*There is no higher or lower knowledge,*
*but one only, flowing out of experimentation.*
—Leonardo da Vinci, Inventor & Artist

# 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 (generalised) 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 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) is a random variable. These two properties, correctness of the solution computed and run-times

characterised by a random variable, define the class of *(generalised) Las Vegas algorithms (LVAs)*.

DEFINITION 4.1 **Las Vegas Algorithm**

*An algorithm $A$ for a problem class $\Pi$ is a* (generalised) Las Vegas algorithm (LVA) *if, and only if, it has the following properties:*

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

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

***Remark:*** The concept of Las Vegas algorithms was introduced by Babai for the (theoretical) study of randomised algorithms [Babai, 1979]. In theoretical computer science, usually two (equivalent) definitions of a Las Vegas algorithm are used. Both of these definitions assume that the algorithm terminates in finite time; they mainly differ in that the first definition always requires the algorithm to return a solution, while the second definition requires that the probability of returning a correct solution is larger or equal than 0.5 [Hromkovič, 2003]. Note that our definition allows for the possibility that, with an arbitrary probability, the algorithm does not return any solution; in this sense, it slightly generalises the established concept of a Las Vegas algorithm. In the following, we will use the term *Las Vegas algorithm* to refer to this slightly more general notion of an LVA.

Here and in the following, we treat the case in which an SLS algorithm does not return a solution as equivalent to the case where the algorithm does not terminate. Under this assumption, 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 perform 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 Uninformed Random Picking (as introduced in Section 1.5) is a Las Vegas algorithm. Because a solution is generally 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 and/or in the initialisation 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 Chapter 1, 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 perform 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 have the additional property that for fixed run-time, the solution quality achieved by the algorithm, that is, the objective function value of the incumbent candidate solution, is also a random variable.

DEFINITION 4.2 **Optimisation Las Vegas Algorithm**

*An algorithm $A$ for an optimisation problem $\Pi'$ is a* (generalised) optimisation Las Vegas algorithm (OLVA) *if, and only if, it is a (generalised) Las Vegas algorithm, and for each problem instance $\pi' \in \Pi'$ the solution quality achieved after any run-time $t$ is a random variable $SQ(t)$.*

Note that for OLVAs, the solution quality achieved within a bounded run-time is a random variable, and the same holds for the run-time required for achieving or exceeding a given solution quality.

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 SAT (see also Chapter 6), make use of non-deterministic decisions such as randomised tie-breaking rules and fall into the category of generalised Las Vegas algorithms.

It should 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 instance) as well as false negative results (missed correct solutions), while for (generalised) 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 for 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 that do not hold in practical situations. This is, for example, the case for Simulated Annealing, which has been proven to converge towards an optimal solution under certain conditions, one of which is infinitesimally slow cooling in the limit [Hajek, 1988]—which obviously is not practical. Secondly, most complexity results apply to worst-case behaviour, and in the relatively few cases where theoretical average-case results are available, these are often 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 that 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 nondeterministic quantum effect 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 in physics, chemistry and biology, it is largely a collection of these models that constitutes theoretical frameworks, whereas in computer science, theory is almost exclusively derived from mathematical foundations. Historical reasons aside, this difference is 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 insufficient 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 to be 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, that is, 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 it is often also relevant in practical applications. In the following, we distinguish not only between complete and incomplete Las Vegas algorithms, but also introduce a third category, the 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 when 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 $\Pi$, 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, and only if, it is not PAC, that is, 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 $\Pi$, if the respective properties hold for the corresponding sets of instances instead of $\Pi$.

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, many variants of Iterated Local Search and most tabu search algorithms are essentially incomplete (see also in-depth section on page 155*ff.*).

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 interest. A similar situation arises in many practical situations for search algorithms whose completeness is achieved by different means, such as systematic backtracking.

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 sufficient time, can be used to 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 be applied to the associated decision problems in a straightforward way, using the following generalisations:

DEFINITION 4.4 **Asymptotic Behaviour of OLVAs**

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

*$A'$ is called*

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

With respect to finding optimal solutions, we use the terms *complete*, *approximately complete* and *essentially incomplete* synonymously for 1*-complete*, *approximately* 1*-complete* and *essentially* 1*-incomplete*, where $q'$ is the optimal solution quality for the given problem instance.

**IN DEPTH** PROBABILISTIC APPROXIMATE COMPLETENESS AND 'CONVERGENCE'

The PAC property states that by running an algorithm sufficiently long, the probability of not finding a (optimal) solution can be made arbitrarily small. Hence, an increase of the run-time typically pays off in the sense that it also increases the chance that the algorithm finds a solution.

As previously stated, several extremely simple SLS algorithms have the PAC property. For example, it is rather straightforward to show that Uninformed Random Picking is PAC. Under some simple conditions, several more complex SLS algorithms can also be proven to have the PAC property. One condition that is sufficient for guaranteeing the PAC property for an SLS algorithm $A$ is the following: there exists $\epsilon > 0$ such that in each search step, the distance to an arbitary, but fixed (optimal) solution $s^*$ is reduced with probability greater than or equal to $\epsilon$, where distance is measured in terms of a minimum length search trajectory of $A$ from its current position to $s^*$. To see why this condition implies that $A$ is PAC, consider a situation where the distance between the current candidate solution and $s^*$ is equal to $l$. In that case, we can compute a lower bound on the probability of reaching $s^*$ in exactly $l$ steps as $\epsilon^l$. Since the diameter $\Delta$ of the given neighbourhood graph is an upper bound for $l$, we can give a worst-case estimate for the probability of reaching $s^*$ from an arbitrary candidate solution within $\Delta$ steps as $\epsilon^\Delta$. Any search trajectory of length $t > \Delta$ can be partitioned into segments of length $\Delta$, for each of which there is an independent probability of at least $\epsilon^\Delta$ of reaching solution $s^*$; consequently, the probability that $A$ does not reach $s^*$ within a trajectory of length $t$ can be bounded by

$$(1 - \epsilon^\Delta)^{\lfloor t/\Delta \rfloor}.$$

Since $\epsilon^{\Delta} > 0$, by choosing $t$ sufficiently large, this failure probability can be made arbitrarily small, and consequently, the success probability of $A$ converges to 1 as the run-time approaches infinity, that is, $A$ is PAC.

A proof along these lines can easily be applied to SLS methods such as Randomised Iterative Improvement (see Chapter 2, page 72*ff.*) and — with some additional assumptions on the maximum difference between the evaluation function values of neighbouring candidate solutions — Probabilistic Iterative Improvement (see Chapter 2, page 74*f.*). The PAC property has also been proven for a number of other algorithms, including Simulated Annealing [Geman and Geman, 1984; Hajek, 1988; Lundy and Mess, 1986], specific Ant Colony Optimization algorithms [Gutjahr, 2002; Stützel and Dorigo, 2002], Probabilistic Tabu Search [Faigle and Kern, 1992], deterministic variants of Tabu Search [Glover and Hanafi, 2002; Hanafi, 2001] and Evolutionary Algorithms [Rudolph, 1994].

For many SLS algorithms, properties that are stronger than the PAC property have been proven. (In some sense, the strength of the result depends on the notion of probabilistic convergence proven; see Rohatgi [1976] for the different notions of probabilistic convergence.) In particular, in some cases it can be proven that if $s_k$ is the candidate solution at step $k$, then $\lim_{k\mapsto\infty} P(s_k \in S) = 1$ (i.e., the probability that the current search position $s_k$ is a (optimal) solution tends to one as the number of iterations approaches infinity). In other words, if run sufficiently long, the probability for the algorithm to visit any non-solution position becomes arbitrarily small. This is exactly the type of result that has been proven, for example, for Simulated Annealing [Hajek, 1988]. This type of convergence can be nicely contrasted with the definition of PAC which implies directly that $\lim_{k\mapsto\infty} P(\hat{s}_k \in S) = 1$, where $\hat{s}_k$ is the incumbent solution after step $k$. Clearly, the former type of convergence result is stronger in that it implies the PAC property but not vice versa.

