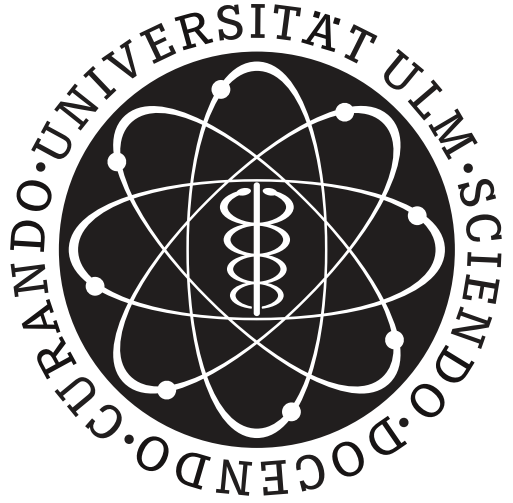
RESTART STRATEGIES



Universität Ulm
Institut für Theoretische Informatik
Leiter: Prof. Dr. Uwe Schöning

DISSERTATION

zur Erlangung des Doktorgrades **Dr. rer. nat.** der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

vorgelegt von
Jan-Hendrik Hans Lorenz aus **Dortmund**
Ulm, 2021

Jan-Hendrik Hans Lorenz
Restart Strategies

AMTIERENDER DEKAN:
Prof. Dr. Maurits Ortmanns

GUTACHTER:
Prof. Dr. Uwe Schöning
Prof. Dr. Angelika Steger
Prof. Dr. Jacobo Torán

TAG DER PROMOTION:
26.07.2021

ACKNOWLEDGMENT

First and foremost, I wish to thank my supervisor, Prof. Dr. Uwe Schöning, for the many discussions and insights without which my thesis would never have been possible. I would also like to extend my thanks to Prof. Dr. Jacobo Torán as well as Prof. Dr. Angelika Steger for reviewing my thesis. I am sincerely grateful to Prof. Dr. Hans Kestler and Prof. Dr. Birte Glimm for their committee membership.

I deeply appreciate the proofreading as well as the many helpful comments and remarks I received from my colleagues Julian Nickerl and Florian Wörz. Additionally, I would like to thank my other colleagues for the stimulating discussions and the good atmosphere we have in the department.

I owe my deepest gratitude to my parents, Ursula and Klaus Lorenz, for the support and love I received from a young age. Finally, I am forever indebted to my fiancée, Jiao Hu, who always stayed by my side and has contributed so much to enable me to complete this thesis.

This work is supported by the state of Baden-Württemberg through bwHPC.

ABSTRACT

Restarting is a technique employed by many randomized algorithms for a variety of different problems. If the algorithm has not been successful after a certain time, the algorithm is reset, and a new try is started that is independent of the past attempt. For some problems, restarts can be used to achieve improved expected runtimes, while for others, restarts deteriorate the expected runtimes. For this reason, it is essential to gain insight into when restarts are beneficial and which restart strategy should be chosen.

In this thesis, we first show that the problem of deciding whether restarts have a positive effect is NP-hard, even when the underlying probability distribution is fully known. Furthermore, we prove that the computation of the expected runtime for a given restart strategy is #P-hard. On the other hand, an efficient $(4 + \varepsilon)$ approximation algorithm for the optimal restart strategy is provided.

In this context, we frequently assume that a given function is, in fact, a valid description of a probability distribution. The complexity of verifying these assumptions is also analyzed. We show that deciding whether a given function corresponds to a cumulative probability function is coNP-hard. Moreover, deciding whether a given function represents a probability mass function is $\mathrm{P}^{\#\mathrm{P}}$-hard.

Although these problems are challenging in general, we show certain special cases in which they are simple. In particular, we introduce conditions for the usefulness of restarts as well as for determining an optimal restart strategy. It is shown that some typical parameters have either no or only a limited influence on the restart properties. Furthermore, it is proven that restarts are always useful for so-called long-tail distributions. By means of these results, it is possible to considerably simplify the analysis of many probability distributions; consequently, the NP-hardness does not pose an obstacle in many practical applications.

Luby's strategy [123] is a popular restart strategy in practical applications. Suppose the underlying probability distribution is defined on the natural numbers. In that case, Luby's strategy provides a certain performance guarantee: The expected runtime using Luby's strategy is at most worse by a logarithmic factor than the expected runtime using the optimal restart strategy [123]. However, we prove that this is not a general property, and furthermore, there is no universal restart strategy

providing such a performance guarantee. On the other hand, we identify conditions under which such universal strategies do exist.

A particular obstacle for determining an optimal restart strategy is that the underlying probability distribution has to be known. However, that is usually not the case for unsolved instances. To circumvent this problem, we design a machine learning pipeline predicting the runtime distribution of unseen instances. An optimal restart strategy is derived for the SAT solver PROBSAT based on these predicted distributions. We experimentally demonstrate that this approach statistically significantly improves the performance of PROBSAT. We achieve an average speedup factor of more than 50 on the domain on which PROBSAT encounters the most significant difficulties.

Lastly, restarts are also considered in an environment in which a deadline is present and in which the corresponding algorithm runs in parallel. We specify a condition for finding the optimal restart strategy and show that increasing the number of utilized processors has a super-linear benefit in this scenario.

ZUSAMMENFASSUNG

Restarting ist eine Technik, die von vielen randomisierten Algorithmen für eine Vielzahl von verschiedenen Problemstellungen verwendet wird. Falls der Algorithmus nach einer bestimmten Zeit nicht erfolgreich war, wird der Algorithmus zurückgesetzt und es wird ein neuer Versuch gestartet, der unabhängig vom letzten Versuch ist. Mit Restarts lassen sich bei manchen Problemen verbesserte erwartete Laufzeiten erzielen, während bei anderen Problemen Restarts die erwarteten Laufzeiten verschlechtern. Aus diesem Grund ist es wichtig, Einblicke zu gewinnen, wann Restarts von Vorteil sind beziehungsweise welche Restart-Strategie gewählt werden sollte.

In dieser Arbeit zeigen wir zunächst, dass die Entscheidung, ob Restarts einen positiven Effekt haben, NP-schwer ist, sogar dann, wenn die zugrunde liegende Zufallsverteilung vollständig bekannt ist. Weiterhin beweisen wir, dass die Berechnung der erwarteten Laufzeit für eine gegebene Restart-Strategie #P-schwer ist. Demgegenüber wird ein effizienter $(4+\varepsilon)$ Approximationsalgorithmus für die optimale Restart-Strategie vorgestellt.

In diesem Zusammenhang wird häufig angenommen, dass es sich bei einer gegebenen Funktion tatsächlich um eine valide Beschreibung einer Wahrscheinlichkeitsverteilung handelt. Die Komplexität der Verifizierung dieser Annahmen wird ebenfalls analysiert. Wir zeigen, dass die Entscheidung, ob eine gegebene Funktion einer kumulativen Wahrscheinlichkeitsfunktion entspricht, coNP-schwer ist. Außerdem ist das Entscheiden, ob eine gegebene Funktion eine Wahrscheinlichkeitsmassenfunktion darstellt, $\mathrm{P}^{\#\mathrm{P}}$-schwer.

Obwohl diese Probleme im Allgemeinen schwer sind, zeigen wir bestimmte Spezialfälle, in denen sie simpel sind. Insbesondere führen wir Bedingungen für die Nützlichkeit von Restarts sowie für die Bestimmung einer optimalen Restart-Strategie ein. Es wird nachgewiesen, dass einige typische Parameter entweder keinen oder nur einen geringen Einfluss auf die Restart-Eigenschaften haben. Weiterhin wird bewiesen, dass Restarts für sogenannte Long-Tail-Verteilungen immer nützlich sind. Mithilfe dieser Ergebnisse ist es möglich, die Analyse vieler Zufallsverteilungen erheblich zu vereinfachen, sodass die NP-Härte in vielen praktischen Anwendungen kein Hindernis darstellt.

Die Luby-Strategie [123] ist eine in der Praxis beliebte Restart-Strategie. Wenn die zugrunde liegende Wahrscheinlichkeitsverteilung auf den natürlichen Zahlen definiert ist, bietet die Luby-Strategie eine Leistungsgarantie: Die erwartete Laufzeit unter Verwendung der Luby-Strategie ist höchstens um einen logarithmischen Faktor schlechter als die erwartete Laufzeit unter Verwendung der optimalen Restart-Strategie [123]. Allerdings beweisen wir, dass dies keine allgemeine Eigenschaft ist, und darüber hinaus, dass es keine universelle Restart-Strategie gibt, die eine solche Leistungsgarantie bietet. Andererseits identifizieren wir Voraussetzungen, unter denen solche universellen Strategien doch existieren.

Ein besonderes Hindernis für die Bestimmung einer optimalen Restart-Strategie ist, dass die zugrunde liegende Wahrscheinlichkeitsverteilung bekannt sein muss. Für ungelöste Probleminstanzen ist diese jedoch zumeist unbekannt. Um dieses Problem zu umgehen, entwerfen wir eine Machine Learning Pipeline, die die Laufzeitverteilung von ungelösten Instanzen vorhersagt. Auf der Basis dieser vorhergesagten Verteilungen wird eine optimale Restart-Strategie für den SAT-Solver PROBSAT ermittelt. Wir zeigen experimentell, dass dieser Ansatz die Leistung von PROBSAT statistisch signifikant verbessert. Auf der Domäne, auf der PROBSAT die größten Schwierigkeiten hat, erreichen wir einen durchschnittlichen Speedup-Faktor von mehr als 50.

Schließlich werden Restarts auch in einer Umgebung betrachtet, in der eine Deadline vorhanden ist und in der der entsprechende Algorithmus parallel läuft. Wir geben eine Bedingung für das Finden der optimalen Restart-Strategie an und zeigen, dass man in diesem Szenario superlinear von der Anzahl der eingesetzten Prozessoren profitiert.

PUBLICATIONS

Parts of this thesis appeared in the following publications:

Jan-Hendrik Lorenz.
Completion Probabilities and Parallel Restart Strategies under an Imposed Deadline.
In: *PLOS ONE* 11.10 (2016), pp. 1–15.

Jan-Hendrik Lorenz.
Runtime Distributions and Criteria for Restarts.
In: *arXiv preprint arXiv:1709.10405* (2017).

Jan-Hendrik Lorenz.
Runtime Distributions and Criteria for Restarts.
In: *SOFSEM 2018: Theory and Practice of Computer Science*. Springer International Publishing, 2018, pp. 493–507.
This work received the best student paper award of the conference.

