# Chapter 2

# Empirical Analysis of Las Vegas Algorithms

In this chapter, we introduce a novel methodology for empirically investigating the behaviour of stochastic local search algorithms. We use this methodology later for the empirical analysis of SLS-Algorithms for SAT; here, it is developed in the more general context of Las Vegas Algorithms. After motivating the need for a more adequate empirical methodology and providing some general background on Las Vegas Algorithms, we introduce our new method which is based on characterising run-time distributions of algorithms on single problem instances. As we will show, our novel approach facilitates the evaluation and comparison of Las Vegas Algorithms; specifically, it can be used for obtaining optimal parameterisations and parallelisations. We then point out some pitfalls which arise by using inadequate empirical methodology and show how these are avoided when using our approach. Finally, we extend our methodology to optimisation problems. The chapter concludes with an overview of related work and a short resume of the main ideas and results.

## 2.1 Motivation and Background

*Las Vegas Algorithms (LVAs)* are nondeterministic algorithms for which, if a solution is found, its correctness is guaranteed. However, it is not guaranteed that for a soluble problem instance, such an algorithm eventually finds a solution. Because of its nondeterministic nature, the run-time of a Las Vegas Algorithm is a random variable.

Las Vegas Algorithms are prominent not only in the field of Artificial Intelligence but also in other areas of computer science and Operations Research. Because of their inherent randomness, stochastic local search (SLS) algorithms (*cf.* Chapter 1) are particular instances of LVAs. In the recent years, SLS algorithms have become quite prominent for solving both NP-complete decision problems and NP-hard combinatorial problems. These algorithms, such as Tabu Search [Glo89, Glo90, HJ90], WalkSAT [SKC94], the Min-Conflicts Heuristic [MJPL92], Simulated Annealing [KJV83], Genetic Algorithms [Hol75, Gol89], Evolution Strategies [Rec73, Sch81], Ant Colony Optimisation algorithms [DMC96], *etc.*, have been found to be very successful on numerous problems from a broad range of domains. But also a number of systematic search methods, like some modern variants of the Davis Putnam algorithm for propositional satisfiability (SAT) problems, make use of non-deterministic decisions (like randomised tie-breaking rules) and can thus be characterised as Las Vegas Algorithms.

However, due to their non-deterministic nature, the behaviour of Las Vegas Algorithms is usually difficult to analyse. Even in the cases where theoretical results do exist, their practical applicability is often very limited. This is, for example, the case for Simulated Annealing, which is proven to converge towards an optimal solution under certain conditions which, however, cannot be met in practice. On the other hand, theoretical results for algorithms which could be shown to be very effective in practice are usually very limited, as is the case for some of the most successful variants of tabu search. Given this situation, in most cases analyses of the run-time behaviour of Las Vegas Algorithms are based on empirical methodology. In a sense, despite dealing with algorithms which are completely known and can be easily understood on a step-by-step execution basis, computer scientists are in a sense in the same situation here as, say, an experimental physicist observing some non-deterministic quantum phenomenon or a microbiologist investigating bacterial growth phenomena.

The methods that have been applied for the analysis of Las Vegas (and particularly SLS) algorithms in AI, however, are rather simplistic. In most cases, the performance of the algorithms being tested is characterised by measuring basic statistics over a number of runs on a given problem instance, like the mean, median and/or standard deviation of the run-time. Run-times are often reported as CPU-times or, sometimes, using basic step counts for a particular algorithm scheme [AO96], rarely a combination of both. Quite often, the algorithms are evaluated on a set of problem instances drawn from some randomised problem distribution, like Random-3-SAT [MSL92, CA96], randomly generated constraint satisfaction problems (CSPs) [Smi94], or randomly generated graph colouring problems [MJPL92, HW94b]. Quite frequently, when comparing algorithms on such problem classes, the sets are generated using a given generator algorithm instead of being fixed, widely used test sets.

At the first glance, all these methods seem to be quite admissible, especially since the results are usually consistent in a certain sense. In the case of SLS algorithms for SAT, for instance, advanced algorithms like WalkSAT [SKC94] usually outperform older algorithms (such as GSAT [SLM92]) on a large number of problems from both randomised distributions and structured domains. The claims which are supported by empirical evidence are usually quite simple (like "algorithm A outperforms algorithm B"), and the basic methodology, as described above, is both easy to apply and powerful enough to get the desired results.

Or is it really? Recently, there has been some severe criticism regarding the empirical testing of algorithms [Hoo94, Hoo96a, McG96]. It has been pointed out that the empirical methodology that is used to evaluate and compare algorithms does not reflect the standards which have been established in other empirical sciences for quite a time. Also, it was argued that the empirical analysis of algorithms should not remain at the stage of collecting data, but should rather attempt to formulate hypotheses based on this data which, in turn, can be experimentally verified or refuted. Up to now, most work dealing with the empirical analysis of Las Vegas algorithms in AI has not lived up to these demands. Instead, recent studies still use basically the same methods that have been around for years, often investing tremendous computational effort in doing large scale experiments [PW96] in order to ensure that the basic descriptive statistics thus collected are sufficiently stable. At the same time more fundamental issues, such as the question whether the particular type of statistical analysis that is done (usually estimating means and standard deviations) is adequate for the type of evaluation that is intended, are often somewhat neglected or not addressed at all. Therefore, a more adequate methodology for the empirical analysis of Las Vegas algorithms is needed as a basis for application, investigation, and further development.

### 2.1.1 Las Vegas Algorithms

Formally, an algorithm $A$ is a *Las Vegas algorithm (LVA)* if it has the following properties:

- If for a given problem instance $\pi$, algorithm $A$ returns a solution $s$, $s$ is guaranteed to be a valid solution of $\pi$.
- On each given instance $\pi$, the run-time of $A$ is a random variable $RT_A$.

According to this definition, Las Vegas algorithms are always correct, while they are not necessarily complete, *i.e.*, even if a given problem instance has a solution, a Las Vegas algorithm is generally not guaranteed to find it. Since completeness is an important theoretical concept for the study of algorithms, we classify Las Vegas algorithms into the following three catogories:

- *complete Las Vegas algorithms* are those, which can be guaranteed to solve each problem instance for which a solution exists within run-time $t_{max}$, where $t_{max}$ is an instance-dependent constant. Let $P_s(RT_{A,\pi} \leq t)$ denote the probability that $A$ finds a solution for a soluble instance $\pi$ in time $\leq t$, then $A$ is complete exactly if for each $\pi$ there exists some $t_{max}$ such that $P_s(RT_{A,\pi} \leq t_{max}) = 1$.
- *approximately complete Las Vegas algorithms* solve each problem instance for which a solution exists with a probability converging to 1 as the run-time approaches $\infty$. Thus, $A$ is approximately complete, if for each soluble instance $\pi$, $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) = 1$.
- *essentially incomplete Las Vegas algorithms* are Las Vegas algorithms which are not approximately complete (and therefore also not complete), *i.e.*, for which there exists a soluble instance $\pi$, for which $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) < 1$.

Examples for complete Las Vegas algorithms in AI are randomised systematic search methods like Satz-Rand [GSK98]. Many of the most prominent stochastic local search methods, like Simulated Annealing or GSAT with Random Walk, are approximately complete, while others, such as basic GSAT, the Min-Conflicts Heuristic, and most variants of Tabu Search are essentially incomplete.

In literature, approximate completeness is often also referred to as *convergence*. Convergence results are established for a number of SLS algorithms, such as Simulated Annealing [KJV83], or Genetic Algorithms [Bäc94]; another class of complete Las Vegas algorithms which is less well known in AI, are certain learning automata schemes [NT89]. Interestingly, completeness can be enforced for most SLS algorithms by providing a restart mechanism, as can be found in GSAT [SLM92], and selecting the starting configurations in a systematic way. However, both forms of completeness are mainly of theoretical interests, since the time limits for finding solutions are far too large to be of practical use. For SLS algorithms, essential incompleteness is usually caused by the algorithm getting trapped in local minima of the search space they are operating on. Modern SLS algorithms provide methods, like restart, random walk, probabilistic tabu-lists, etc., to escape from these local minima and thus achieve approximate completeness.

Since most relevant Las Vegas algorithms are either complete or approximately complete, or can at least be easily modified to achieve (approximate) completeness as indicated above, we assume for the following that the Las Vegas algorithms we are dealing with are at least approximately complete unless explicitly mentioned otherwise. Consequently, we know that each run of the algorithm terminates after some time, or, in other words, the algorithm never gets stuck.

### 2.1.2 Application Scenarios

Before even starting to evaluate any algorithm, it is crucial to find the right evaluation criteria. Especially for Las Vegas algorithms, since their run-time is non-deterministic, there are fundamentally different criteria for evaluating their run-time behaviour, depending on the characteristics of the environment they are supposed to work in. Thus, we classify possible application scenarios in the following way:

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

**Type 2:** There is a time limit for finding the solution such that the algorithm has to provide a solution after a time of $t_{max}$, while solutions which are found later are of no use. In real-time applications, like robotic control or dynamic scheduling tasks, $t_{max}$ 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 a time of $t_{max}$ a solution has been found.

**Type 3** The usefulness or utility of a solution depends on the time which 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 after time $t$. As can be easily seen, types 1 and 2 are special cases of type 3 which can be characterised by utility functions which 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).