However, for practical purposes, this stronger sense of convergence is irrelevant. This is because for decision problems, once a solution is found that satisfies all logical conditions, the search is terminated, and for optimisation problems, the best candidate solution encountered so far is memorised and can be accessed at any time throughout the search. Additionally, these stronger convergence proofs are often based on particular parameter settings of the algorithm that result in an extremely slow convergence of the success probability that is not useful for practically solving problems; this is the case for practically all 'strong' convergence proofs for Simulated Annealing.

The significance of a PAC result is that the respective algorithm is guaranteed to not get permanently trapped in a non-solution area of the search space. What is of real interest in practice, however, is the rate at which the success probability approaches one. While PAC proofs typically give, as a side effect, a lower bound on that rate, this bound is typically rather poor, since it does not adequately capture the heuristic guidance utilised by the algorithm. In fact, in proofs of the PAC property, one typically has to assume a worst-case scenario in which the search heuristic is maximally misleading. However, empirical results indicate that in many cases variants of SLS methods that are PAC perform significantly better in practice than non-PAC variants [Hoos, 1999a; Hoos and Stützle, 2000a; Stützle and Dorigo, 2002], which gives a strong indication that in practice the convergence rate of PAC algorithms is much higher than the theoretical analyses would suggest. In general, proving better bounds on the convergence rate of

state-of-the-art SLS algorithms appears to be very challenging, but is doubtlessly an interesting direction of theoretical work on SLS algorithms.

## 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, that is, 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 off line or in a non-realtime environment where it does not really matter how long 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 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 some given time $t_{max}$; 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).

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) this measure is basically meaningless. Type 3 is not only the most general class of application scenario, but these scenarios are also the most realistic. The reason for this is the fact that real-world problem solving usually involves time-constraints that 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 (in particular, CPU time) of finding a solution.

As an example, consider a situation where hard combinatorial problems have to be solved on line 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, which use utility functions of the form $U(t) := \max\{u_0 - c \cdot t, 0\}$ and $U(t) := e^{-\lambda \cdot t}$, respectively (see, e.g., [Poole et al., 1998]). Based on the utility function, the weighted solution probability $U(t) \cdot P_s(RT \leq t)$ 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 $t$, weighted by the utility $U(t, q)$ can be used as a performance criterion.

## 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 required for solving a problem instance or for 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 Las Vegas algorithms are evaluated without *a priori* knowledge of the application scenario, such that the 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 decision problems class $\Pi$, 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 $t$. The* run-time distribution (RTD) of $A$ on $\pi$ *is the probability distribution of the random variable $RT_{A,\pi}$, which is characterised by 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 an optimisation problem $\Pi'$ 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 $\pi'$ finds a solution of quality less than or equal to $q$ in time less than or equal to $t$. The* run-time distribution (RTD) *of $A'$ on $\pi'$ 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 $rtd(t,q) = P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q)$.*

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 a typical run-time distribution for an SLS algorithm applied to an instance of a hard combinatorial decision problem. The RTD is represented by a cumulative probability distribution curve $(t, \widehat{P}_s(RT \leq t))$ that 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); $\widehat{P}_s(RT \leq t)$ represents an empirical estimate for the success probability $P_s(RT \leq t)$.

Figure 4.1 (right) shows the bivariate RTD for an SLS optimisation algorithm applied to an instance of a hard combinatorial optimisation 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 algorithm applied to instance `pcb442` with 442 vertices from TSPLIB, a benchmark library for the TSP (details on SLS algorithms

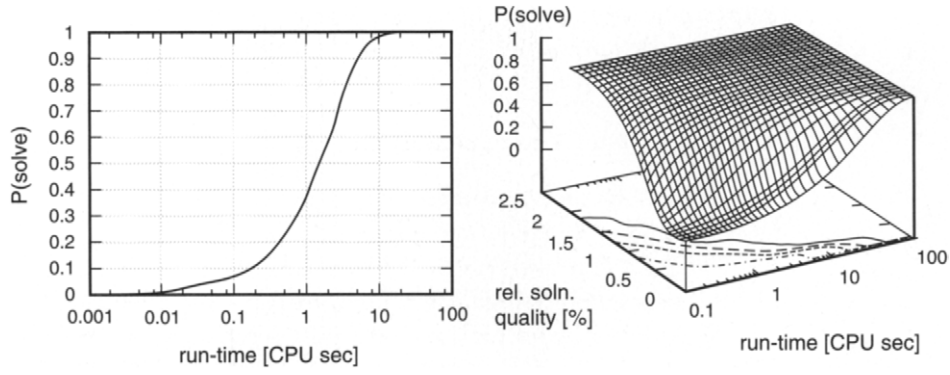



***Figure* 4.1** Typical run-time distributions for SLS algorithms applied to hard combinatorial decision (*left*) and optimisation problems (*right*); for details, see text.

and benchmark problems for the TSP will be discussed in Chapter 8). Note how the contours of the three-dimensional bivariate RTD surface projected into the run-time/solution quality plane reflect the tradeoff 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, quantiles 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.

Unlike 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 case, 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 Run-Time Distributions

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

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 for increasingly tight solution quality thresholds 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 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 $q^*$ is calculated as $q/q^* - 1$; the *relative solution qualities* thus obtained are often expressed in percent. (In cases where $q^* = 0$, sometimes the solution quality is normalised by dividing it by the maximal possible objective function value.) If optimal solutions are not known, one possibility is to evaluate the SLS algorithms w.r.t. the best

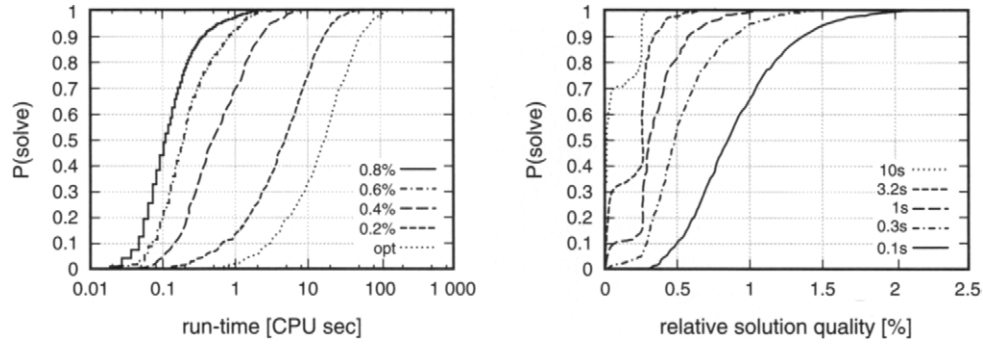



***Figure 4.2*** *Left:* Qualified RTDs for the bivariate RTD from Figure 4.1. *Right:* SQDs for the same RTD .

known solutions. This method, however, has the potential disadvantage that best known solutions may change. 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]. Alternatively, there are statistical methods for estimating optimal solution qualities in cases where tight lower bounds are not available [Dannenbring, 1977; Golden and Steward, 1985].