Jan-Hendrik Lorenz.
On the complexity of restarting.
In: *International Computer Science Symposium in Russia*. Springer, 2019, pp. 250–261.
This work received the best student paper award of the conference.

Jan-Hendrik Lorenz.
Restart Strategies in a Continuous Setting.
In: *Theory of Computing Systems* (2021), pp. 1–22.

Jan-Hendrik Lorenz and **Julian Nickerl**.
The Potential of Restarts for ProbSAT.
In: *International Conference on Computer Aided Systems Theory*. Springer, 2019, pp. 352–360.

Jan-Hendrik Lorenz and **Julian Nickerl**.
The Potential of Restarts for ProbSAT.
In: *arXiv preprint arXiv:1904.11757* (2019).

Jan-Hendrik Lorenz and **Uwe Schöning**.
Promise Problems on Probability Distributions.
In: *Complexity and Approximation*. Springer, 2020, pp. 57–66.

Florian Wörz and **Jan-Hendrik Lorenz**.
Evidence for Long-Tails in SLS Algorithms.
In: *29th Annual European Symposium on Algorithms (ESA 2021)*. 2021, 82:1–82:16.
This work received the best student paper award of the conference.

Furthermore, software and data relevant to this dissertation in which the author was involved are listed below.

Jan-Hendrik Lorenz.
The Potential of Restarts for ProbSAT on Instances with a Hidden Solution.
Supplementary data and source code. Version v1.0. The data and code is available at `https://doi.org/10.5281/zenodo.4533804`. Zenodo, 2021.

Jan-Hendrik Lorenz and **Julian Nickerl**.
The Potential of Restarts for ProbSAT on Uniform Instances.
Supplementary data and source code. Version v1.0. The data and code is available at `https://doi.org/10.5281/zenodo.4533842`. Zenodo, 2021.

Jan-Hendrik Lorenz and **Florian Wörz**.
concealSATgen.
Source code. The code is available at `https://github.com/FlorianWoerz/concealSATgen`. 2020.

Jan-Hendrik Lorenz and **Florian Wörz**.
On the Effect of Learned Clauses on Stochastic Local Search.
Supplementary data. Version 1.0. The data is available at `https://doi.org/10.5281/zenodo.3776052`. Zenodo, 2020.

FURTHER PUBLICATIONS

In addition, the author has contributed to the following publications.

Jan-Hendrik Lorenz and **Florian Wörz**.
On the Effect of Learned Clauses on Stochastic Local Search.
In: *Theory and Applications of Satisfiability Testing*. Springer International Publishing, 2020, pp. 89–106.

David Mödinger, **Jan-Hendrik Lorenz**, and **Franz J. Hauck**.
Statistical privacy preserving message dissemination for peer-to-peer networks.
In: *arXiv preprint arXiv:2102.01615* (2021).

David Mödinger, **Jan-Hendrik Lorenz**, **Rens W. van der Heijden**, and **Franz J. Hauck**.
Unobtrusive monitoring: Statistical dissemination latency estimation in Bitcoin's peer-to-peer network.
In: *PLOS ONE* 15.12 (2020), pp. 1–21.

Julian Nickerl, **David Mödinger**, and **Jan-Hendrik Lorenz**.
From Local Network Formation Game to Peer-to-Peer Protocol.
In: *2021 International Symposium on Electrical, Electronics and Information Engineering*. Association for Computing Machinery, 2021, pp. 483–492.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS

SETS

$\mathbb{N}$	The set of natural numbers: $\mathbb{N} = \{1, 2, 3, \dots\}$.
$\mathbb{N}_0$	The set of natural numbers including zero: $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.
$\mathbb{Z}$	The set of whole numbers: $\mathbb{Z} = \{x \mid x \in \mathbb{N}_0 \text{ or } -x \in \mathbb{N}\}$.
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}_+$	The set of strictly positive real numbers: $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x > 0\}$.
$\mathcal{P}(S)$	The power set of a set S: $\mathcal{P}(S) = \{T \mid T \subseteq S\}$.

LANGUAGES

ε	The empty word.
L^n	The n-fold concatenation of a set L: $L^0 = \{\varepsilon\}$ and $L^n = \{xy \mid x \in L^{n-1} \text{ and } y \in L\}$ for $n \geq 1$.
L^*	The Kleene star on a set L: $L^* = \bigcup_{i \geq 0} L^i$.

LANDAU SYMBOLS

$$
\begin{aligned}
f(n) = O\big(g(n)\big) &\quad\Leftrightarrow\quad \exists C > 0\, \exists n_0\, \forall n > n_0 : |f(n)| \leq C \cdot g(n) \\
f(n) = o\big(g(n)\big) &\quad\Leftrightarrow\quad \forall C > 0\, \exists n_0\, \forall n > n_0 : |f(n)| < C \cdot g(n) \\
f(n) = \Omega\big(g(n)\big) &\quad\Leftrightarrow\quad \exists C > 0\, \exists n_0\, \forall n > n_0 : |f(n)| \geq C \cdot g(n) \\
f(n) = \omega\big(g(n)\big) &\quad\Leftrightarrow\quad \forall C > 0\, \exists n_0\, \forall n > n_0 : |f(n)| > C \cdot g(n) \\
f(n) \sim g(n) &\quad\Leftrightarrow\quad \forall \varepsilon > 0\, \exists n_0\, \forall n > n_0 : \left|\tfrac{f(n)}{g(n)} - 1\right| < \varepsilon
\end{aligned}
$$

1

INTRODUCTION

Search processes are an all-pervasive part of everyday life. For instance, looking for a misplaced wallet or seeking a familiar person in a crowd of people corresponds to a search process. Various types of such processes have also been studied scientifically. Issues as diverse as detecting enemy submarines [38], finding castaways [65], the migratory behavior of people [59], and animal foraging patterns [50, 135] have been considered. In fields such as biophysics, search processes are even studied at microscopic scales [127].

Accordingly, it is of major importance to determine which search strategies are particularly promising. In this context, the foraging behavior of a number of animals has been observed, and an intriguing pattern has been identified. Some species tend to scan the immediate environment thoroughly; if they are unsuccessful, they relocate to another area, where they continue their search [105]. Intuitively, one could say that after some time, it is assumed that the current search environment is unfavorable and another area is probably more promising. For an in-depth elaboration on these search strategies, we refer the reader to [26]. If one interprets the animals' foraging behavior as an algorithm, relocation corresponds to resetting the algorithm and trying again under new initial conditions.

Such a reset is an essential part of many search strategies. Alt et al. [2] and especially Luby et al. [123] pioneered a particular form of such resets known as **restarts**. If a restart is performed, then the current state is reset to the original, initial state; furthermore, all future steps are conducted independently of the past. Simply put, it has been found that in many situations, it is beneficial to terminate the current process and retry from the original configuration. Naturally, this type of restart only makes sense if the process is randomized. In recent years, restarts have received considerable attention in disciplines such as physics (e. g., [25, 53]) and biophysics (e. g., [150]). However, we primarily focus on applications in computer science.

In practical applications, restarts are applied to improve algorithms' performance for a wide range of scenarios. These include classical problems such as the traveling salesperson problem [96], the satisfiability problem [28], the vertex cover problem [63], as well as mixed-integer programming [57]. Restarts are also success-

fully employed in other settings, such as finding the most probable explanation in Bayesian networks [125] or in training algorithms for neural networks [119]. Outside of the field of algorithms, a type of restart is used in network protocols to ensure reliable communication [141]. Moreover, it is also considered to what extent restarts affect the power of proof systems [8, 109].

As can be seen, a wide range of problems can benefit from the use of restarts. It is, therefore, unsurprising that the properties of restarts are intensively studied. From an empirical point of view, a key objective is to answer when and why restarts are beneficial, i. e., lead to improved performance. A popular explanation is that some randomized algorithms exhibit a heavy-tail [75–77]. Informally, such algorithms tend to find a solution quickly, but with a small yet non-negligible probability, they take an exceedingly long time to terminate. Restarts are an effective measure for eliminating the heavy-tailed behavior in such a case.

On the other hand, there is also a large influx of results dedicated to which restart strategy should be implemented. In particular, the SAT community has been active in this regard. The kind of restarts used in modern SAT solvers violates the requirements described above. First, most SAT solvers are capable of transferring learned information to the next run [24, 131, 161]; thus, the runs are not independent of each other. Moreover, many SAT solvers delay an imminent restart if the current run is promising [9]. Restart strategies possessing such and similar characteristics are known as dynamic restarts. Some theoretical considerations on dynamic restarts are presented in [98]. Since this is a different type of restart, we will not go into further detail on dynamic restarts in this work.

Theoretical studies have been devoted, among other things, to the comparison of various restart strategies. Notable strategies include the geometric [176], Luby's universal [123], and the fixed-cutoff strategies [123]. In the latter approach, restarts are always performed after the same number of steps t. This simple strategy turns out to be optimal for a suitable choice of t; in other words, the fixed-cutoff strategy optimizes the expected runtime compared to all other possible restart strategies [123]. However, in order to appropriately determine t, the probability distribution describing the runtime of the algorithm would have to be known. Therefore, alternative restart strategies are often applied in practice.

One such applied strategy (e. g., [91, 136]) is Luby's universal strategy, which, simply put, relies on reluctantly exponentially growing restart times. This strategy comes with several appealing theoretical properties. Among them is the fact that the expected runtime is at most a logarithmic factor worse than the runtime of the

optimal fixed-cutoff strategy [123]. Second, it is a universal strategy, implying that the restart times can be easily determined [123].

Another research topic is the question under which conditions restarts are useful. Since, as described above, the fixed-cutoff strategy is optimal, such analyses are typically performed using the fixed-cutoff strategy. Moreover, runtime distributions are often utilized: Suppose X is a random variable describing an algorithm's runtime behavior and $t \in \mathbb{R}_+$ is a restart time. Van Moorsel and Wolter [129] have shown that using the fixed-cutoff strategy with restart time t has a positive effect on the expected runtime if $E[X] < E[X \mid X - t]$ holds. Here, E represents the expected value operator. Furthermore, it is a naturally occurring question in which algorithms implementing a restart strategy enhances the worst-case runtime. Examples for such analyses can be found in [155] and [62].

Lastly, some authors (e. g., [41, 122, 159]) compare restarts with other established techniques. For example, restarts are compared with parallelization in [159]; the authors show that a parallelized algorithm achieves a super-linear speed-up in the number of processors only if restarts are useful for this algorithm. Another methodology related to restarts are portfolio strategies [22, 74]. In a portfolio, different algorithms for the same problem are executed independently of each other (usually in parallel); whereas in a restart strategy, the same algorithm is executed repeatedly, with the individual runs being independent of each other. Shylo et al. [160] examined the relationship between parallelized portfolios and parallelized restart strategies. They proved that the speedup factor of portfolios compared to restart strategies is bounded by a small constant.

