestingly, the corresponding RTDs for the randomised systematic search algorithms they studied can be approximated by "heavy-tailed" distributions, a fact which can be exploited for improving the performance of these algorithms by using a static restart mechanism [Gomes *et al.*, 1997]. Similar results have been obtained for randomised complete algorithms for SAT; at the time, the resulting algorithms showed state-of-the-art performance on many types of SAT instances [Gomes *et al.*, 1998]. Interestingly, the RTDs for some of the most widely known and best-performing SLS algorithms for SAT appear to be well approximated by exponential distributions [Hoos, 1998a; Hoos and Stützle, 1999; **?**] or mixtures of exponentials [**?**]. To our best knowledge, heavy-tailed RTDs have generally not been observed for any SLS algorithm.

A number of specific techniques have proven to be useful in the context of certain types of experimental analyses: Estimates for optimal solution qualities for combinatorial optimisation problems can be obtained using techniques based on insights from mathematical statistics [Golden and Steward, 1985; **?**]. Using solution quality distributions, interesting results have been obtained regarding the behaviour of SLS algorithms as instance size increases [Schreiber and Martin, 1999]. Techniques from experimental design were shown to be helpful in deriving automated (or semi-automated) procedures for tuning algorithmic parameters [Xu *et al.*, 1998; Coy *et al.*, 2000; Birrattari *et al.*, 2002].

Various general aspects of empirical algorithms research are covered in a number of works. There have been several early attempts to provide guidelines for the experimental investigation of algorithms for combinatorial optimisation problems and to establish reporting procedures that improve the reproducibility of empirical results [Crowder *et al.*, 1980; Jackson *et al.*, 1990]. Guidelines on how to report results that are more specific to heuristic methods, including SLS algorithms, are given in [Barr *et al.*, 1995].

Hooker advocates a scientific approach to experimental research in Operations Research and Artificial Intelligence [Hooker, 1994]; this approach

is based on the formulation and careful experimental investigation of hypotheses about algorithm properties and behaviour. General guidelines for the experimental analysis of algorithms are also given by McGeoch and Moret [McGeoch, 1996; McGeoch and Moret, 1999; Moret, 2002]. A recent article by Johnson provides an extensive collection of guidelines and potential pitfalls in experimental algorithms research, including some very practical advice on the topic [Johnson, 2002]. Smith *et al.* give a a similar but more limited overview of potential problems in the experimental analysis of algorithms [**?**].

Statistical methods are at the core of any empirical approach to investigate the behaviour and the performance of SLS algorithms. Cohen's book on empirical methods in artificial intellligence [Cohen, 1995] is becoming a standard text and reference book for the presentation and application of statistical methods not only in AI but also in other fields of Computer Science. For an additional introduction to statistical methods we also recommend [**?**]. The handbook by Sheskin [Sheskin, 2000] is an excellent guide to statistical tests and their proper application; a more specialised introduction to non-parametric statistics can be found in [Conover, 1999; S. Siegel, 1988]. Furthermore, for general techniques of experimental design and the analysis of experimental data we refer to the work of Dean and Voss [Dean and Voss, 2000], and Montgomery [Montgomery, 2000].

4.6 Summary

Empirical methods play a crucial role in analysing the performance and behaviour of SLS algorithms, and appropriate techniques are required for conducting empirical analyses competently and correctly. In this chapter, we motivated why run-time distributions provide a good basis for empiricially analysising the behaviour of SLS algorithms and more generally, members of the broader class of Las Vegas algorithms. We discussed the asymptotic behaviour of Las Vegas algorithms and introduced three application scenarios with different requirements for empirical performance analyses. We then introduced formally the concepts of run-time distributions, qualified run-time distributions, and solution quality distributions, as well as time-dependent solution quality statistics and solution quality dependent run-time statistics. Empirical RTDs can be easily obtained from the same data re-

quired for stable estimates of mean run-times or time-dependent solution quality. We presented and discussed RTD-based methods for the empirical analysis of individual LVAs as well as for the comparative analysis of LVAs, on single problem instances and instance ensembles. We also contrasted peak-performance and robustness analysis and argued that the latter is important to capture dependencies of an algorithm's performance on parameter settings, problem instances, or instance sizes.

The measures of efficiency and stagnation are derived from a given RTD and characterise an algorithm's performance over time; intuitively, these measures indicate how much an algorithm's performance can be improved by a fixed cutoff restart mechanism.

Functional approximations of RTDs with known probability distributions can be used to summarise and mathematically model the behaviour of Las Vegas algorithms. The regularities of LVA behaviour captured by such functional characterisations can facilitate performance analysis, *e.g.*, by suggesting simplified experimental designs in which only the parameter values of a functionally characterised family of RTDs are analysed instead of the complete distributions. Applied to SLS algorithms, this approach can also suggest fundamental properties of the algorithm and provide deeper insights into its behaviour.

Results from the empirical analysis of a SLS algorithms can provide significant leverage for further improvement of their performance. We gave an overview of various approaches to achieving such improvement, including static and dynamic restart mechanisms, adaptive diversification, random walk extension, and parallelisation based on multiple independent tries.

Overall, the importance of empirical analyses in the development and application of SLS algorithms can hardly be overestimated. We believe that the methods and techniques presented in this chapter provide a solid basis for sound and thorough empirical studies on SLS algorithms and thus facilitate the development of better algorithms and an improved understanding of their characteristics and behaviour.

4.7 Exercises

Exercise 4.1 (easy) Give three examples for Las Vegas Algorithms and identify all stochastic elements in these.

Exercise 4.2 (easy) Describe a concrete application domain where the value of a solution to a given problem instance changes over time.

Exercise 4.3 (medium) You are comparing the performance of two SLS algorithms A and B for a combinatorial decision problem. Applied to a well-known benchmark instance you observe the RTDs shown in Figure **??**. What do you learn from these RTDs? Which further experiments do you suggest to decide which algorithm is superior? **[hh: add figure with crossing RTDs, A: PAC, B: essentially incomplete – TODO(hh)]**

Exercise 4.4 (medium) Explain why it is desirable to mathematically model observed RTDs using functional approximations. Do the approximations have to be perfect to be useful?

Exercise 4.5 (medium) [Given: specific description of empirial study - dectect flaws and suggest improvements – TODO(hh)]

Exercise 4.6 (hard) [Outline empirical approach to decide a competition an the best SLS algorithm for TSP. Given: algorithms, TSPLIB instances, you are an expert on TSP and know many applications. – TODO (ts)]