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 4.1 (page 159*f.*). Note that when tightening the solution quality bound, the qualified RTDs get shifted to the right and appear somewhat steeper in the semi-log 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, for example, in the variation coefficient, that is, the standard deviation of the RTD divided by its mean). The latter observation reflects a rather typical property of SLS algorithms for hard optimisation problems.

## Solution Quality Distributions

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 for an optimisation problem $\Pi'$ and let $\pi' \in \Pi'$ be a solvable problem instance. If $rtd(t,q)$ is the RTD of $A'$ on $\pi'$, then for any run-time $t'$, the* solution quality distribution (SQD) of $A'$ on $\pi'$ 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. 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 of solution quality variability. In particular, for a 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, that is, marginal distributions of the bivariate empirical RTD from Example 4.1 (page 159*f.*), 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 correspond to the peaks in probability density (modes).

An interesting special case arises for iterative improvement algorithms that cannot escape from local minima regions of the given evaluation function. Once they have encountered such a local minima region, these essentially incomplete

algorithms are unable to obtain any further improvements in solution quality. Consequently, as the run-time is increased towards infinity, the respective SQDs approach a non-degenerate probability distribution. For simple iterative improvement methods that only allow strictly improving steps and always perform such steps when they are possible, this *asymptotic SQD* is reached after finite run-time on any given problem instance. Moreover, in this case asymptotic SQDs can be easily sampled empirically by simply performing multiple runs of the algorithm and recording the quality of the incumbent solution upon termination of each of these runs. Asymptotic SQDs are useful for characterising the performance of simple iterative improvement algorithms (cf. Example 2.1, page 64*f.*).

The information provided by asymptotic SQDs is typically well complemented by the (univariate) run-time distribution that captures the time spent by the algorithm before terminating, independent of the final solution quality reached in this run. To distinguish this type of run-time distribution from the complete bivariate run-time distribution that characterises the behaviour of any optimisation LVA and from the notion of a qualified run-time distribution discussed above, we refer to it as *termination-time distribution (TTD)*. Note that unlike qualified RTDs, TTDs are not marginal distributions of the underlying bivariate RTD.

Asymptotic SQDs are also very useful for characterising the performance of purely constructive search algorithms, such as the Nearest Neighbour Heuristic for the TSP (cf. Chapter 1, Section 1.4), which terminate as soon as a complete candidate solution has been obtained. Unlike (perturbative) iterative improvement algorithms, constructive search algorithms typically terminate after a fixed, instance-dependent number of search steps. Consequently, they typically show much less variability in run-time (or no variability at all), which simplifies comparative performance analyses.

## 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. It is often preferable to use SQTs that reflect the development of quantiles (e.g., the median) of the underlying SQDs over time, since quantiles are typically statistically more stable than means. Furthermore, SQTs based on SQD quantiles 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

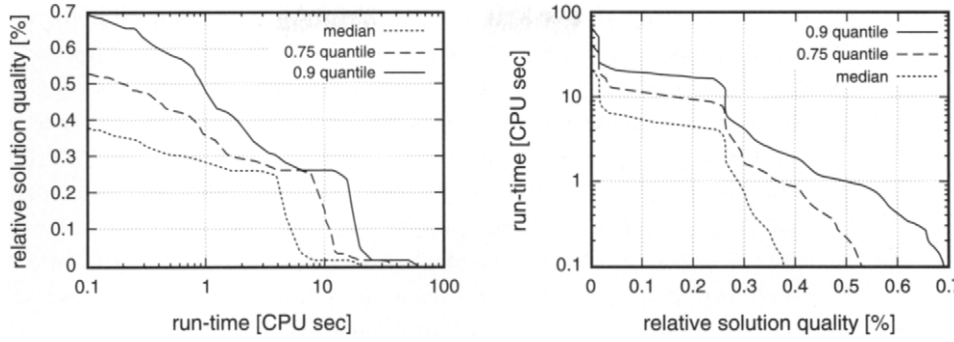



*Figure 4.3* *Left:* Development of median solution quality, 0.75 and 0.9 SQD quantiles over time for the same TSP algorithm and problem instance as used in Figure 4.1 (page 160). *Right:* RTQ for the same algorithm and problem instance.

a full SQD series, and hence a complete bivariate RTD; they are particularly well 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 median solution quality and its variability over time, obtained from the same empirical data underlying the bivariate RTD from Example 4.1 (page 159*f.*). 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 quantiles improve substantially and consistently; in this particular example, we can also see that there is a large and rapid improvement in solution quality after 4–20 CPU seconds. The gradual decrease in solution quality variability 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; it 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 of 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 are more intuitively related to the run-time behaviour of an optimisation LVA, and that empirical SQTs can be measured more easily (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 quantiles of the qualified RTDs from Figure 4.2 (page 162) 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 Uninformed Random Picking, it is typically not possible to analytically determine RTDs for 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 $\pi$ of a decision problem, the empirical RTD of an LVA $A$ can be easily determined by performing $k$ independent runs of $A$ on $\pi$ and recording for each successful run the time required to find a solution. The empirical run-time distribution is given by the cumulative distribution function associated with these observations. Each run corresponds to drawing a sample from the true RTD of $A$ on $\pi$, and clearly, the more runs are performed, the better will the empirical RTD obtained from these samples approximate the true underlying RTD. For algorithms that are known to be either complete or probabilistically approximately complete (PAC), it is often desirable (although 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 $\pi$ can be obtained. In cases where not all runs are successful, either because the algorithm is essentially incomplete or because some runs were terminated before a solution could be found, a truncated approximation of the true RTD

can be obtained from the successful runs. Practically, nearly always a cutoff 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, that is, runs during which a solution was found. Furthermore, let $rt(j)$ denote the run-time for the $j$th entry in a list of all successful runs, ordered according to increasing run-times. The cumulative empirical RTD is then defined by $\widehat{P}_s(RT \leq t) := \#\{j \mid rt(j) \leq t\}/k$. The ratio $sr := k'/k$ is called the success ratio of $A$ on $\pi$ with cutoff $t'$. For algorithms that are known or suspected to be essentially incomplete, the success ratio converges to the asymptotic maximal success probability of $A$ on the given problem instance $\pi$, 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 ratio 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 ratio, the decision whether a given cutoff time is high enough to obtain 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 $\tau$ (where $\tau$ is typically between 10 and 100) does not result in an increased success ratio, it is assumed that the asymptotic behaviour of the algorithm is observed and that the observed success ratio is a reasonably good approximation of the asymptotic success probability.

Note that in these situations, as well as in cases where success ratios equal to one cannot be achieved for practical reasons (e.g., due to limited computing resources), certain RTD statistics, in particular all quantiles 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 can be estimated by the success ratio $sr := k'/k$. 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, quantiles 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 solution as well as the mean run-time of the failed runs (see also Parkes and Walser [1996]):

$$\widehat{E}(RT') = \widehat{E}(RT_s) + (\frac{1}{sr} - 1) \cdot \widehat{E}(RT_f), \tag{4.1}$$