Based on these application types, we now discuss the criteria for evaluating the performance of Las Vegas algorithms in each scenario, respectively. While in the case of no time limits being given (type 1), the mean run-time might suffice to roughly characterise the run-time behaviour, in real-time situations (type 2) it is basically meaningless. An adequate criterion for type 2 situation with time-limit $t_{max}$ is $P(RT \leq t_{max})$, the probability of finding a solution within the given time-limit. For type 3, the most general scenario, the run-time behaviour can only be adequately characterised by the run-time distribution function $rtd : \mathbb{R} \mapsto [0,1]$ defined as $rtd(t) = P(RT \leq t)$ or some approximation of it. The run-time distribution (RTD) specified by this distribution function completely and uniquely characterises the run-time behaviour of a Las Vegas algorithm. Therefore, given this information, other criteria, like the mean run-time, its standard deviation, median, percentiles, or success-probabilities $P(RT \leq t')$ for arbitrary time-limits $t'$ can be easily obtained.

In some sense, type 3 is not only the most general class of application scenarios, but these scenarios are also the most realistic. The reason for this is the fact that real-world problem solving usually involves time-constraints which are less strict than in type 2. Instead, at least within a certain interval, the value of a solution gradually decreases over time. In particular, this situation is given when taking into account the costs (like CPU time) for finding a solution. As an example, consider a situation where hard combinatorial problems have to be solved online, using extremely expensive hardware in a time-sharing mode. Even if the immediate benefit of finding a solution is invariant over time, the costs for performing the computations will diminish the final payoff. Two common ways of modelling this effect are constant or proportional discounting, *i.e.*, to use utility functions of the form $U(t) = \max\{u_0 - ct, 1\}$ or $U(t) = e^{-\lambda t}$, respectively [PMG98]. Based on the utility function, the weighted solution probability $U(t) \cdot P(RT \leq t)$ can be used as a performance criterion. If $U(t)$ and $P(RT \leq t)$ are given, optimal cutoff times $t^*$ which 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 underlying run-time distributions.

## 2.2 RTDs and RLDs

As we have argued in the previous section, characterising their run-time distributions is a good basis for empirically studying the behaviour of LVAs. In practice, the empirical RTDs are determined by running the respective algorithm $A$ for $k$ times on a given problem instance up to some (very high) cutoff time and recording for each successful

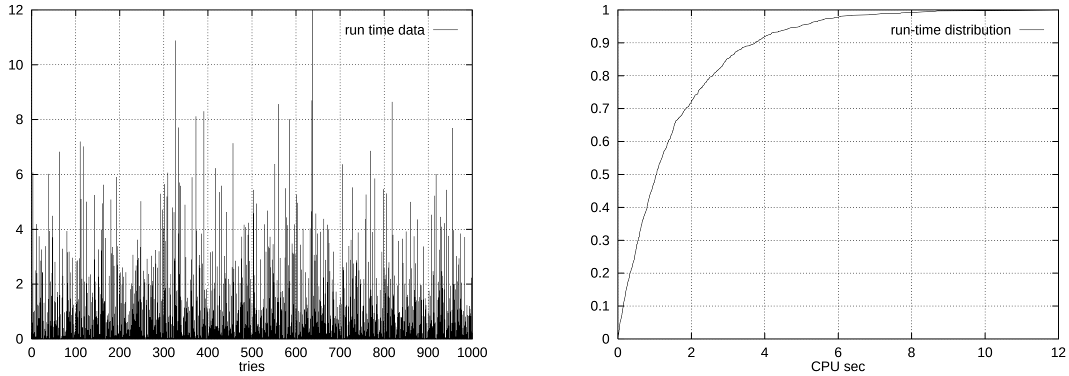


Figure 2.1: Run-time data of WalkSAT, applied to a hard Random-3-SAT instance for approx. optimal noise setting, 1,000 tries; *left:* bar diagramm of $rt(i)$, the vertical axis indicates the CPU-time; *right:* corresponding RTD; the run-times are measured in CPU-seconds on a Sun SPARC 20 machine.

run the time required to find a solution. For the empirical studies presented in this work, we chose $k$ between 200 and 1,000 in order to get reasonable estimates. The empirical run-time distribution is the cumulative distribution associated with these observations. More formally, let $RT(j)$ denote the run-time for the $j$th successful run, the cumulative empirical RTD is defined by $\widehat{P}(RT \leq t) = \#\{j | RT(j) \leq t\}/k$. For essentially incomplete algorithms, we report the success-rate $k'/k$, where $k'$ is the number of successful runs. For sufficiently high run-times, this approximates the asymptotic maximal success probability of $A$ on the given problem instance $\pi$, which is formally defined as $\widehat{p_s}^* := \lim_{t \to \infty} P_s(RT_{A,\pi} \leq t)$. Furthermore, we filter out the unsuccessful runs and characterise the behaviour of $A$ in the successful cases using the conditional empirical RTD defined by $\widehat{P}_c(RT \leq t) = \#\{j | \text{ run } j \text{ successful } \wedge RT(j) \leq t\}/k'$.

Figure 2.1 (left) shows the raw data from running a state-of-the-art SLS algorithm for SAT on a hard problem instance with 100 variables; each vertical line represents one try of the algorithm and the height of the lines corresponds to the CPU time needed. The right side of the same figure shows the corresponding RTD as a cumulative probability distribution curve $(t, \widehat{P}_c(RT \leq t))$; note that an extreme variability in run-time can be observed. We will see later that this is typical for SLS algorithms when applied to hard combinatorial problems like SAT.

### 2.2.1 CPU-times *vs* Operation Counts

Often, run-times are measured and reported as CPU-time for some concrete implementation and run-time environment (machine and operating system). But it has been argued that it is more advisable, especially in the context of comparative studies, to measure run-times using operation counts [AO96]. This way, an abstraction from the influence of the implementation and run-time environment is achieved, which facilitates comparing empirical

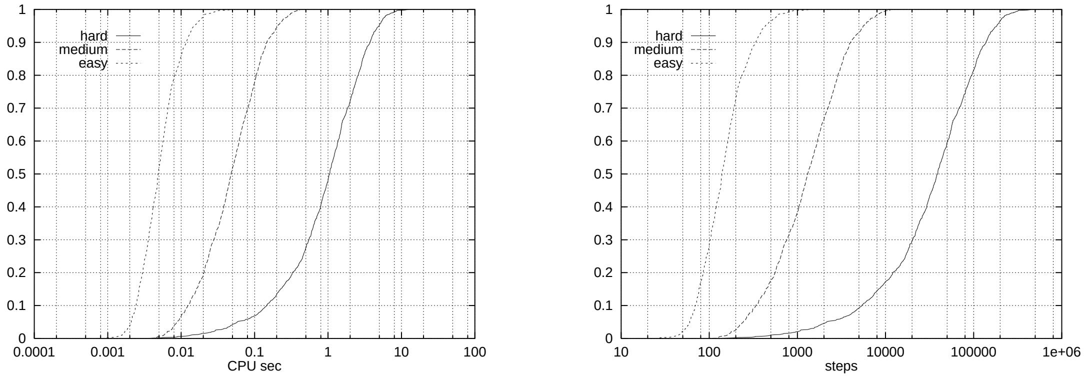


Figure 2.2: RTDs *(left)* and RLDs *(right)* for WalkSAT, applied to three Random-3-SAT instances with varying difficulty, using approx. optimal noise setting, 1,000 tries.

results across various platforms. At the same time, by reporting typical CPU-times for the relevant operations, concrete run-times can be easily calculated for a given implementation and run-time environment.

To make a clear distinction between actual CPU-times and abstract run-times measured in operation counts by referring to the latter as *run-lengths*. If the CPU-time per operation is roughly constant (as is the case for most SLS algorithms), run-times measured in CPU-time and run-lengths measured in basic operations are related to each other by scaling with a constant factor. Otherwise, to obtain a reasonable characterisation of the run-time behaviour, a more complex cost-model has to be used for weighting the different types of operations (or steps) which comprise the run-length. In the case of SLS algorithms for SAT and CSP, it is very natural and quite common to use local search steps as basic operations. Thus, run-times are measured in terms of local search steps instead of absolute CPU-times. In the following, the RTDs which are thus obtained will be called RLDs (run-length distributions).

Figure 2.2 shows RTD and RLD data for the same experiments (solving hard SAT instances with a state-of-the-art SLS algorithm). Note that, when comparing the RTDs and the corresponding RLDs in a semi-log plot, both distributions always have the same shape. This indicates that the CPU-time per step is roughly constant. However, closer examination of the example reveals that the CPU-time per step is not constant for the three instances; the reason for this is the fact that the hard problem was solved on a faster machine than the medium and easy instances. Using RLDs instead of RTDs abstracts from these machine specific aspects of experimental results.