1.1 STRUCTURE OF THIS WORK

This work is organized as follows. Chapter 2 presents the necessary theoretical foundations of complexity theory, probability theory, and restarts.

Chapter 3 then addresses various issues concerning restarts through a complexity theoretical approach. Among other results, it is shown that even if the underlying distribution is fully known, the decision of whether restarts are useful is NP-hard. A number of other relevant questions also turn out to be complexity-theoretically difficult. On the other hand, an efficient approximation algorithm for the best restart time is given. To the best of our knowledge, the complexity-theoretic properties of restarts have not been studied before.

Chapter 4 is dedicated to cases in which the decision of whether restarts are useful or not is easy. In particular, we prove conditions for the usefulness of restarts and the optimal restart strategy. We also show that these conditions can be easily evaluated for many distributions, and, accordingly, the NP-hardness is not an obstacle.

Chapter 5 examines the properties of Luby's universal strategy. As described above, one much-cited benefit of Luby's universal strategy is that the expected runtime is at most a logarithmic factor worse than that of the best fixed-cutoff strategy [123]. In this chapter, however, we show that this statement does not hold in general; at the same time, we state conditions under which it does.

Besides the NP-hardness, one reason why the fixed-cutoff strategy is typically not employed in practice is that one would have to know the runtime distribution in order to determine the optimal restart time. However, it is mostly unknown for so far unsolved instances. In Chapter 6, this problem is handled by predicting the runtime distributions of previously unseen instances through machine learning techniques. Using this predicted distribution and the techniques described in Chapter 4, good restart times can then be determined. This approach is tested on the SAT solver PROBSAT resulting in a significant performance improvement. On instances on which PROBSAT faced the greatest difficulties, the performance is improved by an average speedup factor of more than 50.

Chapter 7 considers a different model. In the examined scenario, the algorithm is operating under a deadline. Thus, a solution has to be found within a certain time limit. In addition, it is assumed that the algorithm is working in parallel. Among other results, a criterion for the optimal fixed-cutoff strategy regarding this model is established, and furthermore, it is proven that increasing the number of utilized processors yields a super-linear benefit in this scenario.

Lastly, Chapter 8 summarizes the main findings and provides an outlook on possible further research directions.

2

PRELIMINARIES

This chapter introduces the theoretical foundations which are used throughout this work. It is separated into four sections. Section 2.1 provides a brief introduction to Boolean formulas and related concepts. In Section 2.2, the essential concepts of computational complexity are covered. Section 2.3 presents the fundamentals of probability theory. Section 2.4 provides an overview of the theory of restarts.

2.1 PROPOSITIONAL LOGIC

Some of the most prominent problems in computer science are based on propositional logic. This section introduces the terms and definitions to the extent necessary for this work. We use the definitions and explanations as provided in [44, 139, 156].

Definition 2.1 ([44])**. Boolean functions** are functions on $\{\texttt{true}, \texttt{false}\}^n$ mapping to $\{\texttt{true}, \texttt{false}\}$, where $n \in \mathbb{N}$ is called the number of variables.

Boolean formulas provide a syntactic description of Boolean functions, which we shall describe in more detail below. A **Boolean variable** can assume the **truth values** `true` and `false`. As is established practice, we identify the truth values `false` with 0 and `true` with 1. Boolean variables and truth values can be combined with **Boolean connectives**. The most commonly used connectives are the logical `and` ($\wedge$), the logical `or` ($\vee$), and the logical `not` ($\neg$). The interpretation of these Boolean connectives is given in Table 2.1.

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	$\neg x$
0	1
1	0

Table 2.1: The definition of the Boolean connectives `and`, `or`, and `not`.

The combination of Boolean variables (and truth values) yields a formula. For the technical definition of a formula, we fix a countable infinite set of Boolean variables $X = \{x_1, x_2, \dots\}$.

Definition 2.2 ([139]). A **(Boolean) formula** F is inductively defined to have one of the following forms:

1. Either of the truth values `true` and `false`.
2. A Boolean variable $x \in X$.
3. $(\varphi \wedge \psi)$ if φ and ψ are Boolean formulas.
4. $(\varphi \vee \psi)$ if φ and ψ are Boolean formulas.
5. $\neg\varphi$ if φ is a Boolean formula.

The set of variables appearing in F is denoted by $\mathrm{Var}(F)$.

Any Boolean formula can be equivalently expressed in certain canonical forms. Among these standard forms, the conjunctive normal form is of particular importance.

Definition 2.3 ([156]). A Boolean formula F is in **conjunctive normal form (CNF)** if it has the following structure:

$$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m.$$

The C_i are Boolean formulas that are called **clauses** and have the following form:

$$C_i = (x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,k}).$$

The **literals** $x_{i,j}$ are either variables or negated variables. If all clauses in F have at most k literals, then F is in **k-conjunctive normal form (k-CNF)**.

Definition 2.2 and Definition 2.3 cover the syntax of a Boolean formula. If all Boolean variables in a formula F are exchanged for truth values, then this leads to a semantic interpretation: The resulting expression is either `true` or `false`. This concept is described more formally in the following definition.

Definition 2.4 ([156])**.** An **assignment** α maps a subset of the variables in X to the set of truth values $\{0, 1\}$. The result of applying α to a Boolean formula F is symbolized as $F\alpha$ and has the following meaning: The variables in F are substituted with truth values according to the mapping provided by α. If $F\alpha = 1$, then α is a **satisfying assignment** for F and, equivalently, α **satisfies** F.

The set of variables being mapped is denoted by $\mathrm{Var}(\alpha)$. We call α a **complete assignment** for F if $\mathrm{Var}(\alpha) = \mathrm{Var}(F)$.

In this work, when the term „assignment“ is used, we implicitly mean a complete assignment. One can classify Boolean formulas based on whether a satisfying assignment exists or not.

Definition 2.5 ([156])**.** A Boolean formula F is **satisfiable** if there is a satisfying assignment for F. Otherwise, F is **unsatisfiable**. The Boolean formula F is a **tautology** if and only if $\neg F$ is unsatisfiable.

2.2 COMPLEXITY THEORY

The field of computational complexity theory studies the inherent complexity of problems. Namely, how many resources (often time or memory space) are required to solve a problem. This section provides an overview of the required concepts. The definitions are standard and can be found in textbooks such as [7, 73, 139].

A lot of results in computational complexity theory are concerned with **decision problems**. A decision problem is a yes-no question for a given input. It is often asked whether the input has some desired property. For example, the question could be whether the input represents a satisfiable Boolean formula.

An alphabet Σ is a non-empty, finite set of symbols. A **language** is a set of strings over an arbitrary alphabet. We will use the terms *language* and *decision problem* interchangeable because the inputs to a decision problem yielding a yes-answer form a language. Moreover, deciding whether a word is part of a language is a typical question concerning languages; consequently, this problem is a decision problem.

The complexity classes discussed in this work are typically defined in terms of Turing machines. In this context, we assume that the reader is already familiar with deterministic Turing machines; otherwise, we refer to a standard resource such as [139]. It is nevertheless sufficient to specify a suitable algorithm at an abstract level

for a given problem for our purposes. One reason for this stems from the Cobham-Edmonds Thesis (see, e. g., [73]). Broadly speaking, this thesis states that the time complexities of every "reasonable and general" computational model differ by, at most, a polynomial factor. As a result, algorithms can be expressed in a practically arbitrary manner; the Cobham-Edmonds Thesis ensures that a Turing machine with a comparable runtime exists. In this chapter, we use the term algorithm as a specification of a deterministic Turing machine.

We first focus on algorithms computing Boolean functions. In other words, algorithms whose output is either 1 or 0. For an algorithm $\mathcal{A}$, the result of $\mathcal{A}$ is denoted by $\mathcal{A}(x)$ where $x \in \Sigma^*$ is some valid input string. The algorithm $\mathcal{A}$ **accepts** x if $A(x) = 1$, otherwise if $A(x) = 0$, then $\mathcal{A}$ **rejects** x. If, furthermore, $\mathcal{A}$ either accepts or rejects every input, i. e., $\mathcal{A}$ always terminates after a finite number of steps, then $\mathcal{A}$ **decides** some language.

Usually, the question is asked in the reverse direction: Is there an algorithm deciding a given language? This issue gives rise to the following definition.

Definition 2.6 ([162])**.** A decision problem L is **decidable** if and only if there is an algorithm deciding L.

Even if a language L is decidable, the associated algorithm might have a running time, which makes deciding L practically infeasible. Consequently, the question is often limited to whether a language can be decided *efficiently*. Most commonly, **polynomial-time algorithms** are used to classify efficiently decidable languages. In contrast, we refer to problems that cannot be solved in polynomial time as *difficult*. A polynomial-time algorithm terminates within a polynomial number of steps w. r. t. the length of its input.

Definition 2.7 ([162])**.** The complexity class **P** contains all languages decidable by a polynomial-time algorithm.

While manifold decision problems have polynomial-time algorithms, there are other problems seemingly not admitting polynomial-time algorithms. For some of the most interesting problems, it is (probably) difficult to decide whether the input has the desired property. However, given a proof, it is easy to verify its validity. This characteristic leads to the definition of the famous complexity class NP.

Definition 2.8 ([7]). A language L is in the complexity class **NP** if there is a polynomial $p : \mathbb{N}_0 \to \mathbb{N}_0$ and a polynomial-time algorithm $\mathcal{A}$ such that for every input string x,

$$x \in L \Leftrightarrow \mathcal{A} \text{ accepts } (x, c) \text{ for some string } c \in \Sigma^{p(|x|)}.$$

Here, $|x|$ denotes the length of the string. Any c with $\mathcal{A}(x, c) = 1$ is called **solution**.

A related concept to showing the presence of some property is proving its absence. We can assume the absence of a feature if all possible proofs fail to show the feature's existence. This idea results in the definition of the complexity class coNP.

Definition 2.9 ([7]). A language L is in the complexity class **coNP** if there is a polynomial $p : \mathbb{N}_0 \to \mathbb{N}_0$ and a polynomial-time algorithm $\mathcal{A}$ such that for every input string x,

$$x \in L \Leftrightarrow \mathcal{A} \text{ accepts } (x, c) \text{ for all strings } c \in \Sigma^{p(|x|)}.$$

Here, $|x|$ denotes the length of the string x.