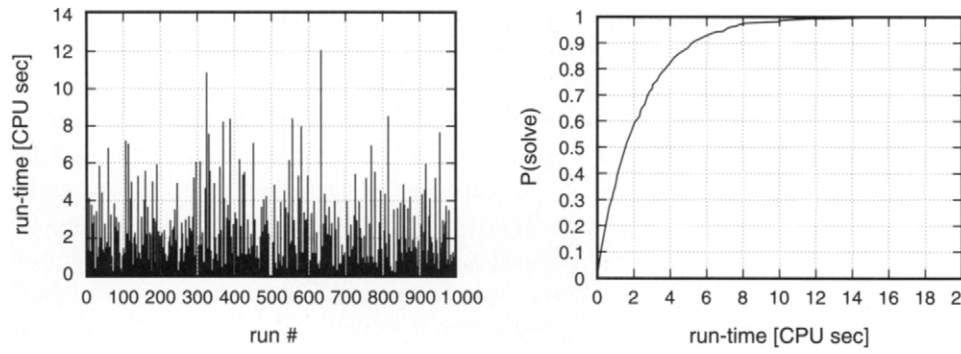



***Figure 4.4*** Run-time data for WalkSAT/SKC, a prominent SLS algorithm for SAT, applied to a hard Random 3-SAT instance for approx. optimal noise setting, 1 000 tries. *Left:* bar diagram $rt(j)$; *right:* corresponding RTD .

where $\widehat{E}(RT_s) := 1/k' \cdot \sum_{j=1}^{k'} rt(j)$ is the average run-time of a successful run, and $\widehat{E}(RT_f)$ is the average run-time of a failed run. Using a static restart mechanism with a fixed cutoff time $t'$ results in $\widehat{E}(RT_f) := t'$. Note that in this case, $\widehat{E}(RT')$ depends on the cutoff time $t'$; in fact, RTD information can be used for determining values $t'$ that lead to optimal performance in the sense of minimal expected solution time $\widehat{E}(RT')$ (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, on 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 $(t, \widehat{P}_s(RT \leq t))$. 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 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 used in Example 4.1 on page 159*f*.)

For empirically approximating the bivariate RTDs of an optimisation LVA $A'$ on a given problem instance $\pi'$, a slightly different approach is used. During each run of $A'$, whenever the incumbent solution (i.e., the best candidate solution found during this run) is improved, the quality of the improved incumbent solution and the time at which 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 $\pi'$. 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 $\pi'$ is defined by $\widehat{P}_s(RT \leq t', SQ \leq q') := \#\{j \mid sq(t', j) \leq q'\}/k$. Qualified RTDs and SQDs as well as SQT and RTQ data and, where appropriate, asymptotic SQDs and TTDs can also be easily derived from the solution quality traces. With regard to the use of cutoff times and their impact on the completeness of the empirical RTDs, considerations very similar to those 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 (i.e., 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 one to abstract from these factors and that facilitates comparisons of empirical results across various platforms. This can be done using *operation counts*, which reflect the number of operations that 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 one 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 obtained

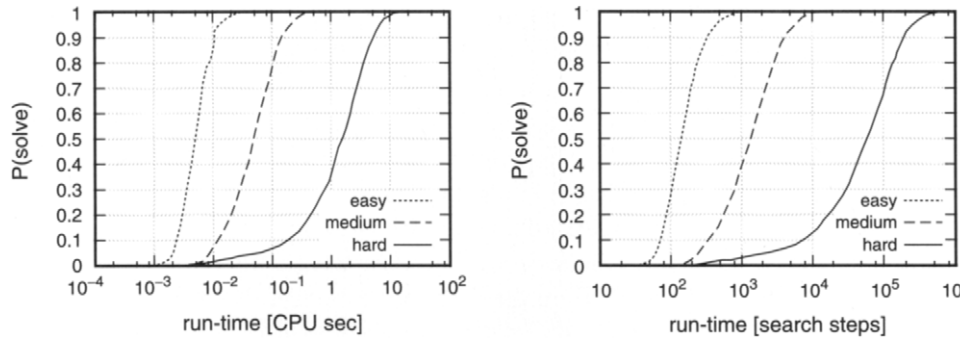



***Figure* 4.5** RTDs *(left)* and RLD *(right)* for WalkSAT/SKC, a prominent SLS algorithm for SAT, applied to three Uniform Random 3-SAT instances of varying difficulty, based on 1 000 runs per instance (using an approx. optimal noise parameter setting).

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 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, each local search step corresponds to flipping the truth value assigned to one 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 is roughly constant. However, closer examination of the RTD and RLD data reveals that the CPU time per step differs between 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 the case of hybrid SLS algorithms characterised by GLSM models with multiple frequently used states, such as Iterated Local Search (cf. Chapter 2, Section 2.3 and Chapter 3, Section 3.3), the search steps for each state of the GLSM model may have significantly different execution costs (i.e., run-time per step) and, consequently, they should be counted separately. By weighting 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 may be necessary to use finer-grained elementary operations, such as the number of evaluations of the underlying objective function, or 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 several 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 many cases, graphic representations of empirical RTDs provide a good starting point. As an example,

Figures 4.6 and 4.7 show the RTD for the hard problem instance from Figure 4.5 (page 170) in three different views. Compared to standard representations, semi-log plots (as shown on the right side of Figure 4.7) give a better view of the distribution over its entire range; this is especially relevant for RTDs of SLS algorithms, which often show an extreme variability in run-time. Also, when using semi-log plots to compare RTDs, 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 4.5, page 170). On the other hand, log-log plots of an RTD or its associated failure rate decay function,

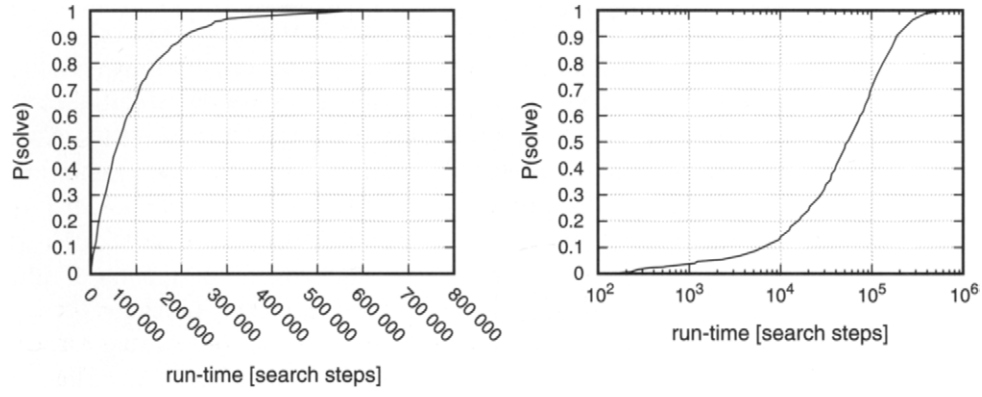



***Figure 4.6*** *Left:* RLD for WalkSAT/SKC, a prominent SLS algorithm for SAT, on a hard Random 3-SAT instance for approx. optimal noise parameter setting. *Right:* Semi-log plot of the same RLD.

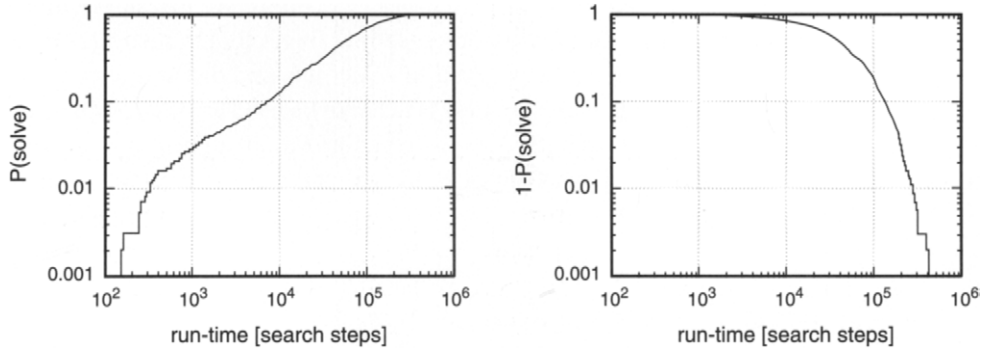



***Figure 4.7*** Log-log plot of the same RLD as in Figure 4.6 *(left)* and log-log plot of the corresponding failure probability over time *(right)*.