### 2.2.2 Empirical Evaluation Based on RLDs

When analysing or comparing the behaviour of Las Vegas algorithms, the empirical RLD (or RTD) data can be used in different ways. In our experience, graphic representations

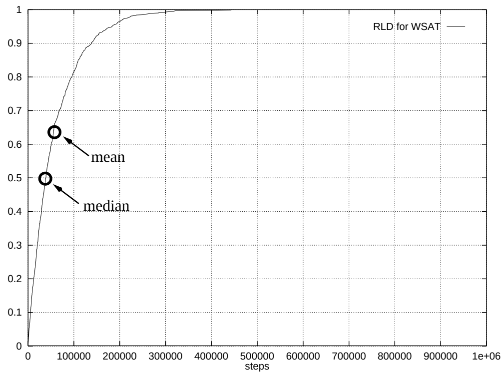


Figure 2.3: RLD for WalkSAT on a hard Random-3-SAT instance for approx. optimal noise parameter setting; median vs. mean.

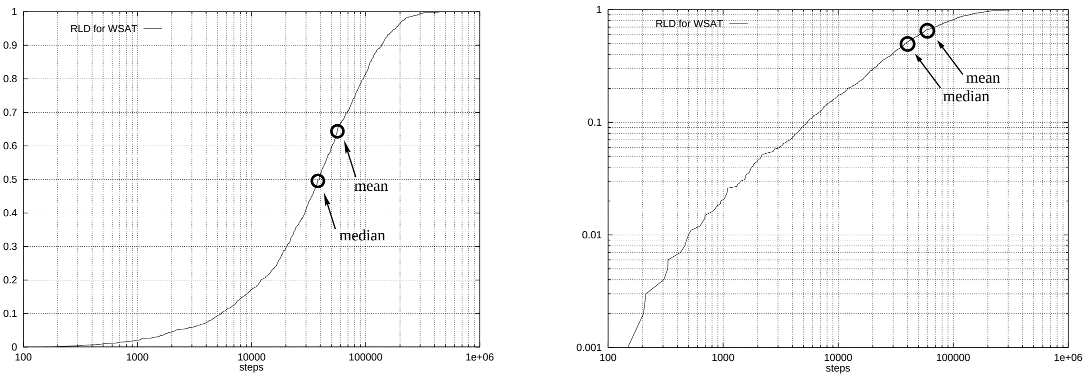


Figure 2.4: Same RLD data as above as a semi-log *(left)* and log-log *(right)* plot.

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

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

Table 2.1: Basic descriptive statistics for the RLD given in Figures 2.3 and 2.4; $Q_x$ is the $x$-percentile, while the percentile ratios $Q_x/Q_{1-x}$ are measures for the variability of the run-length data.

of empirical RLDs provide often a good starting point. As an example, Figures 2.3 and 2.4 show the RLD for the hard problem instance from Figure 2.2 in three different views. Compared to standard representations, semi-log plots (as shown in Figure 2.4, left), give a better view of the distribution over its whole range; this is especially important, since SLS algorithms show often RLDs with extreme variability. Also, when using semi-log plots to compare RLDs, uniform performance differences characterised by a constanct factor can be easily detected, as they correspond to a simple translation along the horizontal axis (for an example, see Figure 4.12, page 92, the curves for GWSAT and WalkSAT). On the other hand, when examining the behaviour of an algorithm for extremely short runs, log-log plots (as shown in Figure 2.4, right) are very useful.

While these graphical representations of RLDs are well suited for investigating and describing the qualitative behaviour of Las Vegas algorithms, quantitative analyses are usually based on the basic descriptive statistics of the RLD data. For our example, some of the most common standard descriptive statistics, like the empirical mean, standard deviation, minimum, maximum, and some percentiles, are reported in Table 2.1. Note again the huge variability of the data, as indicated by the large standard deviation and percentile ratios. The latter, like the *normalised the standard deviation* (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 RLDs.

Generally, it should be noted that for directly obtaining sufficiently stable estimates for statistics, basically the same number of test-runs have to be performed then for measuring reasonable empirical RLDs. Thus, measuring RLDs does not cause a computational overhead in data aquisition when compared to measuring only averages and empirical standard deviations. At the same time, arbitrary percentiles and other descriptive statistics can be easily calculated from the RLD data.

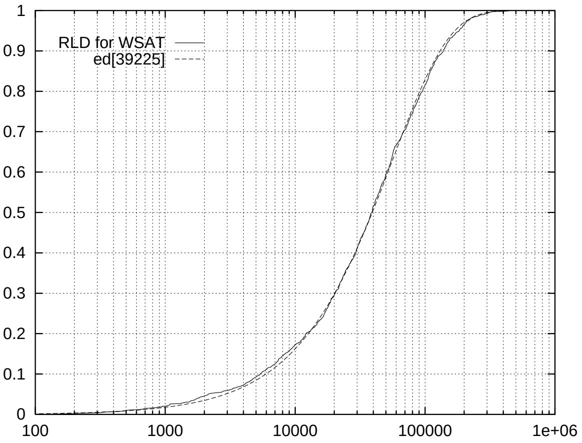


Figure 2.5: Same RLD data as in Figure 2.3 and best-fit approximation by an exponential distribution.

### Functional Characterisation of RLDs

But of course, an RLD-based approach can go considerably beyond an empirical evaluation on the basis of basic descriptive statistics. One of the particularly attractive methodological approaches is the direct characterisation of run-time distributions using functional approximations. In this work, we make strong use of continuous distribution functions and statistical goodness-of-fit tests for characterising the behaviour of Las Vegas algorithms like stochastic local search for SAT and CSP. The main reason for using continuous distribution functions for characterising an essentially discrete behaviour, especially in the context of RLDs, is that these are mathematically easier to handle: by abstracting from the discrete nature of the RLDs, a more uniform characterisation is facilitated.

Coming back to the example from Figure 2.4, one might notice that the RLD resembles 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. The validity of this hypothesis can be tested using the $\chi^2$-test, a goodness-of-fit test well-known from statistical literature. In our example, we first fit the RLD data with a cumulative exponential distribution function of the form $ed[m](x) = 1 - exp(x/m)$, using the Marquart-Levenberg algorithm (as realised in C. Gramme's Gnufit software) to determine the optimal value for the parameter $m$. This approximation is depicted in Figure 2.5. Then, the goodness-of-fit test gives a $\chi^2$ value of 26.24, which means that the approximation passed the test at a standard significance level $\alpha = 0.05$. Therefore, our distribution hypothesis has been confirmed for the given example.

Generally, the actual run-time distribution of a given Las Vegas algorithm depends on the problem instance. Thus, it should be clear that, in the first place, RLDs should be estimated and characterised on single problem instances. Often, however, one is interested in

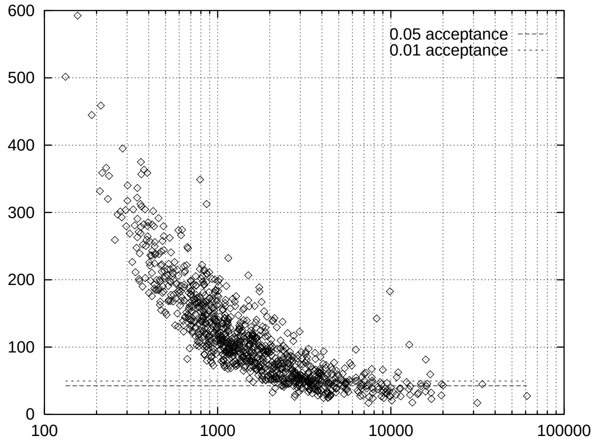


Figure 2.6: Correlation between hardness of problem instances and $\chi^2$ values from testing RLDs of individual instances versus a best-fit exponential distribution for a test-set of 1,000 Random-3-SAT instances with 100 variables and 430 clauses each. The horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance levels of the $\chi^2$-test.

an algorithm's performance on a family or distribution of problem instances. The more general type of analysis required in this situation can be easily based on the methodology used for single problem instances. We do this by formulating hypotheses on the type of RLDs which can be observed when applying a Las Vegas algorithm to sets or distributions of problem instances. These hypotheses can be empirically tested by estimating the parameters of the respective generic distribution functions for individual instances and using statistical goodness-of-fit-tests, like the well-known $\chi^2$-test [Roh76] to ensure the validity of the corresponding functional approximations.

Following this approach, it is also possible to characterise the differences between problem instances within the same distribution, or to analyse and adequately describe the scaling of LVA behaviour dependent on problem size. As we will see, such characterisations have very direct consequences for important issues like parallelisation or optimal parameterisation of Las Vegas algorithms. At the same time, they suggest novel interpretations of LVA behaviour which, as in the case of stochastic local search in this work, contributes do improving our understanding of these algorithms. Thus, by using our RLD-based methodology, an experimental approach of testing Las Vegas algorithms along the lines proposed by Hooker [Hoo96a] can be undertaken.