So far, three relevant complexity classes are introduced. At the same time, we have not discussed which problems are in these classes. Broadly speaking, complete problems are considered the most important members of a complexity class. In some respect, describing a complete problem for a particular complexity class also provides insights into the nature of the whole complexity class.

At the core of the definition of completeness is a concept called reducibility. A **reduction** transforms one problem into another insomuch that deciding the second problem helps decide the first problem. We use several kinds of reductions in this work. Subsequently, the notions of completeness and polynomial-time reductions are formally introduced.

Definition 2.10 ([7, 162]). A function $f : \Sigma^* \to \Sigma^*$ is a **polynomial-time computable function** if there is a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}(x)$ calculates $f(x)$.

Functions transforming instances of one decision problem into instances of another decision problem are of particular interest.

Definition 2.11 ([162])**.** A language A is **polynomial-time reducible** to a language B if and only if there is a polynomial-time computable function f such that for every $x \in \Sigma^*$:

$$x \in A \Leftrightarrow f(x) \in B.$$

The function f is called **polynomial-time reduction**.

For many well-known complexity classes, there are problems such that each problem in the complexity class can be reduced to those problems. This property is called hardness.

Definition 2.12 ([139])**.** Let $\mathcal{C}$ be a complexity class and fix a type of reducibility. A decision problem L is **$\mathcal{C}$-hard** (w. r. t. to the type of reduction) if every decision problem $L' \in \mathcal{C}$ in $\mathcal{C}$ is reducible to L with the fixed type of reduction. If L is $\mathcal{C}$-hard and lies in $\mathcal{C}$, then L is **$\mathcal{C}$-complete** (w. r. t. to the type of reduction).

Whether a decision problem is complete for a complexity class may depend on the used notion of reductions. The type of reducibility is mentioned for each hardness and completeness result. Initially, only polynomial-time reductions are utilized. Later on, other reduction types are gradually introduced and applied.

Many complexity classes have complete problems based on propositional logic. One of the most prominent of those problems is called the satisfiability problem. Its decision version is defined as follows:

SAT

Input: A Boolean formula F in conjunctive normal form

Question: Is F satisfiable?

We also use several related problems: In the K-SAT problem, the inputs are limited to k-CNF formulas, and the UNSAT problem asks whether the input is unsatisfiable. Cook [43] famously proved the NP-completeness of SAT. The NP-completeness of 3-SAT is shown by Karp [97]. Finally, as a corollary, UNSAT is coNP-complete.

Theorem 2.13 ([7, 43, 97])**.** SAT *and* K-SAT *with* $k \geq 3$ *are* NP*-complete w. r. t. to polynomial-time reductions.* UNSAT *is* coNP*-complete w. r. t. polynomial-time reductions.*

The complexity classes covered so far are concerned with decision problems. Sometimes, however, one is not only interested in whether a given Boolean formula is satisfiable but also in the number of satisfying assignments. This setting yields a

generalization known as counting problems. For SAT, the corresponding counting problem is therefore defined as follows.

#SAT

Input: A Boolean formula F in conjunctive normal form

Question: What is the number of satisfying assignments for F?

This concept of generalizing to counting problems is captured by the complexity class #P, initially introduced by Valiant [168].

Definition 2.14 ([7])**.** A function $f : \Sigma^* \to \mathbb{N}_0$ is in the complexity class **#P** if there is a polynomial $p : \mathbb{N}_0 \to \mathbb{N}_0$ and a polynomial-time algorithm $\mathcal{A}$ such that for every input string x,

$$f(x) = \left|\left\{c \in \Sigma^{p(|x|)} \mid \mathcal{A} \text{ accepts } (x, c)\right\}\right|.$$

Here, $|x|$ denotes the length of the string x.

The standard definition of polynomial-time reductions has to be slightly amended such that the number of solutions is either preserved or at least easily recoverable.

Definition 2.15 ([45])**.** Let $f, h : \Sigma^* \to \mathbb{N}$ be functions. The function f is **weakly parsimonious polynomial-time reducible** to h if and only if there are two polynomial-time computable functions $g_1 : \Sigma^* \to \Sigma^*$ and $g_2 : \mathbb{N} \to \mathbb{N}$ such that $f(x) = g_2\Big(h\big(g_1(x)\big)\Big)$. The reduction is **parsimonious** if g_2 is the identity function.

The #SAT problem described above is a #P-complete problem.

Theorem 2.16 ([139])**.** #SAT *is* #P*-complete w. r. t. parsimonious polynomial-time reductions.*

As can be noticed, #P is conceptually different from the complexity classes introduced so far; this is because #P contains functions, while NP and coNP include decision problems. Despite these underlying differences, a complexity class containing decision problems can be derived from #P. In order to define this complexity class, the concept of algorithms with oracle access has to be introduced.

Definition 2.17 ([162])**.** Let B be a language. An algorithm $\mathcal{A}$ with **oracle access** to B can query a subroutine deciding B. The algorithm $\mathcal{A}$ is allowed to query the subroutine multiple times with possibly differing inputs. For the time complexity, executing the subroutine counts as only one computational step.

The class of languages decidable in polynomial-time by an algorithm with oracle access to B is denoted by P^B.

The definition of oracles can easily be generalized from languages to functions. Then, the subroutine returns the value of the queried function. Specifically, we use the complexity class $\mathrm{P}^{\#\mathrm{SAT}}$ (or equivalently: $\mathrm{P}^{\#\mathrm{P}}$), i. e., the class of problems decidable by a polynomial-time algorithm with oracle access to #SAT. To comprehend how difficult the problems in a complexity class like $\mathrm{P}^{\#\mathrm{SAT}}$ are, it is again helpful to consider complete problems. To this end, we introduce the notion of Cook reducibility.

Definition 2.18 ([139])**.** A language A is **Cook reducible** to a language B if and only if there is a polynomial-time algorithm M with oracle access to B such that M decides A.

Defining a lexicographic order of assignments in Boolean formulas yields a natural complete problem for $\mathrm{P}^{\#\mathrm{SAT}}$.

LEXICAL K-TH SAT

Input: An integer k and a Boolean CNF formula G with variables $x_1, \ldots, x_n$.

Question: Is $x_n = 1$ in the k-th satisfying assignment of G?

Toda [165] showed that LEXICAL K-TH SAT is complete under polynomial-time reductions. However, we only require Cook completeness for our purposes.

Theorem 2.19 ([165])**.** LEXICAL K-TH SAT *is* $\mathrm{P}^{\#\mathrm{P}}$*-complete w. r. t. Cook reduction.*

So far, we have considered problems over all possible inputs. However, sometimes it is desirable to consider only inputs meeting specific structural properties. For example, it might be of interest to restrict oneself to SAT-formulas with at most one satisfying assignment[1]. For the complexity-theoretical considerations, one may then assume that only inputs of this kind are given. This concept is captured by promise problems introduced in [54].

Definition 2.20 ([73])**.** A pair of disjoint languages $(Q_{\mathrm{YES}}, Q_{\mathrm{NO}})$ is a **promise problem**. The **promise** is the union $Q_{\mathrm{YES}} \cup Q_{\mathrm{NO}}$. Let $\mathcal{A}$ be an algorithm. If

$$\mathcal{A}(x) = \begin{cases} 1, & x \in Q_{\mathrm{YES}} \\ 0, & x \in Q_{\mathrm{NO}} \end{cases},$$

1 This particular problem is known as USAT whose complexity is studied in more detail in [169].

then $\mathcal{A}$ decides the promise problem $(Q_{\text{YES}}, Q_{\text{NO}})$. The algorithm may have an arbitrary behavior on inputs not in $Q_{\text{YES}} \cup Q_{\text{NO}}$.

Promise problems are implicitly used in several decision problems. For example, restricting the input to 3-CNF formulas is technically a promise. However, such a promise can be considered trivial since it is easy to verify whether an input string fulfills the requirement. We, therefore, only use the term promise problem if the promise is not easily verifiable (or more precisely: it is not known whether the promise can be easily checked).

A promise problem $(Q_{\text{YES}}, Q_{\text{NO}})$ is hard for a complexity class $\mathcal{C}$, if every language L with $Q_{\text{YES}} \subseteq L$ and $Q_{\text{NO}} \subseteq \overline{L}$ is hard for $\mathcal{C}$. Therefore, a polynomial-time reduction f from a $\mathcal{C}$-hard problem L' to $(Q_{\text{YES}}, Q_{\text{NO}})$ with $x \in L' \Leftrightarrow f(x) \in Q_{\text{YES}}$ and $x \notin L' \Leftrightarrow f(x) \in Q_{\text{NO}}$ implies the $\mathcal{C}$-hardness of $(Q_{\text{YES}}, Q_{\text{NO}})$.

Many decision problems can be extended to an optimization problem. Thereby, the goal is to minimize a certain objective function. For example, finding the assignment with the minimal number of unsatisfied clauses in a given CNF formula constitutes an extension of SAT to an optimization problem.

Definition 2.21 ([139])**.** An **optimization problem** is a 3-tuple (I, F, f) such that the following conditions hold.

- I is the set of valid instances.
- $F(i)$ is the (non-empty) set of feasible solutions for each instance $i \in I$.
- The cost function f maps each pair (i, s) with $i \in I, s \in F(i)$ to a non-negative number called the cost.

For an instance $i \in I$, an **optimal solution** is an $x \in F(i)$ such that

$$f(i, x) = \min_{s \in F(i)} f(i, s)$$

holds. The cost of an optimal solution is denoted by $\text{OPT}(i)$.

If the optimization version of SAT would be solvable in polynomial-time, then this would imply P = NP. Hence, efficient exact algorithms are unknown for many well-known optimization problems. Therefore, in practical applications, it is often attempted to solve the given optimization problem as well as possible. Algorithms that provide a certain guaranteed solution quality are of particular importance. These kinds of algorithms are known as approximation algorithms.

Definition 2.22 ([171])**.** Let (I, F, f) be an optimization problem and let $\delta \geq 1$ be a real number. An algorithm $\mathcal{A}$ is a **δ-approximation algorithm** if and only if $\mathcal{A}(i)$ returns a feasible solution $x \in F(i)$ with

$$f(i, x) \leq \delta \cdot \mathrm{OPT}(i)$$

for every instance $i \in I$.

An especially valuable special case are approximation algorithms that can both approximate arbitrarily well and have a polynomial runtime in the approximation factor. The most useful type of such algorithms is known as fully polynomial-time approximation schemes.