$1 - rtd(t)$, 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 summarising the RTD data with basic descriptive statistics. For our example, some of the most common standard descriptive statistics, such as the empirical mean, standard deviation, minimum, maximum and some quantiles, are reported in Table 4.1. Note again the huge variability of the data, as indicated by the large standard deviation and quantile 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 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 reasonably accurate 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 quantiles and other descriptive statistics can be easily calculated from the RTD data. Furthermore, in the case of 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

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

***Table* 4.1** Basic descriptive statistics for the RLD shown in Figures 4.6 and 4.7; $q_x$ denotes the $x$-quantile; the variation coefficient $vc := stddev/mean$ and the quantile ratios $q_x/q_{1-x}$ are measures for the relative variability of the run-length data.

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 from relatively large numbers of successful runs. This potential problem can be alleviated by using quantiles and quantile 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.

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 semi-log 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 instances from the same instance

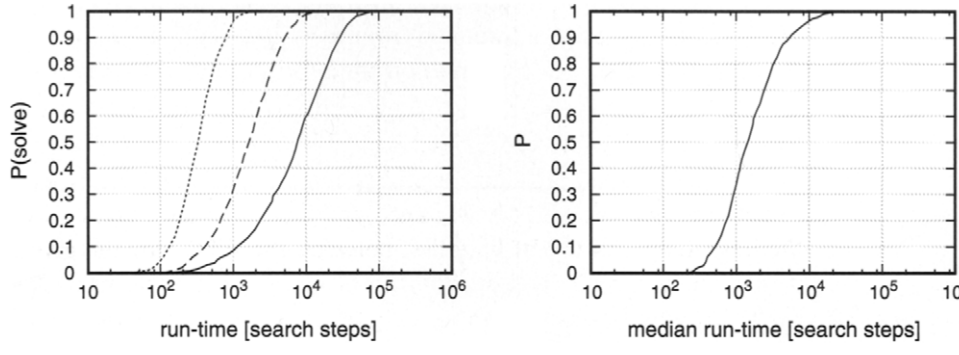



***Figure 4.8*** *Left:* RLDs for WalkSAT/SKC (using an approx. optimal noise parameter setting), a prominent SLS algorithm for SAT, applied to three hard Random 3-SAT instances. *Right:* Distribution of median local search cost for the same algorithm across a set of 1 000 Uniform Random 3-SAT instances.

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 analogous data for the other instances are only briefly summarised. It is very important, however, to not naïvely assume 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 Section 4.4, 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 entire 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 run-times required for achieving this solution probability for the two instances is roughly constant. This is equivalent to the observation that in a semi-log 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 quantile) 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 *search cost distribution (SCD)*. An example for such a SCD, 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, is shown 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—for example, a distribution of variation coefficients or quantile ratios—can be useful for obtaining 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 semi-log plot for the functional dependence of mean run-time 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-world 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 (see, e.g., Hoos and Stützle [2000a]). In these cases, it is often important to ensure that the respective encoding schemes 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 [1998], 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 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 for 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* $\Pi$*, and let* $A$ *and* $B$ *be two Las Vegas algorithms for* $\Pi$. $A$ probabilistically dominates $B$ on $\pi$ *if, and only 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* $\Pi'$ *and optimisation LVAs* $A'$ *and* $B'$ *for* $\Pi'$, $A'$ probabilistically dominates $B'$ on $\pi'$ for solution quality less than or equal to $q$ *if, and only if,* $\forall t : P_s(RT_{A',\pi'} \leq t, SQ_{A',\pi'} \leq q) \geq P_s(RT_{B',\pi'} \leq t, SQ_{B',\pi'} \leq q)$ *and* $\exists t : P_s(RT_{B',\pi'} \leq t, SQ_{B',\pi'} \leq q) > P_s(RT_{B',\pi'} \leq t, SQ_{B',\pi'} \leq q)$.

$A'$ probabilistically dominates $B'$ on $\pi'$ *if, and only if,* $A'$ *probabilistically dominates* $B'$ *on* $\pi'$ *for arbitrary solution quality bounds* $q$.

***Remark:*** A probabilistic domination relation holds between two Las Vegas algorithms on a given problem instance if, and only if, their respective (qualified) RTDs do not cross each other. 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 performs better, that is, obtains higher solution probabilities (for a given solution quality bound), depends on the time the algorithm is allowed to run.

Statistical tests 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 qualities achieved by two SLS optimisation algorithms are identical. (The widely used $t$-test generally fulfils 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, in many cases, the $t$-test is not applicable.) 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].

One important question when assessing the statistical significance of performance differences observed between algorithms is that of sample size: How many runs should be performed for measuring the respective empirical RTDs? Generally, the precision of statistical tests, that is, their ability to correctly distinguish situations in which the given null hypothesis is correct from those where it is incorrect, crucially depends on sample size. Table 4.2 shows the performance differences between two given RTDs that can be detected by the Mann-Whitney U-test for standard significance levels and power values in dependence of sample size. (Note that the significance level and power value indicate the maximum probabilities that the test incorrectly rejects or accepts the null hypothesis that the medians of the given RTDs are equal, respectively.)

| *sign. level 0.05, power 0.95* | | *sign. level 0.01, power 0.99* | |
|---|---|---|---|
| *sample size* | $m_1/m_2$ | *sample size* | $m_1/m_2$ |
| 3 010 | 1.1 | 5 565 | 1.1 |
| 1 000 | 1.18 | 1 000 | 1.24 |
| 122 | 1.5 | 225 | 1.5 |
| 100 | 1.6 | 100 | 1.8 |
| 32 | 2 | 58 | 2 |
| 10 | 3 | 10 | 3.9 |

**Table 4.2** Performance differences detectable by the Mann-Whitney U-test for various sample sizes (runs per RTD); $m_1/m_2$ denotes the ratio between the medians of the two given RTDs. (The values in this table have been obtained using a standard procedure based on adjusting the statistical power of the two-sample $t$-test to the Mann-Whitney U-test using a worst-case Pitman asymptotic relative efficiency (ARE) value of 0.864.)

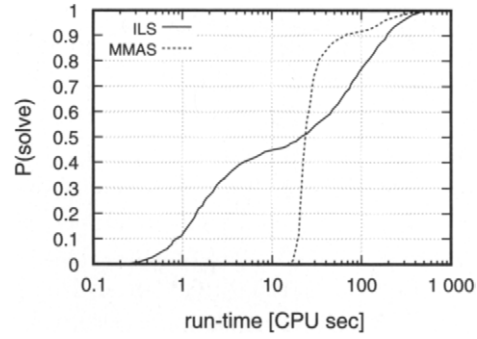



***Figure* 4.9** Qualified RTDs for two SLS algorithms for the TSP that, applied to a standard benchmark instance, are required to find a solution of optimal quality. The two RTDs cross over between 20 and 30 CPU seconds.