As an example for this kind of methodology, we extend our example from Figure 2.5. Before, we verified the hypothesis that for the given problem instance, the performance of WalkSAT, a modern SLS algorithm for SAT, could be charactised by an exponential RLD. Now, we generalise this hypothesis by assuming, that for a whole problem distribution, WalkSAT's behaviour can be characterised by exponential run-time distributions. Concretely, we test

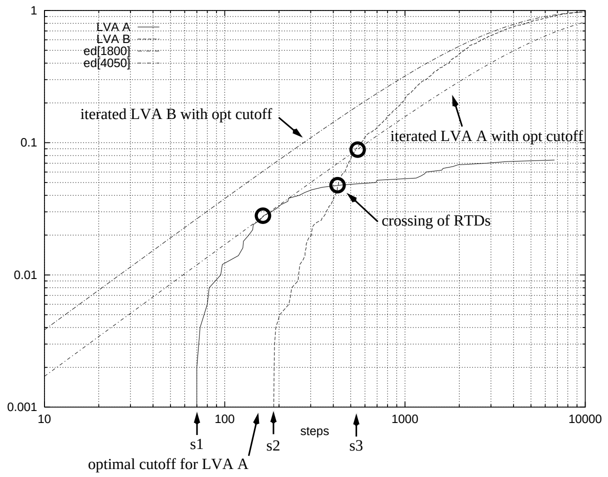


Figure 2.7: RLDs for two SLS algorithms (LVA A and LVA B) for the propositional satisfiability problem on a hard Random-3-SAT instance (the two RLDs cross over at ca. 420 steps), see text for a detailed discussion.

this hypothesis for a test-set of Random-3-SAT instances with 100 variables and 430 clauses.[1] Fitting the data for the individual instances with exponential distributions and calculating the $\chi^2$ as outlined above, we get the result shown in Figure 2.6, where we plot the median values of the RLDs against the corresponding $\chi^2$ values: although, for most instances, the distribution hypothesis is rejected, we observe a clear correlation between the hardness 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, it could be easily modified to match the results of our analysis.[2]

### Comparing LVA performance based on RLDs

RLDs also provide a good basis for comparing the behaviour and performance of Las Vegas algorithms. Comparing two Las Vegas algorithms is relatively straight-forward if a dominance relation holds between them, *i.e.*, one of them consistently gives a higher solution probability than the other. Formally, this can be captured by the concept of domination, defined in the following way: LVA A dominates LVA B if $\forall t : P(RT_A \leq t) \geq P(RT_B \leq t)$ and $\exists t : P(RT_A \leq t) > P(RT_B \leq t)$. In practice, such a dominance relation cannot always be observed, which makes the comparison substantially more difficult. This situation is characterised by the occurrence of cross-overs between the corresponding RLDs, indicating,

[1]Random-3-SAT is a syntactically restricted subclass of SAT; the distribution we use here corresponds to the phase transition for Random-3-SAT, a region where particularly difficult problem instances can be found. Details on this will be given in Chapter 4.

[2]The result indicated here will be further investigated in Chapter 4.

that which of the two algorithms gives better performance, *i.e.*, higher solution probabilities, depends on the time-limit.

As an example for this, consider the situation presented in Figure 2.7 in which RLDs for two specific SLS algorithms (LVA A and LVA B) are plotted. LVA B is approximately complete w.r.t. the given problem instance and its performance monotonically improves with increasing run-length (as can be seen from the decreasing distance between the RLD and the projected optimal exponential distribution $ed[1800]$). LVA A is essentially incomplete, the success probability converges to ca. 0.08. For very short runs we also observe "convergent behavior": Apparently for both algorithms there exists a minimal (sample size dependent) number of steps (marked $s_1$ and $s_2$ on the $x$-axis) below which the probability for finding a solution is negligible. Interestingly, both curves cross in one specific point at ca. 420 steps; *i.e.*, without using restarts, LVA A gives a higher solution probability than LVA B for shorter runs, whereas LVA B is more effective for longer runs. Yet, for LVA A an optimal cutoff value of ca. 170 steps exists. Thus, repeated execution of LVA A for a reasonably well chosen cutoff time, after which the algorithm is restarted, gives a much higher solution probability as actually observed when running the algorithm for a long time. In fact, if the optimal cutoff parameter is chosen, one more point of interest is at ca. 500 steps (marked $s_3$ on the horizontal axis): for a lower number of steps as $s_3$, using independent runs of LVA A with optimal cutoff value one improves upon the performance of LVA B, while past $s_3$ LVA B is strictly superior to LVA A. Observations like this can generally be exploited to enhance the overall performance; in particular, any-time behaviour can be achieved by combining suitably parameterised SLS algorithms into algorithm portfolios (*cf.* Chapter 5).

### Deriving Optimal Parameterisations from RLD Data

As shown in the example above, run-time distributions can be used for determining optimal parameters for Las Vegas algorithms. In particular, given the RLD data, it is relatively easy to decide whether restarting the algorithm after a fixed cutoff-time will improve performance, and if so, to derive optimal cutoff-times.

To decide wether restart should be used, we simply compare the observed RLD with an exponential distribution. As it is well-known that for exponentially distributed run-times, due to the properties of the exponential distribution [Roh76], we get the same solution probability running an algorithm once for time $t$ or $p$ times for time $t/p$. Therefore, if the observed RLD is exponential, restart will neither improve nor worsen the algorithms performance. As a consequence, in this case, we can run the individual tries in parallel on different processors and obtain an optimal speedup (this issue will be further discussed in Chapter 5).

If, however, the observed RLD is steeper than an exponential distribution, the probability of finding a solution within a given time interval increases when running the algorithm for a longer time (*cf.* LVA B in Figure 2.7). In this a case, restarting the algorithm after some fixed cutoff time would result in a loss of performance; consequently, when using multiple independent tries parallelisation, the speedup will generally be suboptimal. On the other side, if the run-time distribution of an algorithm is less steep than an exponential distribution, by restarting this algorithm its performance can be improved, as illustrated for algorithm LVA A in Figure 2.7. In this case, an optimal cutoff time $RT_{\text{opt}}$ can be identified in the following way:

$$
\begin{aligned}
m' &= \min\{m \mid \exists x > 0 : ed[m](x) - rt(x) = 0\} && (2.1)\\
RT_{\text{opt}} &= \min\{x \mid x > 0 \wedge ed[m'](x) - rt(x) = 0\} && (2.2)
\end{aligned}
$$

Generally, there are two special cases to be considered. 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, it can be shown that the optimal cutoff is either equal to zero (if the infimum is equal to zero), or it is equal to $+\infty$, *i.e.*, no restart should be used at all. Secondly, if $m'$ exists it might still not be possible to determine $RT_{\text{opt}}$, because the set in the second equation does not have a minimum. In this case, obviously there are arbitrary small common points of $ed[m']$ and $rt$, such that the optimal cutoff would be equal to zero. In practice, the cases with an optimal cutoff of zero will hardly occur, since they would only occur for algorithms which can solve a problem with some probability $> 0$ for arbitrary small run-times. The case with $RT_{\text{opt}} = \infty$ corresponds to the RLD being generally steeper than an exponential distribution.

Thus, if an optimal cutoff exists it can be easily determined from the RLD data. In this case, multiple independent tries parallelisation results in super-optimal speedup compared to the sequential case; this situation is given for many greedy local search algorithms like GSAT which easily get stuck in local minima. Generally, this demonstrates how the RLD-based methodology facilitates the analysis and improvement of Las Vegas algorithms.

## 2.3 Pitfalls of Inadequate Methodology

While in the last section, we discussed the benefits of an RLD-based empirical methodology for analysing, comparing, and improving the behaviour of Las Vegas algorithms, in this section we point out pitfalls which arise when using inadequate methodology. In particular, we identify two types of problematic methodology and show how in each of these cases, erroneous or inadequate conclusions are obtained. We give theoretical and empirical justifications for our arguments and highlight our criticism based on selected examples.

### 2.3.1 Problem Type 1: Superficial Analysis of Run-time Behaviour

In many studies, the evaluation or comparison of Las Vegas algorithms is solely based on average run-times, or averages and standard deviations, while nothing is known about the type of their run-time distributions. One problem with this method is that these statistics alone give only a very imprecise impression of the run-time behaviour. Thus, analysing and reporting only means and possibly variances is a very wasteful use of the empirical data when compared to the RLD-based approach which, as detailed before, essentially uses the same data as required for estimating stable means and standard deviations.

Before we demonstrate this, let us briefly recapitulate how the mean run-time of a Las Vegas algorithm is estimated in practice. First, the algorithm is executed $n$ times on a given problem instance with some cutoff time $RT_{\max}$. Then, if $k$ of these runs are successful and $RT_i$ is the run-time of the $i$th successful run, the expected run-time is estimated by averaging over the successful runs and accounting for the expected number of runs required to find a solution:[3]

$$\widehat{E}(RT) = \frac{1}{k}\sum_{i=1}^{k} RT_i + \frac{n-k}{k} \cdot RT_{\max} \tag{2.3}$$

Now, to see the difference in precision when using means, means plus standard deviations, and functional approximations of the RLDs, consider the design of an algorithm for a type 2 application scenario (as defined in Section 2.1) and the specific question of estimating the cutoff time $RT_{\max}$ for solving a given problem instance with a probability $p$. If only the mean run-time $E(RT)$ is known, the best estimate we can obtain for $RT_{\max}$ is given by the Markov inequality [Roh76] $P(RT \geq t) \leq E(RT)/t$:

$$RT_{\max} = E(RT)/(1-p) \tag{2.4}$$

If also the standard deviation $\sigma(RT)$ is taken into account, and under the condition that $RT$ has a finite variance, we can use the Tchebichev inequality $P(\mid RT - E(RT) \mid \geq \epsilon) \leq \sigma^2(RT)/\epsilon^2$ to obtain a better estimate:

$$RT_{\max} = \sigma(RT)/\sqrt{1-p} + E(RT) \tag{2.5}$$

If, however, we know that the run-time of a given Las Vegas algorithm is exponentially distributed,[4] we get a much more accurate estimate.