Definition 2.23 ([171])**.** Let (I, F, f) be an optimization problem. Let $\mathcal{A}$ be an algorithm taking an instance $i \in I$ and a real number $\varepsilon > 0$ as input such that $\mathcal{A}$ is a $(1+\varepsilon)$-approximation algorithm for every fixed ε. The algorithm $\mathcal{A}$ is a **fully polynomial-time approximation scheme (FPTAS)** if the number of computational steps is polynomially bounded w. r. t. both $1/\varepsilon$ and the size of the input instance i.

2.3 PROBABILITY THEORY

This section briefly introduces the groundwork of probability theory and the concepts which are used throughout this work. All definitions in this section are standard and can be found in textbooks, such as [55, 56, 128, 148]. We also refer to these books for a more detailed introduction to the field of probability theory.

The **sample space** Ω describes every possible outcome of a random experiment. It is required that Ω is a non-empty set.

Definition 2.24. Let Ω be an arbitrary sample space. A family $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ of subsets of Ω is a **σ-algebra** if and only if $\mathcal{F}$ is non-empty and closed under complements and countable unions. In other words:

1. Ω is in $\mathcal{F}$;
2. $A \in \mathcal{F} \implies \overline{A} \in \mathcal{F}$; and
3. $A_1, A_2, \cdots \in \mathcal{F} \implies \bigcup_{i \geq 1} A_i \in \mathcal{F}$.

We call an element from a σ-algebra an **event**. Implicitly, we frequently use the σ-algebra $\mathcal{B}(\mathbb{R})$ of the real numbers which is a **Borel σ-algebra**. Broadly speaking, a Borel σ-algebra is the smallest σ-algebra containing all open subsets of the real numbers. The exact definition of Borel σ-algebras is immaterial for this work. We refer the interested reader to standard literature of measure theory, such as [35].

Definition 2.25. Let $\mathcal{F}$ be a σ-algebra on a sample space Ω. A function $\Pr\colon \mathcal{F} \to \mathbb{R}$ is a **probability measure** on $\mathcal{F}$ if and only if

1. for every $A \in \mathcal{F}$: $0 \leq \Pr(A) \leq 1$;
2. $\Pr(\Omega) = 1$; and
3. for any countable sequence of pairwise disjoint sets $A_1, A_2, \dots \in \mathcal{F}$

$$\Pr\Big(\bigcup_{i\geq 1} A_i\Big) = \sum_{i\geq 1} \Pr(A_i)$$

holds.

These concepts suffice to specify probability spaces.

Definition 2.26. A **probability space** is a triple $(\Omega, \mathcal{F}, \Pr)$ such that:

1. Ω is an arbitrary sample space;
2. $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ is a σ-algebra on Ω; and
3. $\Pr$ is a probability measure on $\mathcal{F}$.

Let $(\Omega, \mathcal{F}, \Pr)$ be a probability space and $A \in \mathcal{F}$ be an event. The value $\Pr(A)$ describes how probable the event A is. Hence, we will often refer to a probability measure as **probability** or **(probability) distribution**.

Regarding the σ-algebra $\mathcal{F}$, we will encounter two cases in this thesis. If the sample space Ω is countable, then the power set $\mathcal{P}(\Omega)$ suffices for our purposes. On the other hand, if Ω is uncountable, then the power set $\mathcal{P}(\Omega)$, generally, does not admit useful probability measures [172]. Therefore, the power set is an impractical choice for the σ-algebra in this case. This problem can be resolved by using a Borel σ-algebra. For our purposes, all sets of interest are in the corresponding Borel σ-algebra and appropriate probability measures can be defined. We will not go into further detail regarding this issue as it is beyond this thesis's scope. We refer, for example, to [148] for a detailed discussion of this topic. In summary, we use the power set as σ-algebra whenever the sample space is countable and a Borel σ-algebra otherwise.

Definition 2.27. A function $F\colon \mathbb{R} \to [0,1]$ is a **cumulative distribution function (cdf)** if and only if

1. F is non-decreasing;
2. F is right-continuous;
3. and $\lim_{x\to-\infty} F(x) = 0$ as well as $\lim_{x\to\infty} F(x) = 1$.

In a few cases, we allow the use of **improper** cdfs. Such functions satisfy conditions 1 and 2, but violate condition 3.

For example, an improper cdf is appropriate when describing an algorithm that never terminates with a certain probability. The improper cdf F is used to describe the probability that this algorithm terminates after a certain number of steps. But since the algorithm never terminates with some probability, $\lim_{x\to\infty} F(x) \neq 1$ holds.

Probability spaces and cumulative distribution functions are closely related when the sample space is $\mathbb{R}$. To be precise, let $(\mathbb{R}, \mathcal{B}(\mathbb{R}), \Pr)$ be a probability space with the Borel σ-algebra and define a function F with

$$F(x) = \Pr\big((-\infty, x]\big), \forall x \in \mathbb{R}, \tag{2.1}$$

then it can be verified that F is a cdf. In other words, if a probability space is defined on $\mathbb{R}$, then a cdf can be readily constructed. In this work, however, we often work with given cdfs. That said, given a cdf F, it is not challenging to construct a probability space such that Equation (2.1) holds. For information about constructing such a probability space, we refer to, e. g., Section 2.5 of [148] for further reading.

In some cases, the sample space is not comprised of numeric values; however, numeric values can be assigned to each possible event from the sample space. For example, the sample space associated with a (random) algorithm might be any possible trajectory through the search space. However, if one is not concerned with the specific trajectory, but only with the required steps, the number of required steps can be assigned to each specific trajectory. Random variables capture this concept.

Definition 2.28. A **random variable** X on a sample space Ω is a function such that $X\colon \Omega \to \mathbb{R}$. The associated function $F_X\colon \mathbb{R} \to [0,1]$ with

$$F_X(x) = \Pr(X \leq x),$$

is the **cdf of X**. Here, $X \leq x$ represents the set $\{a \in \Omega \mid X(a) \leq x\}$.

In this work, random variables are denoted by capitalized Latin letters as a convention. If the meaning is evident in the context, then the subscript of the associated cdf F_X is often omitted. We call a random variable **positive** if its image does not contain any negative numbers. A random variable X is a **continuous random variable** if and only if the cdf F_X is continuous. On the other hand, X is a **discrete random variable** if and only if the image of X is countable. In practice, a discrete random variable's image could be any countable subset of the real numbers. In this work, however, we refer to a random variable with the whole numbers (or a subset of them) as image whenever the term discrete random variable is used. As a convention, the notation $X = x$ for $x \in \mathbb{R}$ symbolizes the set $\{a \in \Omega \mid X(a) = x\}$. We use analogous conventions for $X \leq x$ and $X > x$. An important function related to discrete random variables is the probability mass function.

Definition 2.29. Let X be a discrete random variable and $I = \{x_1, x_2, \dots\} \subset \mathbb{R}$ be the image of X. The function $p_X \colon I \to [0,1]$ with

$$p_X(x) = \Pr(X = x), \forall x \in I$$

is the **probability mass function (pmf)** of X.

The cdf and the pmf are linked to each other. Let X be a discrete random variable with cdf F_X and pmf p_X. For every x the value $F_X(x)$ is given by

$$F_X(x) = \Pr(X \leq x) = \sum_{z \leq x} p_X(z).$$

In case the cdf F_X is an improper cdf, we call the pmf p_X **improper** as well. A definition like Definition 2.29 is not useful for continuous random variables: Let X be a continuous random variable with cdf F_X. If there is an $x \in \mathbb{R}$ with $\Pr(X = x) > 0$, then this implies that F_X is not continuous, and X would not be a continuous random variable. Therefore, $\Pr(X = x) = 0$ for all $x \in \mathbb{R}$. Instead, a different concept is necessary for continuous random variables.

Definition 2.30. Let X be a continuous random variable with cdf F_X. If there is a function $f_X \colon \mathbb{R} \to \mathbb{R}$ with

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\,\mathrm{d}t, \forall x \in \mathbb{R},$$

then f_X is the **probability density function (pdf)** of X.

Similarly to the cdf, the subscript of a pmf or pdf can be omitted when the meaning is clear from the context. We sometimes refer to the support of a distribution, where the support describes which events from the sample space may actually occur.

Definition 2.31 ([164])**.** Let X be a random variable. If X is discrete with pmf p, then the **support** is defined by

$$\{x \in \mathbb{R} \mid p(x) > 0\}.$$

If X is continuous with pdf f, then the support is

$$\{x \in \mathbb{R} \mid f(x) > 0\}.$$

Stochastic independence is another important concept in probability theory. In general terms, two events are independent if they do not influence each other. This is a beneficial property for proofs.

Definition 2.32. Let $X_1, \ldots, X_k$ be random variables. They are **independent** if and only if

$$\Pr\left(\bigcap_{i=1}^{k}(X_i \leq x_i)\right) = \prod_{i=1}^{k}\Pr(X_i \leq x_i)$$

for all possible values of $x_1, \ldots, x_k$.

If random variables are not independent, then they are **dependent**. The knowledge that two or more random variables have the same distribution is also a powerful property for proofs. In practice, this is the case if an experiment is repeated under the same conditions.

Definition 2.33 ([37])**.** Let $X_1, \ldots, X_k$ be random variables with respective cdfs $F_{X_1}, \ldots, F_{X_k}$. If and only if

$$F_{X_1}(x) = F_{X_i}(x), \forall x \in \mathbb{R} \text{ and } i \in \{1, \ldots, k\}$$

then $X_1, \ldots, X_k$ are **identically distributed**.

Sometimes, a few typical values are used to characterize a probability distribution. The expected value is among the most important of these values. One reason for

this is that if one were to conduct the same experiment independently and with identical conditions repeatedly, then the observed mean value of that experiment would converge toward the expected value in the long run.

Definition 2.34. Let X be a random variable. The **expected value** $E[X]$ is defined as follows:

1. If X is a discrete random variable with pmf p_X and image $\{x_1, x_2, \dots\}$, then $E[X]$ is given by

$$E[X] := \sum_{i=1}^{\infty} x_i p_X(x_i).$$

   The expected value $E[X]$ is finite iff $\sum_{i=1}^{\infty} |x_i| p_X(x_i)$ converges.

2. If X is a continuous random variable with pdf f_X, then $E[X]$ is given by

$$E[X] := \int_{-\infty}^{\infty} t f_X(t) \, \mathrm{d}t.$$

   The expected value $E[X]$ is finite iff $\int_{-\infty}^{\infty} |t| f_X(t) \, \mathrm{d}t$ converges.

Another reason is that the expected value has many desirable properties, such as linearity.

Lemma 2.35 ([128])**.** *Consider a random variable X as well as two constants $a, b \in \mathbb{R}$. Then,*

$$E[a \cdot X + b] = a \cdot E[X] + b$$

holds.