EXAMPLE 4.8 **Comparative RTD Analysis**

Figure 4.9 shows the qualified RTDs for two SLS algorithms for the TSP, $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) and Iterated Local Search (ILS) under the requirement of finding a solution of optimal quality for TSPLIB instance `lin318` with 318 vertices, each RTD is based on 1 000 runs of the respective algorithm. Although the Mann-Whitney U-test rejects the null hypothesis that the medians of the two RTDs are equal at a significance level $\alpha = 0.05$ (the p-value is $6.4 \cdot 10^{-5}$), taking into consideration the sample size of 1 000 runs per RTD, the difference between the medians is slightly too small to be considered significant at a power of 0.8. On the other hand, the significance of the obvious differences between the two distributions is confirmed by the Kolmogorov-Smirnov test, which rejects the null hypothesis that the observed run-times for the two algorithms stem from the same distribution at a significance level of $\alpha = 0.05$ (the p-value is $\leq 2.2 \cdot 10^{-16}$).

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 $\mathcal{MM}$AS for shorter runs, whereas $\mathcal{MM}$AS is more effective for longer runs. Both algorithms eventually find optimal solutions in all runs and hence do not show any evidence for essentially incomplete behaviour on this problem instance. Interestingly, it appears that $\mathcal{MM}$AS has practically no chance of finding an optimal solution in less than 10 CPU seconds, while ILS finds optimal solutions with a small probability after only 0.2 CPU seconds. (This salient difference in performance is partly explained by the fact that population-based

algorithms such as $\mathcal{MM}$AS typically incur a certain overhead from maintaining multiple candidate solutions.)

## Comparative Analysis for Ensembles of Instances

As previously mentioned, 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 be used to partition a given problem ensemble into three subsets: (i) those on which $A$ probabilistically dominates $B$, (ii) those on which $B$ probabilistically dominates $A$ and (iii) those for which probabilistic domination is not observed, that is, for which $A$'s and $B$'s RTDs cross each other. 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 quantile, 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 to study the correlation between $A$ and $B$ w.r.t. these. For 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 simple 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 is 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 in the performance data. They are preferable over tests based on the Pearson product-moment correlation coefficient, which require 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 100 randomly generated Euclidean TSP instances (the algorithms and problem class are described in Chapter 8). The ILS algorithm has a lower median run-time than the ACO algorithm for 66 of the 100 problem instances; this performance difference is statistically significant, because the Wilcoxon matched pairs signed-rank test rejects the null hypothesis that the performance of the two

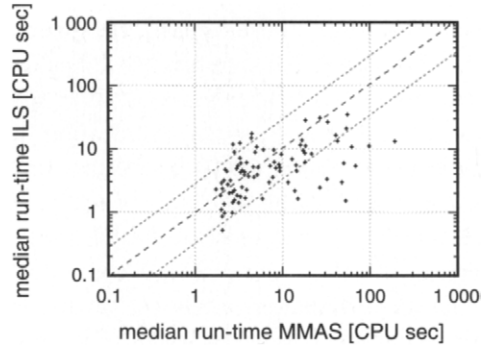



***Figure* 4.10** Correlation between median run-time required by $\mathcal{MM}$AS *vs* ILS for finding the optimal solutions to instances of a set comprising 100 TSP instances with 300 vertices each; each median was measured from 10 runs per algorithm. The band between the two dashed grey lines indicates performance differences that, based on the sample size of the underlying RTDs, cannot be assumed to be statistically significant.

algorithms is equal at a significance level of $\alpha = 0.05$ (the p-value is $7 \cdot 10^{-5}$). (It may be noted that based on the sample size of 10 runs per instance that was used for the RTDs underlying each median value, performance differences of less than a factor of three can not be assumed to be statistically significant, which follows from a power analysis of the Mann-Whitney U-test that is used for assessing such performance differences, when using $\alpha = 0.05$ and a power of 0.95.)

The median run-times required for finding optimal solutions show a significant correlation (the correlation coefficient is equal to 0.39 and Spearman's rank order test rejects the null hypothesis that the performance for the two algorithms is uncorrelated at significance level $\alpha = 0.05$; the p-value is $9 \cdot 10^{-11}$), 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).

## 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 algorithm's 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', that is, the use of unevenly optimised parameter settings, can bring about 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 whose

effects are typically not independent from each other, or when dealing with complex parameters, such as the temperature schedule for Simulated Annealing, for which the domain of possible settings are extremely large and complex. In such cases, it can be infeasible to obtain reasonable approximations of optimal parameter settings; in the context of comparative studies, this situation should then be clearly acknowledged and approximately 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 automated parameter tuning approaches that are based on techniques from experimental design [Xu et al., 1998; Coy et al., 2001; Birattari et al., 2002].

In practice, optimal parameter settings are often not known *a priori*; furthermore, optimal parameter settings for a given algorithm 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 have been collected for determining optimal parameter settings.

A more general notion of robustness of an LVA's behaviour additionally covers other types of performance variation, such as the variation in run-time 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 RTD s. 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 has been 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 respective time 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 [Hoos, 1999a]. 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 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 .

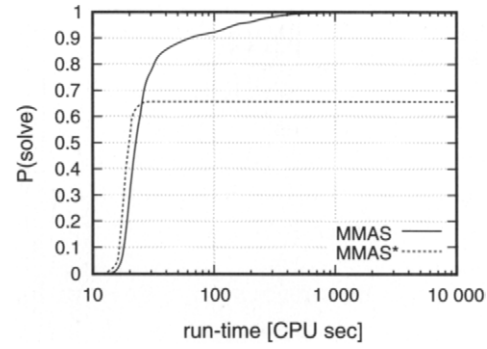



***Figure* 4.11** Qualified RTDs for two SLS algorithms for the TSP that are required to find an optimal solution of a well-known benchmark instance; $\mathcal{MM}$AS is provably PAC, whereas $\mathcal{MM}$AS* is an essentially incomplete variant of the same algorithm (see text for details). Each RTD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

EXAMPLE 4.10 **Asymptotic Behaviour in Empirical RTDs**

Figure 4.11 shows the qualified RTDs for two variants of an ACO algorithm required to find an optimal solution for TSPLIB instance `lin318` with 318 vertices. 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, Section 8.4).

In practice, true asymptotic behaviour (such as probabilistic approximate completeness) is less relevant than 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 static 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 (up to discretisation effects), static 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 that the given algorithm fails to find an existing solution of a given problem instance within a given amount of time. When augmenting any LVA with a static restart mechanism, the resulting algorithm will show RTDs with exponentially decaying right tails. Based on this observation, efficiency and stagnation can be measured by comparing the decay rate of the failure probability at time $t$, denoted $\lambda(t)$, with the tail decay rate obtained when using static restarts with cutoff $t$, denoted $\lambda^*(t)$. This leads to the following definition:

DEFINITION 4.9 **LVA Efficiency and Stagnation**

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

*Then we define $\lambda_{A,\pi}(t) := -d/dt[\ln(1-rtd_{A,\pi})](t) = 1/(1-rtd_{A,\pi}(t)) \cdot d/dt[rtd_{A,\pi}](t)$, where $d/dt[f]$ denotes the first derivative of a function $f$ in $t$. Furthermore, we define $\lambda^*_{A,\pi}(t) := -\ln(1 - rtd_{A,\pi}(t))/t$.*

The efficiency of $A$ on $\pi$ at time $t$ *is then defined as* $eff_{A,\pi}(t) := \lambda_{A,\pi}(t)/\lambda^*_{A,\pi}(t)$. *Similarly, the* stagnation ratio of $A$ on $\pi$ at time $t$ *is defined as* $stagr_{A,\pi}(t) := 1/eff_{A,\pi}(t)$, *and the* stagnation of $A$ on $\pi$ at time $t$ *is given by* $stag_{A,\pi}(t) := \ln(stagr_{A,\pi}(t))$.