In the following example, we see the drastic differences between these three estimates. For a given Las Vegas algorithm applied to some problem the mean run-time and the standard deviation are 100 seconds each, a situation which is not untypical, *e.g.*, for stochastic local search algorithms for SAT. We are interested in the run-time required for obtaining

[3] This method is equivalent to the one used in [PW96], where a more detailed derivation of this estimate can be found.

[4] As we will discuss later, assuming an exponential RLD is not as far-fetched as it might seem, since we found that these can be observed for a number of modern stochastic local search algorithms on various problem classes.

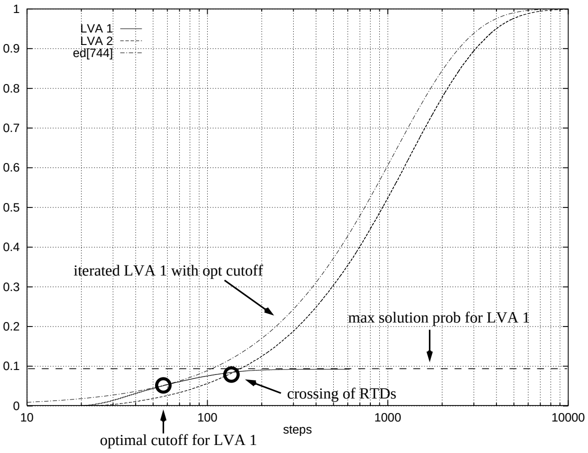


Figure 2.8: RLDs for two different Las Vegas algorithms on the same problem instance. Note the crossover of the two RLDs. By executing independent tries of LVA 1 using the optimal cutoff value, compared to LVA 2 for all run-lengths a speedup by a constant factor is achieved. A detailed discussion is given in the text.

a solution probability of 0.99. If nothing is known about the run-time distribution, we have to use the Tchebichev inequality and get an estimate of 1,100 sec. Assuming that the run-time is exponentially distributed, we get an estimate of 460 sec. If, on the other hand, even the standard deviation is unknown, we can only use the Markov inequality and estimate the run-time required for a solution probability of 0.99 as 10,000 sec! Intuitively, this is hardly surprising – just imagine some wildly different probability distributions having identical mean and standard deviation. The unspecific estimates based on the Markov and Tchebichev inequalities, of course, have to take into account all possible actual RLDs and therefore have to be quite inaccurate when compared to the precise estimates obtained from detailed knowledge of the actual RLD.

**Parameter Dependencies**

Another problem with numerous studies from recent literature on stochastic local search is the tacit assumption that several parameters of the considered algorithms can be studied independently. In specific cases, it is known that this assumption does not hold [HS96, SSS97]. For the evaluation of Las Vegas algorithms in general, it is crucial to be aware of possible parameter dependencies, especially those involving the cutoff time $RT_{\max}$ which plays an important role in type 2 and 3 application scenarios. By applying an RLD-based methodology, such parameter dependencies can be easier detected and characterised than when using basic descriptive statistics only. The following example shows, how basing the comparison of two Las Vegas algorithm's performance on expected run-times alone can lead to erroneos conclusions, because parameter dependencies are not taken into account.

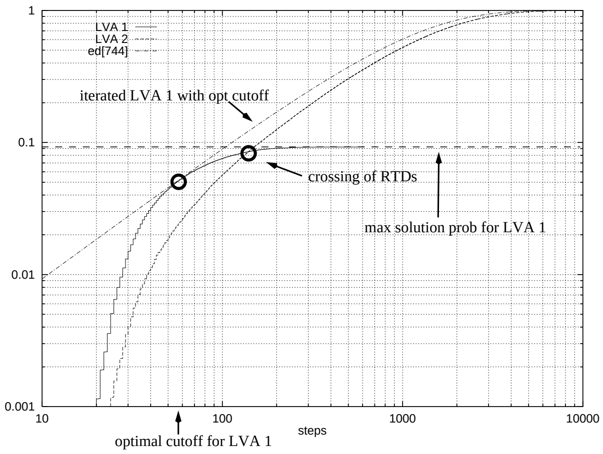


Figure 2.9: Same as Figure 2.8 using log-log representation. Note how for extremely short runs the relative difference in performance between LVA 1 and LVA 2 decreases again, while the performance of both algorithms rapidly deteriorates.

Figure 2.8 shows the RLDs of two different Las Vegas algorithms LVA 1 and LVA 2 for the same problem instance. The same data is shown as a log-log plot in Figure 2.9, giving a better view on the algorithm's behaviour for short runs. As can be easily seen, LVA 1 is essentially incomplete with an asymptotic solution probability approaching ca. 0.09, while LVA 2 is approximately complete. Now note the crossing of the two RLDs at ca. 120 steps. For smaller cutoff times, LVA 1 achieves considerably higher solution probabilities, while for greater cutoff times, LVA 2 shows increasingly superior performance. Consequently, if comparing only estimated run-times as described above, the result entirely depends on the cutoff chosen for conducting the experiments. Thus, when being not aware of the cross-over in the RLDs, there is a considerable risk of misinterpretation, as this comparison will suggest a (non-existing) dominance relation for almost all chosen cutoff-times.

Furthermore, we can easily observe that the RLD of LVA is less steep than an exponential distribution. Therefore, as argued in Section 2.2.2, using restart or multiple independent parallel runs of the algorithm, its performance can be considerably improved. Inn particular, using the optimal cutoff time of ca. 57 steps, ideally, the exponential run-time distribution marked "ed[744]" can be obtained, which realises a speedup of ca. 24% compared to LVA 2. Thus, the performance of LVA 2 is not only superior to that of LVA 1 for small cutoff times, but based on the RLDs, it is possible to modify algorithm LVA 1 such that its overall performance dominates that of LVA 2.

As this example demonstrates, basing the comparison of Las Vegas algorithms on expected run times is in the best case imprecise, in the worst case it leads to erroneous conclusions. The latter case occurs, when the two corresponding RLDs have at least one intersection. Then, obviously, the outcome of comparing the two algorithms depends entirely on the cutoff time $RT_{\max}$ which was chosen for the experiment. While in reality, one of the two algorithms is superior for small $RT_{\max}$ but is outperformed by the other candidate for large $RT_{\max}$, a comparison based on the mean run-times obtained for one specific $RT_{\max}$ value will give the incorrect impression that one algorithm is generally superior to the other. This fallacy arises especially in situations, where one or both of the candidate algorithms are not approximately complete. Then, while one algorithm might quickly converge to a lower asymptotic success probability, the other might be less effective in the initial phase of the search, but later on reach a higher asymptotic success probability. As we will show in Chapter 4, this situation can be often observed in practice.

### 2.3.2 Problem Type 2: Inhomogeneous Test-sets

Often, Las Vegas algorithms are tested on sets of randomly generated problem instances. For analysing a Las Vegas algorithm's behaviour on such a test-set, on each instance, a number of runs is performed and evaluated. Then, the final performance measure is usually obtained by averaging over all instances from the test set.

This last step, however, is potentially extremely problematic. Since the run-time behaviour on each single instance is characterised by an RLD (as discussed above), averaging over the test set is equivalent to averaging over these RLDs. Since in general, averaging over a number of distributions yields a distribution of a different type, the practice of averaging over RLDs (and thus the averaging over test sets) is quite prone to producing observations which do not reflect the behaviour of the algorithm on the individual instances, but rather a side-effect of this method of evaluation.

We exemplify this for Random-3-SAT near the phase-transition, a problem class which has been used in many studies of stochastic local SAT procedures, such as GSAT or GSAT with random walk [SLM92, GW95] (the problem class itself will be discussed in more detail in Chapter 4). Practically, Random-3-SAT test-sets are obtained by generating a number of sample instances and, as typical SLS algorithms for SAT are incomplete, filtering out insoluble instances using a complete SAT algorithm. By measuring the median run-lengths for each problem instance from a typical randomly generated and filtered test set, we obtain a distribution over the median hardness of the problems as shown in Figure 2.10. Since the median run-lengths were determined using 1,000 runs per instance, they are very stable which leaves the random selection of the instances as the main source of noise in the measured distribution.

Our own experimental studies have shown that some of the most popular SLS algorithms for SAT, such as WalkSAT [SKC94] or Novelty [MSK97], show approximately exponential RLDs. Specifically, under certain conditions, the RLDs on single instances from Random-3-SAT instance distribution can be well approximated by exponential distributions $ed[m]$, where $m$ is the number of local search steps required to solve the problem with probability 0.5, *i.e.*, the median of the RLD; these results will be presented in detail in Chapter 5 of this work.

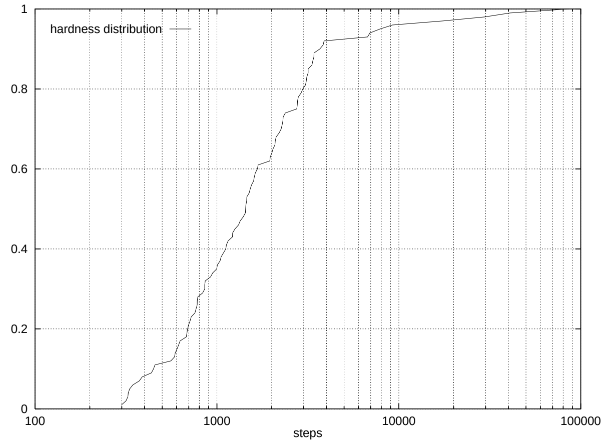


Figure 2.10: Distribution of median run-length for WalkSAT (noise=55/100) over a test set of 100 satisfiable Random-3-SAT formualae (unforced). Note the huge variance, especially for the hardest 10% of the test-set.