In many cases, two events are dependent from each other. In other words, the occurrence of one of the events influences the probability of the occurrence of the second event. To analyze these probabilities in more detail, conditional probabilities are necessary.

Definition 2.36. Let E_1 and E_2 be events such that $\Pr(E_2) > 0$. The term

$$\Pr(E_1 \mid E_2) = \frac{\Pr(E_1 \cap E_2)}{\Pr(E_2)}$$

is the probability that E_1 happens given that E_2 happened. The term $\Pr(E_1 \mid E_2)$ is called **conditional probability**.

In addition, conditional expectations are also relevant for our purposes. These are expected values based on the assumption that a particular event has already occurred. In this work, we often use conditions such as $X < t$ for the conditional probability and the conditional expectation. To be precise, in this work, there are two regularly recurring conditional expectations.

Definition 2.37. Let X be a positive random variable. If X is a continuous random variable with cdf F and pdf f, then the conditional expectation $E[X - x \mid X > x]$ is given by

$$E[X - x \mid X > x] = \frac{\int_x^\infty (t - x) f(t) \, \mathrm{d}t}{1 - F(x)}$$

for all $x \geq 0$. Otherwise, if X is a discrete random variable with cdf F and pmf p, then $E[X - x \mid X > x]$ is given by

$$E[X - x \mid X > x] = \frac{\sum_{t>x} (t - x) p(t)}{1 - F(x)}$$

for all $x \in \mathbb{N}_0$ with $x \geq 0$. The conditional expectation $E[X - x \mid X > x]$ is sometimes called **mean residual life**.

For example, let X be a random variable which models the runtime (in minutes) of an algorithm. Assume that the algorithm was started five minutes ago, then the mean residual life $E[X - 5 \mid X > 5]$ represents how much longer the algorithm will run on average. A related concept is the truncated expectation.

Definition 2.38. Let X be a positive random variable. If X is a continuous random variable with cdf F and pdf f, then the conditional expectation $E[X \mid X \leq x]$ is given by

$$E[X \mid X \leq x] = \frac{\int_0^x t \cdot f(t) \, \mathrm{d}t}{F(x)}$$

for all $x \geq 0$. Otherwise, if X is a discrete random variable with cdf F and pmf p, then $E[X \mid X \leq x]$ is given by

$$E[X \mid X \leq x] = \frac{\sum_{t \leq x} t \cdot p(t)}{F(x)}$$

for all $x \in \mathbb{N}_0$ with $x \geq 0$. We call the conditional expectation $E[X \mid X \leq x]$ **truncated expectation**.

For the last part of this section, we introduce some of the most prominent distribution families repeatedly used throughout this thesis. We start with the location-scale family. Broadly speaking, these are distributions that can be transformed into each other by means of shifting and stretching.

Definition 2.39 ([121])**.** Let X be a random variable with cdf F_X. Furthermore, consider two real numbers $a \in \mathbb{R}_+$ and $b \in \mathbb{R}$. The random variable $Y = a \cdot X + b$ has the cdf

$$F_Y(x) = F_X\left(\frac{x-b}{a}\right).$$

Then, a is called the **scale parameter**, and b is the **location parameter** of Y.

Exponential distributions are highly relevant for modeling waiting processes and lifetimes [95, 108]. Since algorithm runtimes are essentially waiting times, it is worth considering the exponential distribution as a candidate in this context.

Definition 2.40 ([95])**.** A continuous random variable X with cdf

$$F(x) = \begin{cases} 1 - e^{-\lambda \cdot x}, & x \geq 0 \\ 0, & \text{else} \end{cases}$$

has an **exponential distribution** with parameter $\lambda \in \mathbb{R}_+$. The pdf f is given by:

$$f(x) = \begin{cases} \lambda \cdot e^{-\lambda \cdot x}, & x \geq 0 \\ 0, & \text{else.} \end{cases}$$

Next, we move on to the lognormal distribution. Just as the well-known normal distribution is a natural description for the sum of independent, identically distributed random variables, the lognormal is a suitable distribution for the product of

independent, identically distributed random variables [47]. Furthermore, the lognormal distribution has been employed to describe the empirical runtime distributions of a number of algorithms (see, e. g., [167]).

Definition 2.41 ([60])**.** A random variable X with pdf

$$f(x) = \begin{cases} \frac{1}{x \cdot \sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{(\ln x - \mu)^2}{2 \cdot \sigma^2}}, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

is **lognormal distributed** with parameters $\sigma \in \mathbb{R}_+$ and $\mu \in \mathbb{R}$. The cdf F of X is given by

$$F(x) = \begin{cases} \frac{1}{2} + \frac{1}{2} \cdot \operatorname{erf}\left(\frac{\ln x - \mu}{\sqrt{2} \cdot \sigma}\right), & x > 0 \\ 0, & x \leq 0 \end{cases}$$

where erf is the error function $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \cdot \int_0^x \exp\left(-t^2\right) \mathrm{d}t$.

We call σ the shape parameter. The lognormal distribution also has a scale parameter, which we derive below. Let X be a lognormal distributed random variable with arbitrary shape $\sigma_X \in \mathbb{R}_+$ and $\mu_X = 0$. Further, let Y be a lognormal distributed random variable with shape $\sigma_Y = \sigma_X$ and an arbitrary $\mu_Y \in \mathbb{R}_+$. Consider the value of the cdf F_X of X at $\frac{x}{\exp \mu_Y}$:

$$\begin{aligned} F_X\left(\frac{x}{\exp(\mu_Y)}\right) &= \frac{1}{2} + \frac{1}{2} \cdot \operatorname{erf}\left(\frac{\ln \frac{x}{\exp(\mu_Y)}}{\sqrt{2} \cdot \sigma_X}\right) \\ &= \frac{1}{2} + \frac{1}{2} \cdot \operatorname{erf}\left(\frac{\ln x - \mu_Y}{\sqrt{2} \cdot \sigma_Y}\right) = F_Y(x) \end{aligned}$$

Thus, e^{μ} satisfies the definition of a scale parameter. Since this value only depends on μ, we sometimes also refer to μ as a scale parameter of the lognormal distribution.

Another distribution that will be used throughout this thesis is the Weibull distribution. This type of distribution is a natural choice for random variables of the form $Y = \min(Y_1, Y_2, \ldots, Y_k)$ due to the Fisher-Tippett-Gnedenko theorem [104]. The Weibull distribution also plays a role in empirical evaluations of algorithms (see, for example, [20]).

Definition 2.42 ([149]). A random variable X with pdf

$$f(x) = \begin{cases} \frac{k}{a} \cdot \left(\frac{x}{a}\right)^{k-1} \cdot e^{-\left(\frac{x}{a}\right)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

is **Weibull distributed** with parameters $k \in \mathbb{R}_+$ and $a \in \mathbb{R}_+$. The cdf F of X is given by

$$F(x) = \begin{cases} 1 - e^{-\left(\frac{x}{a}\right)^k}, & x \geq 0 \\ 0, & x < 0. \end{cases}$$

As can be deduced from the cdf, a is the scale parameter, and furthermore, we call k the shape parameter of the Weibull distribution. The Weibull distribution contains the exponential distribution as a special case for $k = 1$.

The generalized Pareto distribution is an exceptionally flexible distribution, which contains, among others, the exponential, the uniform, and (if a location parameter is introduced) the Pareto distribution. Finally, the generalized Pareto distribution is presented. This distribution is well suited to model extreme events (cf. [18, 143]). Moreover, it is an extremely flexible distribution type that can represent both a finite and an unbounded support. Several important distribution types are also included as special cases in the generalized Pareto distribution. Examples are the exponential, the uniform, and (if a location parameter is introduced) the Pareto distribution.

Definition 2.43 ([95]). A random variable X is **generalized Pareto distributed** if and only if its pdf is given by

$$f(x) = \begin{cases} \frac{1}{a} \cdot \left(1 + \frac{k \cdot x}{a}\right)^{-\frac{1}{k}-1}, & k \neq 0 \\ \frac{1}{a} \cdot e^{-\frac{x}{a}}, & k = 0 \end{cases}$$

with $x \in [0, \infty)$ for $k \geq 0$ and $x \in [0, -\frac{a}{k}]$ for $k < 0$. Otherwise, $f(x) = 0$ holds. The pdf f has two parameters $k \in \mathbb{R}$ and $a \in \mathbb{R}_+$. The cdf F of X is given by

$$F(x) = \begin{cases} 1 - \left(1 + \frac{k \cdot x}{a}\right)^{-\frac{1}{k}}, & x \geq 0 \\ 0, & x < 0. \end{cases}$$

We also write X is **GP-distributed** as an abbreviation for X is generalized Pareto distributed.

Similar to before, a is the scale parameter, and k is the shape parameter of the generalized Pareto distribution.

2.4 FOUNDATION OF RESTARTS

This section lays the theoretical groundwork regarding restarts for the rest of this work.

A **Las Vegas algorithm** (see [10, 132]) is a randomized algorithm that always gives a correct solution. However, the time until a result is found can be treated as a random variable.

Alt et al. [2] first introduced the concept of restarts, which was subsequently refined and analyzed by Luby et al. [123]. Unless stated otherwise, the definitions and results are from the work of Luby et al. [123]. The notation is slightly adapted for this work.

Hereafter let $\mathcal{A}$ be some Las Vegas algorithm and let x be any valid input string. A sequence of numbers $\mathcal{L} = (t_1, t_2, \dots)$ with $t_i \in \mathbb{R}_+ \cup \{\infty\}$ defines a modified algorithm $\mathcal{A}_\mathcal{L}$ as shown in Algorithm 2.1. The algorithm $\mathcal{A}$ restarts in line 3. Each iteration of the for-loop (line 2 to 5) is called a **run** of $\mathcal{A}$.

Algorithm 2.1 $A_\mathcal{L}$: The restarted version of $\mathcal{A}$ with restart strategy $\mathcal{L}$.

Input: Input x and restart strategy $\mathcal{L} = (t_1, t_2, \dots)$.

```
1: procedure A_L(x)
2:     for i ∈ {1, 2, ...} do
3:         Run A(x) for t_i time-units.                ▷ (Re-)start A
4:         if A(x) terminated successfully then
5:             Exit.
```

Definition 2.44 ([123])**.** A **restart strategy** is a sequence $\mathcal{L} = (t_1, t_2, \dots)$ if $\mathcal{L}$ is used to modify an algorithm $\mathcal{A}$ on input x as in Algorithm 2.1. The members $t_i \in \mathbb{R}_+ \cup \{\infty\}$ of the sequence $\mathcal{L}$ are **restart times**. The restart strategy $\mathcal{L}$ is a **universal strategy** if and only if $\mathcal{L}$ neither depends on the algorithm $\mathcal{A}$ nor on the input x.