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

***Remark:*** For empirical RTDs, the decay rates $\lambda_{A,\pi}(t)$ are approximated using standard techniques for numerical differentiation of discrete data such that artifacts due to discretisation effects are avoided as much as possible.

It is easy to see that according to the definition, for any essentially incomplete algorithm $A$ there are problem instances on which the minimal efficiency of

$A$ is zero. Constant minimal 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 Uninformed Random Picking as introduced in Chapter 1, 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 Uninformed 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. Instead, it provides a quantitative measure for relative changes in efficiency of a given LVA over the course of its run-time. (However, the definition can easily be extended such that an absolute performance measure is obtained; this is done by using the restart decay rate $\lambda^*$ over a set of algorithms instead of $\lambda^*_{A,\pi}$ in the definition of LVA efficiency.)

## 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 that 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., if 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 [Marquardt, 1963] or the expectation maximisation (EM) algorithm [Dempster et al., 1977]. The quality of the approximation thus obtained can be assessed using standard statistical goodness-of-fit tests, such as the well-known $\chi^2$-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 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 shown in Figure 4.6 (page 172), one might notice that the 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 $ed[m](x) := 1 - e^{x/m}$, 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).

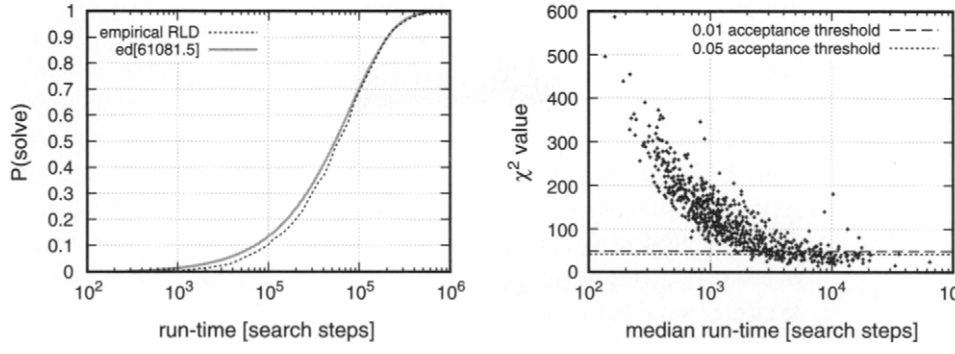



***Figure 4.12*** *Left:* Best-fit approximation of the RLD from Figure 4.6 (page 172) by an exponential distribution; this approximation passes the $\chi^2$ goodness-of-fit test at significance level $\alpha = 0.05$. *Right:* Correlation between median run-length and $\chi^2$ values from testing RLDs of individual instances versus a best-fit exponential distribution for a test-set of 1 000 hard Random 3-SAT instances; the horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance levels of the $\chi^2$-test.

Then, we applied the $\chi^2$ goodness-of-fit test to examine the hypothesis whether the resulting exponential distribution is identical to the theoretical RTD underlying the empirically observed run-lengths. In the given example, the resulting $\chi^2$ 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) 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 detailed — 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, that is, instances that 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, it becomes necessary to automate this procedure, and to analyse and summarise its results 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 one to falsify such hypotheses or to collect arbitrary amounts of evidence for their validity.

EXAMPLE 4.12 **Functional RTD Approximation 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 $\chi^2$ values as outlined above, we obtained the result shown on the right side of Figure 4.12 (page 189), which shows the median values of the RLDs plotted against the corresponding $\chi^2$ values: Although, for most instances, the distribution hypothesis is rejected, we observe a clear correlation between the solution cost of the instances and the $\chi^2$ 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; 2000a].)

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 (in particular, problem size). Furthermore, it facilitates comparative studies of the behaviour of two or more LVA algorithms. In all of 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 the scaling of an algorithm's run-time behaviour with problem size, knowledge of 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 in the following, 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 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 $\pi$. If for all run-times $t$, the efficiency of $A$ on $\pi$ at time $t$, $eff_{A,\pi}(t)$, 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 at $t$ 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 4.13). Furthermore, if, and only if, $eff_{A,\pi}(t) = 1$ 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 if, and only if, the RTD of $A$ on $\pi$ is an exponential distribution. Finally, if there exists a run-time $t'$ such that $eff_{A,\pi}(t) \leq 1$ for all $t > t'$, then restarting the algorithm at time $t$ will lead to an increased solution probability for some run-time $t'' > t$. This is equivalent to the condition that from $t'$ on the cumulative RTD graph of $A$ on $\pi$ is less steep for any time $t > t'$ 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 $\pi$, and

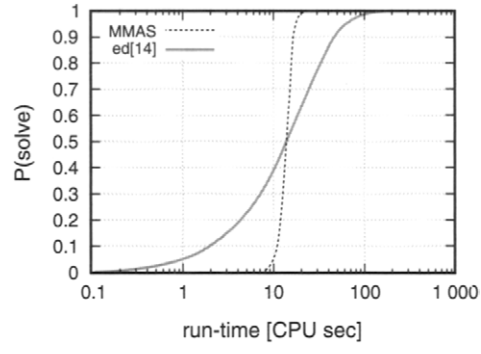



***Figure* 4.13** Qualified RTD of an ACO algorithm for TSP ($\mathcal{MM}$AS) on TSPLIB instance `lin318` with 318 vertices, based on 1 000 independent runs, and exponential distribution with identical median. The fact that this RTD is consistently steeper than an exponential indicates that restart with any fixed cutoff time will lead to performance loss.

the minimal $t$ for which $ed[m^*](t) = rtd_{A,\pi}(t)$. Formally, this is achieved using the following definitions:

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

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

where $rtd_{A,\pi}(t)$ is the theoretical run-time distribution of $A$ on $\pi$, and $A$ is incomplete, that is, $P_s(RT \leq t) < 1$ for any finite run-time $t$ (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 $ed[m^*](t) = rtd_{A,\pi}(t)$, that is, 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 they could only arise if $A$ would solve $\pi$ with probability larger than zero for infinitesimally small run-times.

Equations 4.2 and 4.3 apply to theoretical as well as to 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 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

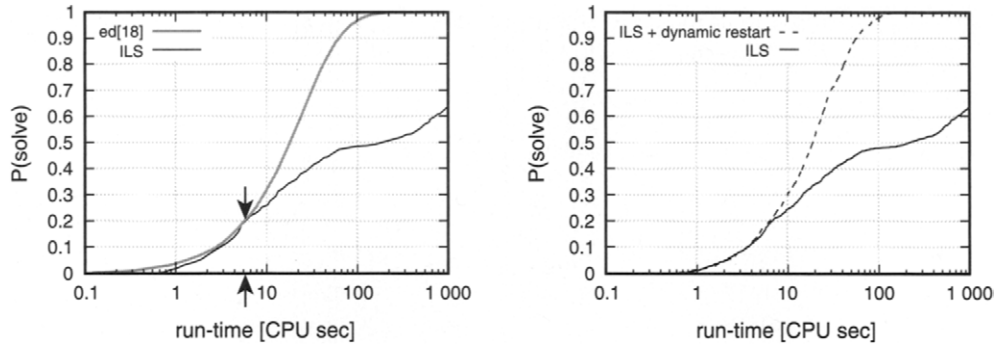



***Figure* 4.14** Qualified RTD for an ILS algorithm required to find optimal solutions for TSPLIB instance `pcb442`; note the stagnation behaviour apparent from the RTD graph. *Left:* Optimal cutoff time for static restarts, $t_{opt}$, and corresponding exponential distribution $ed[m^*]$. *Right:* Effect of dynamic restart strategy. (Details are given in the text.)