Now we assume that we average over two exponential RLDs with different median values $m_1, m_2$. Since the family of exponential distributions is not closed under averaging, this produces a combined distribution $d_{1,2}$ which is *not* exponentially distributed. For this combined distribution, shown in Figure 2.11, obviously an optimal cutoff time exists, which can be determined as detailed in Section 2.2.2. For the single RLDs, however, since they are exponential, optimal cutoffs does *not* exist (*cf.* Section 2.2.2). Thus, when averaging over RLDs, observations (here: existence of an optimal cutoff) can be made, which do not apply for *any* of the individual instances but are rather an effect of the averaging itself.

At the first glance, this might sound a bit paradoxical, but there is a perfect interpretation for this observation: when averaging over the RLDs, we do not distinguish between the probability of solving one or the other instance. Using the "optimal" cutoff inferred from the average RLD then simply means that solving the easier instance with a sufficiently high probability compensates for the very small solution probability for the harder instance going along with using this cutoff. So somehow, solving the easier instance gets priority over solving the harder instance. Under this interpretation, the "optimal" cutoff can be considered meaningful. Assuming, however, that in practice the goal in testing LVAs on test sets sampled from andom problem distributions is to get a realistic impression of the overall performance, *including hard instances*, the "optimal" cutoff inferred from the averaged RLD is substantially misleading.

Of course, a similar argument holds when averaging over test sets containing more than two instances. In Figure 2.11, we show the averaged RLD for the whole test set the hardness distribution of which is plotted in Figure 2.10. Obviously, here as well a "pseudo-optimal" cutoff can be observed, and the interpretation is exactly the same as in the two instance case. By increasing the size of the test-set, this situation is not changed, since it can

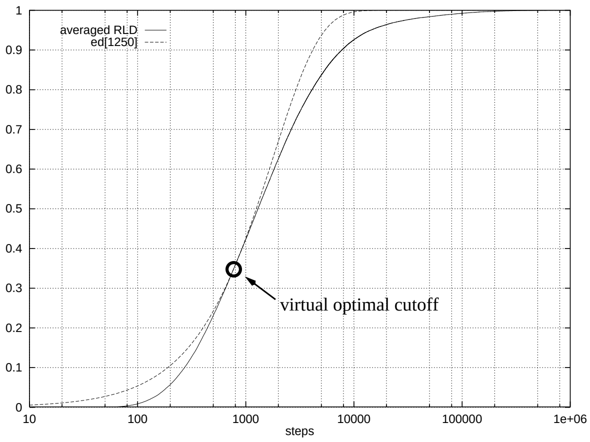


Figure 2.11: Average RLD for Walksat (noise=55/100, very large cutoff) on Random-3-SAT distr, 100 variables, 430 clauses; the virtual optimal cutoff at ca. 750 steps can be observed because the averaged RLD is not exponentially distributed (inhomogeneous test-set).

be theoretically shown that averaging over exponential distributions with different median values, one will *not* obtain another exponential distribution, *i.e.*, always "pseudo-optimal" cutoff times will be observed.

The above discussion shows, that by averaging over test sets, generally in the best case one observes knowingly a bias for solving certain problems, the practical use of which seems rather questionable. But far more likely, being not aware of these phenomena, the observations thus obtained are misinterpreted and lead to erroneous conclusions. One could, however, imagine a situation in which averaging over test sets is not that critical. This would be given if the test sets are very homogeneous in the sense that the RLDs for each single instance are roughly identical. Unfortunately, this sort of randomised problem distributions seems to be very rare in practice. Certainly, Random-3-SAT is not homogeneous in this sense, and at least the authors are currently not aware of any sufficiently complex homogenous randomised problem distribution for SAT or CSP.[5]

The fundamental problem with averaging over random problem distributions is the mixing of two different sources of randomness in the evaluation of algorithms: the nondeterministic nature of the algorithm itself, and the random selection of problem instances. Assuming that in the analysis of Las Vegas algorithms one is mostly interested in the properties of the algorithm, at least a very good knowledge of the problem distribution is required for separating the influence of these inherently different types of randomness.

[5]There is, however, some indication, that certain randomised classes of SAT encoded problems from other domains, such as compactly encoded subclasses of the Hamilton Circuit Problem [Hoo96c], are significantly more homogeneous than Random-3-SAT.

## 2.4 Extension to Optimisation Problems

Many problems occurring in practical contexts are optimisation problems rather than decision problems, *i.e.*, there is a notion of solution quality which, independently from the way a solution was obtained, characterises its value or usefulness. Las Vegas algorithms for optimisation problems (optimisation LVAs) are defined exactly as in the case of decision problems; only now, both run-time $RT$ and solution quality $SQ$ are random variables. Thus, for fixed solution quality $sq'$, the run-time $RT$ is a random variable, and for fixed run-time $rt'$ the solution quality $SQ$ is a random variable.

Often, for a given optimisation problem finding valid solutions is relatively easy, while finding optimal or near-optimal valid solutions might be extremely hard. We therefore make the assumption that for each instance of an optimisation problem there exists always a valid solution and Las Vegas algorithms for solving such an optimisation problem always return a valid solution. In this case, the optimisation problem $\Pi$ is tightly related to the following family of decision problems: given a problem instance $\pi$ and a desired solution quality $q'$, is there a solution $s$ of $\pi$ such that the solution quality $q(s)$ is at most $q'$, *i.e.* (for a minimisation problem), $\exists s : s \text{ solves } \pi \wedge q(s) \leq q'$. If the optimal solution $q^*$ is known, it is often more elegant to use relative solution qualities $r(s) = q(s)/q^*(s)$, such that $r(s) = 1$ always characterises optimal solutions. In the following, we will use relative solution qualities unless explicitly indicated otherwise.

Interestingly, many LVAs for decision problems have been directly derived from optimisation LVAs; and vice versa, most LVAs for decision problems can be extended into optimisation LVAs in a natural and simple way. This merely reflects the close correspondence between certain decision problems and their optimisation variants which will be discussed later in this work. Good examples for LVAs which can be analogously applied to decision and optimisation problems can be found in the stochastic local search domain; some of the most popular SLS algorithms, like Simulated Annealing, Tabu Search, Genetic Algorithms, or Ant Colony Optimisation belong to this category.

### 2.4.1 Norms of Behaviour

With respect to the related decision problems introduced in above, the behaviour of a Las Vegas algorithm can be analysed using the methodology introduced before. Formally, we first extend the definitions of the norms of behaviour introduced in Section 2.1.1:

- A Las Vegas algorithm $A$ is *$q'$-complete* for a problem class $\Pi$, if for each soluble instance $\pi \in \Pi$ it can be guaranteed to find a solution $s$ with quality $q(s) \leq q'$ within run-time $t_{max}$, where $t_{max}$ is an instance-dependent constant. Let $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q')$ denote the probability that $A$ finds a solution of quality $\leq q'$ for a soluble instance $\pi$ in time $\leq t$, then $A$ is complete exactly if for each $\pi$ there exists some $t_{max}$ such that $P_s(RT_{A,\pi} \leq t_{max}, SQ_{A,\pi} \leq q') = 1$.

- A Las Vegas algorithm $A$ is *approximately $q'$-complete* for a problem class $\Pi$, if for each soluble instance $\pi \in \Pi$ its probability for finding a solution $s$ with quality $q(s) \leq q'$ converges to 1 as the run-time approaches $\infty$. Thus, $A$ is approximately $q'$-complete, if for each soluble instance $\pi$, $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q') = 1$.

- A Las Vegas algorithm $A$ is *essentially $q'$-incomplete* for a problem class $\Pi$, if it is not approximately $q'$-complete, *i.e.*, if there exists a soluble problem instance $\pi$, for which $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q') < 1$.

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

### 2.4.2 Application Scenarios

As for the application scenarios discussed in Section 2.1.2, the situation is similar as for decision problems; only now, there is the solution quality as an additional factor to be considered. Generally, there will be a tradeoff between run-time and solution quality. This trade-off will strongly depend on the specific requirements of the application situation: while in some cases run-time does not matter but solution quality is essential, other scenarios will impose tight restrictions on run-time, whereas solution quality is relatively less important. But since we are dealing with probabilistic algorithms, the situation is actually even more complex: for a given time-limit $t'$, one might have to decide whether to prefer an algorithm with a high mean solution quality but a relatively large standard deviation over another algorithm which produces slightly less optimal solutions in a more consistent way. This becomes very obvious when generalising type 3 application scenarios (*cf.* Section 2.1.2) to optimisation problems: the utility of a solution now depends on its quality and the time needed to find it. Compared to the decision problem case, there is an additional degree of freedom in choosing the utility function $U(t,q) : \mathbb{R}^+ \times \mathbb{R} \mapsto [0,1]$, and the nature of the utility function to be used will generally reflect application specific tradeoffs between solution time and solution quality.