The most straightforward, non-trivial restart strategy is given when all restart times are identical.

Definition 2.45 ([123]). A restart strategy $\mathcal{L} = (t_1, t_2, \dots)$ with $t_i = t_1$ for all $i \in \mathbb{N}$ is a **fixed-cutoff strategy**.

Note that t_1 uniquely determines a fixed-cutoff strategy. It should be stressed that each run uses new random choices and no information from prior runs is carried over to a new run. Put differently, each run is independent of all preceding runs. When studying restart strategies, the question arises whether there is some kind of optimal strategy. To this end, we use a probability distribution to describe the runtime behavior of a Las Vegas algorithm on a fixed instance.

Definition 2.46 ([90]). Examine a Las Vegas algorithm $\mathcal{A}$ on some valid input x. Let $\Pr(T_{\mathcal{A}(x)} \leq t)$ be the probability that $\mathcal{A}(x)$ successfully finds a solution within t time-units. The probability measure induced by a random variable $X : \Omega \to \mathbb{R}$ is the **runtime distribution** of $\mathcal{A}(x)$ if and only if

$$F_X(t) = \Pr(T_{\mathcal{A}(x)} \leq t), \forall t \in \mathbb{R}_+,$$

where F_X is the cdf of X. Then, the expected value $E[X]$ is called **expected runtime** of $\mathcal{A}(x)$.

We use the following convention for the rest of this work. Let X be a random variable that describes the runtime distribution of Las Vegas algorithm $\mathcal{A}$ on some valid input x. If $\mathcal{L}$ is a restart strategy, then $X_{\mathcal{L}}$ describes the runtime distribution of $\mathcal{A}$ using restart strategy $\mathcal{L}$. We will write X_t to denote the modified runtime distribution in the particular case of $\mathcal{L}$ being a fixed-cutoff strategy with restart time t.

Naturally, there are different interpretations of what constitutes an optimal restart strategy. For example, the expected runtime, the median runtime, or a low probability for a high runtime all constitute possible points of interest w. r. t. the performance of a restart strategy. The expected runtime is used as the key performance indicator for most of this work.

Luby et al. [123] showed that for every runtime distribution, there is a fixed-cutoff strategy that minimizes the expected runtime. In other words, the fixed-cutoff strategy is optimal w. r. t. to the expected runtime. We provide a sketch of the proof because their result is essential for this thesis. In the first step, the expectation is calculated.

Theorem 2.47 ([129]). *Let X be a random variable with cdf F such that X describes the runtime distribution of a randomized algorithm $\mathcal{A}$ on some valid input x. Then, a restarted*

version $\mathcal{A}_t$ which uses a fixed-cutoff strategy $(t, t, \dots)$ induces a random variable X_t. The expected runtime $E[X_t]$ is given by

$$E[X_t] = \frac{1 - F(t)}{F(t)} t + E[X \mid X \leq t].$$

Proof sketch. Our proof sketch differs from [129].

We argue that the total running time X_t is a random sum of identically distributed random variables. That is,

$$X_t = \sum_{i=1}^{N} Y_i, \tag{2.2}$$

where N is an integer-valued random variable and $Y_1, \dots$ are identically distributed random variables. In that case, the well-known Wald's equation [175] can be applied which states:

$$E[X_t] = E[N]E[Y_1]. \tag{2.3}$$

Let t be the restart time. We define a random variable Y_i which describes the contribution of the i-th run to the total running time of $A_t(x)$. With probability $F(t)$ the run finds a solution within t time-units. Otherwise, $\mathcal{A}(x)$ is restarted, and the next run commences. Therefore, Y_i can be expressed as $\min(X, t)$. The expectation $E[Y_i]$ is $\big(1 - F(t)\big) \cdot t + F(t) \cdot E[X \mid X \leq t]$. Note that all Y_i are identically distributed.

Define a random variable N which describes the number of runs until $\mathcal{A}_t(x)$ finds a solution. By definition, all runs are independent, i. e., each run has the same probability of finding a solution within t time-units. This implies that $\mathcal{A}(x)$ finds a solution with probability $F(t)$ within any run given that it did not find a solution in a previous run. Otherwise, $\mathcal{A}(x)$ is restarted. The probability that exactly k runs are necessary is therefore

$$\Pr(N = k) = \big(1 - F(t)\big)^{k-1} F(t).$$

Then, it is easy to verify that the expected value $E[N]$ is given by $\frac{1}{F(t)}$. Hence, the condition from Equation (2.2) applies, and the expected runtime $E[X_t]$ is given by Equation (2.3):

$$E[X_t] = E[N]E[Y_1] = \frac{1 - F(t)}{F(t)} t + E[X \mid X \leq t].$$

This completes the proof sketch.

The expected runtime can be written differently if the random variable is discrete.

Corollary 2.48 ([123]). *Let X be a discrete random variable over $\mathbb{N}_0$ and let $(t, t, \dots)$ be a fixed-cutoff strategy. Then, the expected runtime $E[X_t]$ is given by*

$$E[X_t] = \frac{1}{F(t)} \left(t - \sum_{x<t} F(x) \right),$$

where F is the cdf of X.

Proof. Let p be the pmf of X. We start by showing an auxiliary identity:

$$\sum_{x=0}^{t} x \cdot p(x) = \sum_{x=1}^{t} x \cdot p(x) = \sum_{x=1}^{t} \sum_{i=1}^{x} p(x) = \sum_{i=1}^{t} \sum_{x=i}^{t} p(x) \tag{2.4}$$
$$= \sum_{i=1}^{t} F(t) - F(i-1) = t \cdot F(t) - \sum_{x<t} F(x).$$

A rearrangement of the sums was used in Equation (2.4). We now proceed to show the corollary.

$$\frac{1}{F(t)} \left(t - \sum_{x<t} F(x) \right) = \frac{1}{F(t)} \left((1 - F(t))t + t \cdot F(t) - \sum_{x<t} F(x) \right)$$
$$= \frac{1 - F(t)}{F(t)} t + \frac{\sum_{x=0}^{t} x \cdot p(x)}{F(t)} = \frac{1 - F(t)}{F(t)} t + E[X \mid X \leq t] \tag{2.5}$$

In the last step, the definition of the truncated expectation is used (Definition 2.38). Thus, Equation (2.5) matches the result from Theorem 2.47. This completes the proof. □

We continue by stating that any restart strategy's expected runtime is a convex combination of expected runtimes of fixed-cutoff strategies. Luby et al. [123] showed this property for discrete distributions. This property is, however, also valid for continuous distributions. The proof is presented in detail in Chapter 5.

Lemma 2.49. *Let X be any continuous or discrete random variable and let $\mathcal{L} = (t_1, t_2, \dots)$ be an arbitrary restart strategy. Let $E[X_{t_i}]$ denote the expected value of X with the fixed-cutoff strategy $(t_i, t_i, \dots)$. Then, $E[X_{\mathcal{L}}]$ can be expressed by*

$$E[X_{\mathcal{L}}] = \sum_{k=1}^{\infty} a_k E[X_{t_k}],$$

where $(a_1, a_2, \dots)$ is a sequence with $a_k \in \mathbb{R}_+, \forall k \in \mathbb{N}$ and $\sum_{k=1}^{\infty} a_k = 1$.

This Lemma can be used to show that there is an optimal fixed-cutoff strategy. This crucial property is shown by Luby et al. [123] for discrete runtime distributions. Pal and Reuveni [138] extended the result to continuous distributions.

Theorem 2.50 ([123, 138])**.** *Let X be any positive continuous or discrete random variable. Let $\mathcal{L} = (t_1, t_2, \dots)$ be any restart strategy. There is a fixed-cutoff strategy $(t, t, \dots)$ such that*

$$E[X_t] \leq E[X_{\mathcal{L}}].$$

Proof. Let $s \in \mathcal{L}$ be a restart time in $\mathcal{L}$ with the minimal expected runtime, i. e.,

$$E[X_s] \leq E[X_\ell], \forall \ell \in \mathcal{L}.$$

We use Lemma 2.49 to obtain the desired result:

$$E[X_{\mathcal{L}}] = \sum_{k=1}^{\infty} a_k \cdot E[X_{t_k}] \geq \sum_{k=1}^{\infty} a_k \cdot E[X_s] = E[X_s].$$

Put differently, the expected runtime $E[X_s]$ of the fixed-cutoff strategy $(s, s, \dots)$ is less or equal to $E[X_{\mathcal{L}}]$. □

In other words, the result of Theorem 2.50 states that we can limit ourselves to fixed-cutoff strategies when investigating optimal restart strategies w. r. t. the expected runtime. Furthermore, it suffices to study the best restart time because fixed-cutoff strategies are unambiguously defined by a single restart time. Of course, it is generally of interest whether there is a restart time $t < \infty$ improving the expected runtime. This notion brings us to the definition of useful restarts.

Definition 2.51. Let X be any positive continuous or discrete random variable. Restarts are **useful** for X if and only if there is a fixed-cutoff strategy $(t, t, \dots)$ with $t \in \mathbb{R}_+$ such that $E[X_t] < E[X]$ holds.

The question under which conditions restarts are useful is examined from several perspectives in the following chapters. An important condition for determining whether restarts are useful has already been stated by Moorsel and Wolter [129]. The condition indicates that restarts are useful if, after the algorithm has run for a certain time, the mean residual life is longer than the expected runtime at the algorithm's initialization. This intuition is formalized in the following theorem.

Theorem 2.52 ([129])**.** *Let X be a positive, real-valued random variable, then restarts are useful if there is a restart time $t \in \mathbb{R}_+$ with*

$$E[X] < E[X - t \mid X > t].$$

This condition is necessary and sufficient if the expected values $E[X - t \mid X > t]$ as well as $E[X]$ exist.

If one knows that restarts are useful, then naturally, the question arises as to what constitutes the best restart time; in other words, the restart time minimizing the expected runtime. This issue is formalized in the following definition.

Definition 2.53. Let X be any positive continuous or discrete random variable. An extended real number $t \in \mathbb{R}_+ \cup \{\infty\}$ is an **optimal restart time** if

$$E[X_t] \leq E[X_\ell], \forall \ell \in \mathbb{R}_+ \cup \{\infty\}.$$

Here, we chose the extended real numbers since this definition also covers cases in which restarts are not useful. These cases are handled via the special case $t = \infty$ by defining $E[X_t] = E[X]$. Furthermore, it should be pointed out that occasionally the restart times of the optimal restart strategy converge to zero. In such a situation, we sometimes refer to zero as the optimal restart time.