$n = 442$ vertices. 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 observed 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 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 current incumbent candidate solution was found; if this time interval exceeds a threshold $\theta$, a restart is performed. (In this scheme, incumbent candidate solutions are not carried over restarts of the search.) The time threshold $\theta$ is typically measured in search steps; $\theta$ 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 solution 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 algorithm 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, page 396.)

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* [Hoos, 1999a]. In terms of the GLSM models 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 has been 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 important components of advanced SLS methods, 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 algorithm 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 median $m$, then the probability of finding a solution in at least one of $p$ independent runs of time $t$ each is exponentially distributed with median $m/p$. Consequently, if we run such an algorithm once for time $t$, we obtain 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 previously, 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 probability $p_s$, 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 $p$ processors 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 accumulated parallel run-time $t_{max}$ can be easily determined. (The accumulated parallel run-time is the total run-time over all processors.) It is interesting to note that for higher success probabilities, the maximal number of processors for which optimal parallelisation can be achieved is typically also higher.

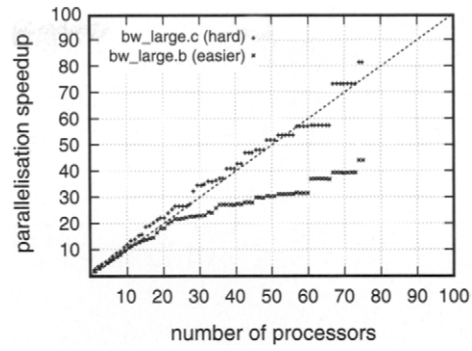



***Figure* 4.15** Speedup achieved by multiple independent runs parallelisation of a high-performing SLS algorithm for SAT applied to two SAT-encoded instances of a hard planning problem. The diagonal line indicates optimal parallelisation speedup. Note that for the easier instance, the parallelisation speedup is increasingly suboptimal for more than 10 processors. (For details, see text.)

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 using Equation 4.4) for a high-performance 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-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 fewer than ten runs. A desired success probability of $p_s = 0.95$ was used for determining the sequential and parallel run-times.

Instance `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 more than 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 multiple instruction / multiple data (MIMD) machines with thousands of processors. The resulting parallel SLS algorithms are precisely captured by the homogeneous co-operative GLSM

model without communication introduced in Chapter 3. They are of particular interest in the context of SLS applications to time-critical tasks (such as robot control or on-line scheduling), as well as to 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 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 published by Luby and Ertel [1994]. The application scenarios for Las Vegas algorithms and norms of LVA behaviour covered here have been introduced by 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 studying SLS algorithms was first proposed by 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 [Aiex et al., 2002; Hoos and Stützle, 2000a; Hoos and Boutilier, 2000; Stützle and Hoos, 2001; Stützle, 1999; Tulpan et al., 2003].

There is some related work on the use of search cost distributions over instance ensembles for the empirical analysis of complete search algorithms. Kwan showed that for different types of random CSP instances, the search cost distributions for several 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 instances of the Quasigroup Completion

Problem, a special type of CSP, in the context of algorithm portfolios design [Gomes and Selman, 1997a]. Interestingly, the corresponding RTDs for the randomised systematic search algorithms they studied can (at least in some cases) 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, 1998; Hoos and Stützle, 1999; Hoos, 1999a] or mixtures of exponentials [Hoos, 2002b]. 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 [Dannenbring, 1977; 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., 2001; Birattari et al., 2002].

Various general aspects of empirical algorithms research are covered in a number of publications. 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., 1979; 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 studies 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]. Gent et al. give a similar, but more limited, overview of potential problems in the experimental analysis of algorithms [Gent et al., 1997].

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 intelligence 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 [Cohen, 1995]. For an additional introduction to statistical methods we also recommend the book by Papoulis [1991]. The handbook by 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] and Siegel et al. [1988]. Furthermore, for general techniques of experimental design and the analysis of experimental data we refer to the work of Dean and Voss [2000] and 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 (RTDs)* provide a good basis for empiricially analysing the behaviour of SLS algorithms and more generally, members of the broader class of *(generalised) 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 (RTDs)*, *qualified run-time distributions (QRTDs)* and *solution quality distributions (SQDs)*, as well as *time-dependent solution quality statistics (SQTs)* and *solution quality dependent run-time statistics (RTQs)*. Empirical RTDs can be easily obtained from the same data required 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 size. 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 *static 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, for example, by suggesting simplified experimental designs in which only the parameter values of a functionally

characterised family of RTDs are analysed instead of the entire distributions. Applied to SLS algorithms, this approach can also reveal fundamental properties of the algorithm and provide deeper insights into its behaviour.

Results from the empirical analysis of an SLS algorithm can provide significant leverage for further improvement of its performance. We gave an overview of various approaches to achieving such improvements, 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.

## Exercises

4.1 [***Easy***] Give three examples for (generalised) Las Vegas algorithms and identify all stochastic elements in these.

4.2 [***Easy***] Describe a concrete application domain where the utility of a solution to a given problem instance changes over time.

4.3 [***Medium***] Prove that Uninformed Random Picking (see Chapter 1, Section 1.5) has the PAC property.

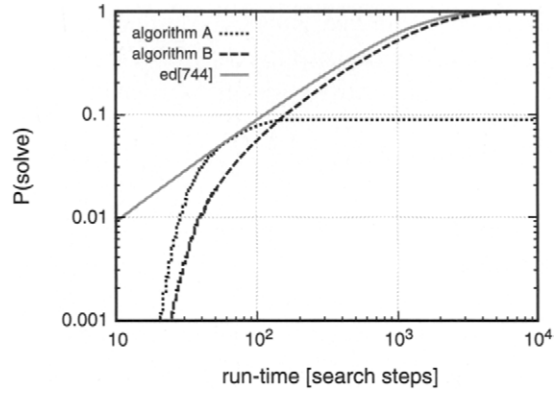
4.4 [***Easy***] Explain the difference between a run-time distribution (RTD), a solution quality distribution (SQD) and a search cost distribution (SCD).

4.5 [***Medium***] In order to investigate the behaviour of an SLS algorithm for a combinatorial optimisation problem on a given problem instance, solution quality traces over $m$ independent runs are recorded. In each of these runs, the known optimal solution quality for the given instance is reached. Explain how qualified run-time distributions (RTDs) for various solution quality bounds and solution quality distributions (SQDs) for various run-time bounds can be obtained from these solution quality traces.

4.6 [***Easy; Hands-On***] Study the behaviour of a simple iterated local search algorithm for the TSP (available from `www.sls-book.net`) on TSPLIB instance

`lin318` (available from TSPLIB [Reinelt, 2003]). In particular, report and compare the solution quality-distributions (SQDs) for increasingly high run-time bounds. (The provably optimal solution quality for this instance is 42 029.) Describe how the SQDs change with the run-time bounds and explain the reasons underlying this phenomenon.

4.7 [***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, these algorithms were found to exhibit the RTDs shown below.




What do you learn from these RTDs? Which further experiments do you suggest to decide which algorithm is superior?

4.8 [***Medium***] Explain why it is desirable to mathematically model observed RTDs using functional approximations. Do the approximations have to be perfect to be useful?

4.9 [***Medium***] What happens if the equations used for determining optimal restart times according to Equations 4.2 and 4.3 (page 193) are applied to a complete Las Vegas algorithm?

4.10 [***Hard***] Outline an empirical approach to run and decide a competition on the best SLS algorithm for the TSP. Discuss all relevant aspects of the competition (selection of problem instances, performance measures, experimental protocol) and justify your approach.