Therefore, for optimisation problems it seems to be even more important to avoid making implicit assumptions on the application scenario by prematurely reducing the data from empirical testing of Las Vegas algorithms to some basic statistics of the bivariate distribution $P_s(RT, SQ)$. This distribution characterises uniquely and completely the behaviour of a Las Vegas optimisation algorithm; running experiments is equivalent to sampling this distribution.

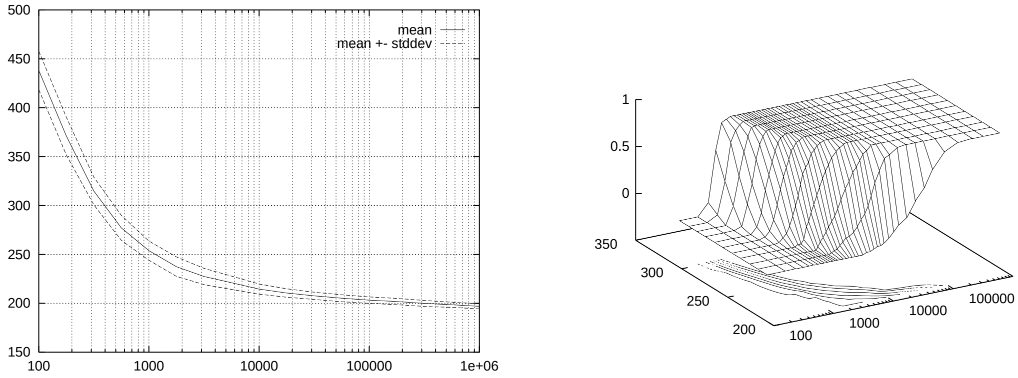


Figure 2.12: *Left:* development of mean solution quality and standard deviation for an SLS optimisation algorithm on a hard minimisation problem; *right:* qualified RLD for the same algorithm and problem instance.

### 2.4.3 Qualified RTDs and RLDs

Formally, a qualified RTD is a bivariate probability distribution characterising the two-dimensional random variable $(RT_{A,\pi}, SQ_{A,\pi})$ which models the run-time behaviour of Las Vegas optimisation algorithm A. The most natural and effective way of empirically approximating qualified RTDs is by measuring solution quality traces over multiple runs of the algorithm. This is most effectively done by recording the run-time and solution quality whenever during a run a solution is found, the quality of which strictly improves on the best solution found so far in this run. More formally, let $k$ be the number of runs and let $sq(t, j)$ denote the quality of the best solution found in run $j$ until time $t$. Then the qualified (cumulative empirical) run-time distribution is defined by $\widehat{P}(RT \leq t', SQ \leq q') = \#\{j | sq(t', j) \leq q'\}/k$. Again, to avoid the type of problems which arise in the context of averaging over test-sets (cf. Section 2.3), the qualified RTDs are based on single problem instances.

Like in the case of Las Vegas algorithms for decision problems, these RTDs describe the behaviour of the algorithm uniquely and completely. Therefore, all kinds of basic descriptive statistics can be easily obtained from the RTD data. Popular choices are mean and standard deviation of the solution quality for a given run-time (cutoff time) as well as mean, standard deviation, and percentiles of the run-time required for obtaining a given solution quality. Beyond these simple descriptive statistics, the two types of marginal distributions of the qualified (bivariate) RTD offer a good basis for characterising the run-time behaviour of optimisation LVAs: the first of these corresponds to the standard RTDs for the related decision problems discussed before and describe the distribution of the run-times needed for obtaining a specific solution quality. The other type of marginal distributions are solution quality distributions (SQDs) for a fixed run-time. Furthermore, when dealing with SQDs, it is often interesting to report and analyse their time-dependent basic statistics, like $\overline{SQ}(t)$, the function characterising the development of the mean solution quality over time.

Figure 2.12 (left) shows for a SLS optimisation algorithm on a particular instance of a

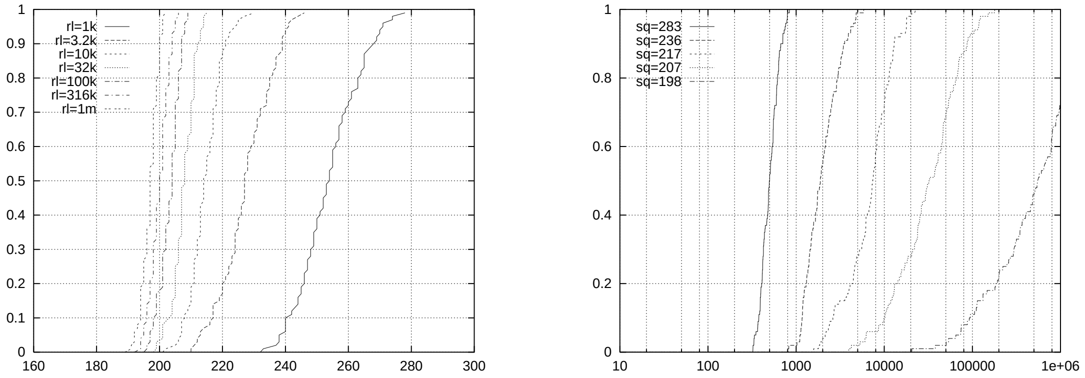


Figure 2.13: *Left:* marginal SQDs for the qualified RLD from Figure 2.12; *right:* marginal RLDs for the same qualified RLD.

| run-length | mean | stddev | stddev/mean | min | max | median | $Q_{25}$ | $Q_{75}$ | $Q_1$ | $Q_{90}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 438.21 | 19.64 | 0.04 | 397 | 491 | 439 | 425 | 452 | 411 | 460 |
| 1,000 | 253.83 | 9.81 | 0.04 | 232 | 278 | 253 | 247 | 261 | 240 | 267 |
| 10,000 | 214.55 | 5.19 | 0.02 | 201 | 230 | 214 | 211 | 218 | 207 | 221 |
| 100,000 | 203.22 | 3.31 | 0.02 | 195 | 209 | 204 | 201 | 206 | 198 | 207 |

Table 2.2: Basic SQD statistics for different run-lengths (same data as Figure 2.13, left).

minimisation problem the development of solution quality and standard deviation over time. 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 an increasing number of local search steps, the mean solution quality improves monotonically while the standard deviation of the solution quality decreases. Thus, for longer runs the algorithm tends to find better solutions in a more consistent way. However, this view of the algorithm's behaviour is rather simplistic compared to the complete qualified RLD shown in Figure 2.12 (right). Here, the contours of the three dimensional qualified RLD surface projected into the run-length / 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 change in slope indicates the differences in variance of the run-time and solution quality distributions.

Details can be easier seen when examining the corresponding marginal distributions shown in Figure 2.13. The marginal SQDs (left) reflect the behaviour observed before, although they provide more information, like the fact that the solution quality distributions are consistently unimodal, which cannot be seen from the development of mean solution quality. Also, based on the marginal SQD data, other basic statistics, such as percentiles and their distances or ratios, can be easily obtained. For our example, some of these statistics can be seen in Table 2.2. Note, how the mean as well as the 10% to 90% percentiles ($Q_{10}$ to $Q_{90}$) and the median monotonically decrease for growing run-lengths.

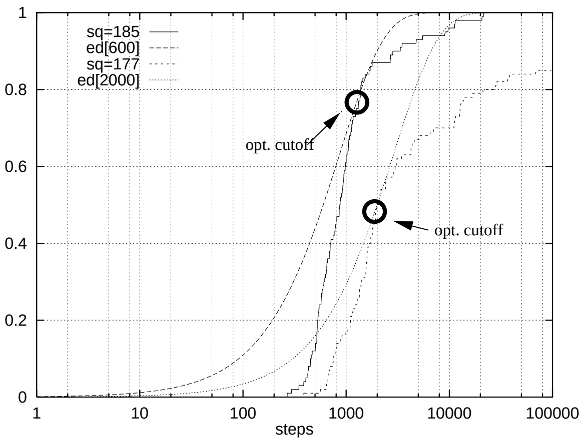


Figure 2.14: Two marginal RLDs for an SLS optimisation algorithm on a hard minimisation problem; for each curve, a different optimal cutoff time can be observed (the optimal cutoff is located where the RTD meets the corresponding exponential distribution).

A different view on the same data is given by the marginal RLDs, shown on the right side of Figure 2.13. Note that when tightening the solution quality bound, the marginal RLDs get less steep. This reflects the fact that for obtaining better solutions, not only the mean run-time increases, but also its variance. Thus, the qualified RLD data, which is the basis for these marginal distributions, provides a much more detailed picture of an SLS optimisation algorithm's behaviour than more established methods for evaluation based on simple solution quality statistics or their development over time.

### 2.4.4 Evaluating Optimisation LVAs

The methodology for evaluating and comparing optimisation LVAs is basically the same as for decision LVAs. Key techniques, as the approximation of empirical RTDs using continuous probability distribution functions, can be analogously applied to the optimisation case. However, directly dealing with bivariate qualified RTDs can be difficult. Therefore, it is usually more feasible to do functional approximations of the marginal RTDs and SQDs or time-dependent SQD statistics. In this case, the methodology for finding optimal approximations and testing their goodness-of-fit is exactly the same as for decision LVAs.

Again, the question of finding optimal cutoff times or optimal parallelisations is of considerable interest when analysing a given optimisation Las Vegas algorithm. Only here the problem is more complex than in the case of decision LVAs, where an optimal cutoff could be easily defined in an intuitive way. Consider a situation where an optimisation LVA A exhibits marginal RLDs for solution qualities $q_1, q_2$ as shown in Figure 2.14. Here, depending on the desired solution quality, two different optimal values for the cutoff time can be derived. Thus, generally the optimal cutoff can be regarded as a function of the desired solution quality rather than as a single value. Consequently, in most cases there is a trade-off between solution quality and run-time; one has to choose between optimal behaviour (*i.e.*, maximal probability) for short run-times while accepting lower quality solutions, and maximal probability for finding high-quality solutions while accepting long run-times.

Similiar considerations hold for comparing optimisation LVAs. The concept of domination which we introduced in Section 2.2.2 can be analogously applied to the optimisation case: For minimisation LVAs A and B, A dominates B if for any given time $t$ and any solution quality $q$, A has a higher probability than B for finding a solution of quality $\leq q$ in time $\leq t$, *i.e.*, $\forall t, q : P(RT_A \leq t, SQ_A \leq q) \geq P(RT_B \leq t, SQ_B \leq q)$ and $\exists t, q : P(RT_A \leq t, SQ_A \leq q) > P(RT_B \leq t, SQ_B \leq q)$. But often, for two given algorithms no such dominance relation holds; instead the two qualified RTDs have at least one intersection. In this situation, analogously to the decision problem case, switching between the algorithms can enhance overall performance. However, for optimisation problems there is considerably more room for tradeoffs. For example, algorithm A might give a better mean solution quality up to some time limit $t'$, while for $t > t'$, B will produce better solutions on average. But although for $t < t'$ A seems to be superior to B, B could still be better than A when just comparing the probabilities for finding high quality solutions within short runs. Formally, in this situation for all $t < t'$, the mean solution quality for algorithm B would be lower than for A, while, *e.g.*, when comparing the 0.1 percentiles of the corresponding SQDs, B would be superior to A. In situations like this, it depends entirely on the application scenario: sometimes, only high quality solutions are useful – then, in our example, algorithm B would be preferable. If, however, all solutions have to be used and average solution quality counts, A would be preferable for $t < t'$.

When evaluating and comparing optimisation algorithms, very often the application scenario is unknown or unspecified. Our considerations and examples have shown that for such general empirical analyses, it is extremely important to get a most comprehensive view of an algorithm's run-time behaviour, since by prematurely reducing the data to specific simple descriptive statistics, like the development of mean or standard deviation of the solution quality over time, important aspects of run-time behaviour are neglected. As for decision LVAs, the RTD-based methodology gives a considerably more detailed and realistic view of the algorithms' behaviour while not requiring a significant overhead in data aquisition.

## 2.5 Related Work

The term *Las Vegas Algorithm* was originally introduced by Laszlo Babai [Bab79]; it was first embraced by European researchers and only later picked up by Americans. The literature on Las Vegas algorithms is relatively sparse Some interesting theoretical work on the parallelisation of Las Vegas algorithms has been done by Michael Luby and Wolfgang Ertel [LE93]. They discuss parallelisation schemes for algorithms with known and unknown run-time distributions. Closely related is a theoretical analysis of optimal strategies for selecting cutoff times presented by Luby, Sinclair, and Zuckerman in [LSZ93]. For the case of a Las Vegas algorithm with known run-time distribution they theoretically derive an optimal strategy based on restarting the algorithm after a fixed cutoff time which can be exactly determined from the RTD. They also derive an optimal strategy for selecting optimal cutoff times in the case where the RTD is unknown. One might argue that this case is more realistic, since usually when solving a problem the characteristics of the particular instance are not known *a priori*. While in the most general case, this is undoubtedly true, it is not unusual to assume that for certain problem classes, general properties which hold for the whole problem class can be determined by studying a small number of instances [Min96]. In the case of known RTDs or distribution types, significantly better strategies for selecting cutoff times can be devised than in the case of unknown RTDs. Therefore, it is advantageous to study the run-time behaviour of a Las Vegas algorithms on specific problem classes and to use this information for improving the algorithm's peformance.

To our best knowledge, SLS algorithms for SAT or related problems have not been investigated in the more general context of Las Vegas Algorithms before. There is some work on the empirical analysis of complete search algorithms for SAT and CSP on randomly generated instances from Random-3-SAT and binary CSPs. Kwan shows that for random CSPs from the under- and overconstrained regions the run-time behaviour of modern complete algorithms cannot be characterised by normal distributions [Kwa96]. 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 [FRV97]. In [RF97], this approach is extended to cost distributions for unsolvable problems from the overconstrained region. All these investigations concentrate on the cost distribution for a sample of instances from a fixed random distribution. In this approach, two sources of randomness are mixed: the randomness which is inherent to the algorithm and the variation in the characteristics of problem instances which are randomly drawn from a random problem distribution. In Section 2.3 we pointed out some pitfalls of this approach, which are avoided using our methodology.

Run-time distributions have been used for other problem domains, but again, this approach was mainly applied to complete algorithms. One exception is the work of Taillard, who reports exponential RTDs for a Tabu Search based algorithm for the Job-Shop Scheduling problem [Tai94]. In earlier work, the same author used a similar approach to improve on the best known solutions of small Quadratic Assignment Problems [Tai91]. Taillard also points out that for certain types of run-time distributions restarting the given algorithm can be used to obtain higher solution probabilities. Similar results are reported by Hogg and Williams, who investigate the behaviour of a backtracking algorithm for the Graph Colouring problem based on the Brelaz heuristic[6] [HW94a]. They use RTDs in the context of parallelisation and find that the obtainable speed-up strongly depends on the actual RTDs. Recently, a similar approach was used by Gomes and Selman, who tried to design algorithm portfolios using backtracking algorithms based on the Brelaz heuristic for solving a

[6] Since the Brelaz heuristic involves random tie-breaking, backtracking algorithms using this heuristic are Las Vegas algorithms, even if they are deterministic in any other respect.

special kind of CSP [GS97b]. Using run-time distributions it was found that one algorithm dominated all others. Consequently, using the optimal portfolio would contain only this algorithm. Closer examination of the RTDs showed that these can be approximated by "heavy-tailed" distributions, a fact which can be exploited for improving the performance of these heuristics by using restart [GSC97]. Latest results from this line of research indicate that a similar approach can be used for improving state-of-the-art complete SAT algorithms (like *Satz* [LA97]) [GSK98]. There is some evidence that the randomised algorithms thus obtained might be the best complete SAT algorithms which are currently known.

Finally, it should be noted that run-time distribution also have been observed occasionally in the Operations Research literature, where Las Vegas algorithms, specifically local search algorithms, play a prominent role for solving many hard problems. However, this kind of approach has been limited to concrete examples and specific application aspects.

## 2.6 Conclusions

In this chapter, we introduced a novel approach for the empirical analysis of Las Vegas Algorithms. Our method is based on measuring and analysing run-time distributions (RTDs) for individual problem instances. Based on a classification of application scenarios for Las Vegas Algorithms, we have shown that in general, only RTDs provide all the information required to adequately describe the behaviour of the algorithm. We demonstrated, how based on functional approximations of RTDs for individual problem instances, the behaviour of Las Vegas Algorithms can be adequately characterised even for sets or distributions of problem instances. This way, our refined methodology can be used to obtain interesting characterisations for the run-time behaviour of some of the most popular stochastic local search algorithms in recent AI research (this issue will be further elaborated in Chapter 5). Generally, compared to the methodology which is commonly used for empirical analyses for LVAs in AI, our approach gives a considerably more detailed and realistic view of the behaviour of these algorithms. At the same time it does not require an additional overhead in data aquisition, but only uses the collected data in a more efficient way.

Furthermore, we have shown how methods which have been commonly used in AI literature on stochastic local search can lead to misinterpretations and erroneous conclusions. In particular, we identified and discussed two pitfalls which are commonly arising in the context of an inadequate empirical methodology: superficial analysis of the run-time behaviour and averaging over inhomogeneous test-sets. As we have shown, by using our RTD-based methodology, these fallacies are effectively avoided. Finally, we demonstrated how the RTD-based empirical methodology can be generalised to Las Vegas Optimisation Algorithms in a straightforward way.

Although run-time distributions have been observed occasionally before, their use has been limited to concrete examples and specific application aspects. Our improved and refined empirical methodology is considerably more general; we believe that it will prove to be very useful for analysing the run-time behaviour of Las Vegas Algorithms in general, and SLS algorithms in particular. In the past, even the most successful applications of Las Vegas Algorithms in various areas of AI have been based on a fairly limited understanding of their behaviour. It appears to be not too bold to assume that using a more adequate empirical methodology will be the basis for improving the understanding of these algorithms and thus facilitate further successes in the development and application of this type of algorithms.

In the following chapters of this work, we will use the methodology developed here as a basis for analysing and comparing the behaviour of SLS algorithms for SAT. As we will see, it is crucial for obtaining many interesting results (particularly the ones presented in Chapter 5) of theoretical as well as practical interest.