3

THE COMPLEXITY OF RESTARTS

Parts of this chapter appeared in the following publications:

Jan-Hendrik Lorenz.
On the complexity of restarting.
In: *International Computer Science Symposium in Russia*. Springer, 2019, pp. 250–261.
This work received the best student paper award of the conference.

Jan-Hendrik Lorenz and **Uwe Schöning**.
Promise Problems on Probability Distributions.
In: *Complexity and Approximation*. Springer, 2020, pp. 57–66.

It is generally accepted that a well-designed restart strategy may considerably speed up an algorithm. Consequently, when designing a new algorithm, it is natural to ask whether restarts should be used. Three fundamental questions arise in this context.

1. Is there any restart strategy improving the performance?
2. If yes, what is the optimal restart time?
3. To which extent can the algorithm be accelerated by restarts?

Within the framework of empirical investigations, such questions are answered through estimation of the runtime distribution (see Definition 2.46) by repeated solving of instances (e. g., [6, 64]). We will, however, examine these issues from the perspective of complexity theory.

Knowledge of the runtime distribution is a powerful tool. It allows a succinct representation of the runtime behavior over a large (possibly infinite) support. Consequently, it also seems reasonable to model the runtime behavior for theoretical purposes with cdfs or pmfs.

3.1 FOUNDATIONS

The premise for the following complexity-related considerations is that the underlying probability distribution is known and provided as either cdf or pmf in symbolic form (as explained later in this chapter).

For any of these functions, it is sufficient for them to be well-defined on a finite set $I = \{0, \dots, a\}$ with $0 < a < \infty$. We usually set a to an exponentially large value, e. g., $a := 2^n - 1$ for a $n \in \mathbb{N}$, meaning that every element in I requires at most n bits. Additionally, there are numerous non-standard functions that might help to express an arbitrary function succinctly. This implies that the more operations are permitted, the harder the complexity-theoretical problems will be. For that reason, the only allowed operations are addition, subtraction, multiplication, and integer division.

So far, it has been clarified which characteristics the input has to satisfy, but it is still necessary to specify in which form a pmf, or respectively a cdf is passed as input. For this purpose, we are utilizing the notion of straight-line programs. Arithmetical circuits [174] constitute an equivalent model.

Definition 3.1 ([174])**.** Let C be a non-empty set and let B be a finite set of binary operations defined on C. A **straight-line program** S over (C, B) with n input variables $\{x_1, \dots, x_n\}$ and m registers $\{R_1, \dots, R_m\}$ is a finite sequence of instructions $S = (s_1, s_2, \dots, s_h)$. Every instruction has one of the following structures

$$R_i = R_j \circ R_k$$
$$R_i = c$$
$$R_i = x_\ell,$$

where $i, j, k \leq m$, $\circ \in B$, $c \in C$, and $\ell \leq n$.

A straight-line program S can compute a function $f : C^n \to C$ by initializing the registers with default values and executing its instructions one by one. When the instructions have been processed, register R_1 contains the result.

It should be noted that we consider two types of straight-line programs: Those programs that receive exactly one input variable and those programs that do not receive any input[1]. We also refer to the latter as **constant straight-line programs**.

1 It would also be possible to allow straight-line programs that accept more than one input variable. However, the following complexity-theoretical problems would only get more difficult.

We write $S(i)$ to indicate the value that a straight-line program outputs on input i. If S is a constant straight-line program, we also use S to represent the program's result.

An essential feature for our purposes is the ability to access individual digits of a number. As it turns out, this access can be realized by integer division. This technique is employed and discussed in Remark 3.2. Thus, for the rest of this chapter, we use straight-line programs over $(\mathbb{Z}, \{+, -, \cdot, \div\})$. Here, $\div$ denotes integer division, i. e., if $p = a \cdot q + r$, with $p, a, q, r \in \mathbb{Z}$ and $r < q$, then $p \div q = a$. The power of integer division in straight-line programs and its implications are discussed in a survey paper by Borwein and Hobart [29].

Recall that cdfs map to $[0, 1]$, while our definition of straight-line programs only allows mapping to whole numbers. For this reason, a discretization of the problem is proposed using a two-step approach. At first, we confine ourselves to cdfs mapping from $\{0, \ldots, 2^n - 1\}$ to $[0, 1]$ for $n \in \mathbb{N}$. Furthermore, we work with cdfs of the form $c(i)/M$ as the second restriction. Here, M is a constant natural number, and c is a straight-line program as described above. A discretization is also necessary for pmfs; we proceed analogously for this purpose. That is, we assume a given straight-line program c and a constant M such that $c(i)/M$ defines a pmf on $\{0, \ldots, 2^n - 1\}$. Checking the assumption that the given straight-line programs are indeed describing a cdf or a pmf is by no means trivial. Initially, we use this assumption as a promise, and later, in Section 3.3, the complexity of these promises is investigated more closely.

For the subsequent results, we frequently use the capability to calculate the cdf $F(i)$ for all i in polynomial-time w. r. t. the size of the straight-line program F and $|i|$ as a promise. Furthermore, the results described below typically require a CNF formula encoding as a straight-line program.

Remark 3.2 ([132])**.** Let F be a CNF formula with n variables. For a fixed, arbitrary lexicographic ordering of the variables, there is a straight-line program S and a unique, bijective mapping g from the complete assignments of F to $\{0, \ldots, 2^n - 1\}$ with

$$\alpha \text{ satisfies } F \Rightarrow S\Big(g(\alpha)\Big) = 1, \text{ and}$$
$$\alpha \text{ does not satisfy } F \Rightarrow S\Big(g(\alpha)\Big) = 0.$$

Additionally, is it possible to construct the straight-line program S in such a manner that it can be evaluated in polynomial-time for all $x \in \{0, \ldots, 2^n - 1\}$.

Construction. The technique used to obtain this result is known as arithmetization [162]. In the first step, an inductive construction converting CNF formulas into polynomials with multiple arguments is provided.

If the Boolean formula F is either $F = \texttt{true}$ or $F = \texttt{false}$, then F is associated with a straight-line program S that outputs either $S(0) = 1$ or $S(0) = 0$, respectively. This straight-line program S meets the requirements stated above. A Boolean formula $F = x_i$ consisting of a sole Boolean variable is associated with the polynomial $G(z_i) = z_i$. Furthermore, let F_1 and F_2 be CNF formulas with associated polynomials G_1 and G_2. We can then obtain a polynomial G describing a related CNF formula F as follows:

$$
\begin{aligned}
&F = \neg F_1 \Rightarrow G = 1 - G_1 \\
&F = F_1 \wedge F_2 \Rightarrow G = G_1 \cdot G_2 \\
&F = F_1 \vee F_2 \Rightarrow G = 1 - (1 - G_1)(1 - G_2)
\end{aligned}
$$

We assume that the Boolean formula F consists of n variables. Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be any assignment for F where α_i denotes the assignment of the i-th variable in F. Using the integer representation of truth values, we can also use α as input argument for the associated polynomial G. From the construction of G it is clear that $G(\alpha_1, \ldots, \alpha_n) = 1$ if and only if α is a satisfying assignment for F. Otherwise, i. e., α is not a satisfying assignment, obviously $G(\alpha_1, \ldots, \alpha_n) = 0$ holds.

The polynomial G can be expressed as a straight-line program with n input variables. In the next step, we use a straight-line program as specified by Borwein and Hobart [29]. It is capable of calculating the k-th digit of the binary representation from a natural number $\ell = \sum_{i=0}^{\infty} b_i 2^i$. Each line includes the resulting value in parentheses. After executing the straight-line program, m contains the desired value.

$$
\begin{aligned}
m &= \ell \div 2^{k+1} \left(= \sum_{i=0}^{\infty} b_{k+1+i} 2^i\right) \\
m &= m \cdot 2^{k+1} \left(= \sum_{i=k+1}^{\infty} b_i 2^i\right) \\
m &= \ell - m \left(= \sum_{i=0}^{k} b_i 2^i\right) \\
m &= m \div 2^k (= b_k)
\end{aligned}
$$

For this reason, we can precede G with a supplementary program for transforming a natural number into n binary digits. These n digits act as input for G. The resulting straight-line program evaluates to 1 for some natural number as input if and only if the corresponding Boolean formula is satisfiable. □

3.2 THE COMPLEXITY OF RESTARTS

At the beginning of the chapter, three questions were posed in order to assess restart strategies. Hereafter, we formulate these questions as computational problems and examine their respective complexity.

As a result, both decision and optimization problems arise. It is shown in Section 3.2.1 that the related decision problems are difficult. In Section 3.2.2, possible reasons for this hardness are explored. Finally, we discuss the possibility of whether efficient approximation algorithms for the optimal restart time exist in Section 3.2.3.

3.2.1 *On the Decision to Restart*

In this section, we mainly investigate whether, for a given probability distribution, there is a fixed-cutoff strategy having a favorable effect on the expected runtime. Any of the following problems are introduced and then discussed with a cdf and a pmf-version. The cdf-version is the problem whose input is a function describing a cdf. Likewise, the pmf-versions are problems receiving functions that describe pmfs.

Suppose X is a runtime distribution with cdf F and $(t, \dots)$ is a fixed-cutoff strategy. Recall (Corollary 2.48 and [123]) that the expected runtime with restarts $E[X_t]$ is given by:

$$E[X_t] = \frac{1}{F(t)} \left(t - \sum_{x=0}^{t-1} F(x) \right).$$

Clearly, $E[X_t]$ is bounded from above by $t/F(t)$. For this reason, we will call $t/F(t)$ **the upper bound** in this chapter. The corresponding decision problems are then initially defined w. r. t. this upper bound. At this point, we specify the cdf version of this problem and prove it to be NP-complete. For the following problems, we usually allow the use of improper cdfs (see Definition 2.27).

RESTART-UPPER-BOUND-CDF

Input: A straight-line program c, an integer k, and an integer M.

Promise: The formula $F(i) = \frac{c(i)}{M}$ describes an (improper) cdf on the set $\{0, 1, \ldots, k\}$.

Question: Is there a restart time t such that $\frac{t}{F(t)} < k$?

Theorem 3.3. RESTART-UPPER-BOUND-CDF *is* NP*-hard with respect to polynomial-time reductions.*

Proof. As stated in the promise, the function F is used in this proof, which is defined by $F(i) := \frac{c(i)}{M}$. The statement is shown by a reduction from SAT. Let G be any CNF formula with n variables. According to Remark 3.2, there is a straight-line program S_G such that $S_G(x)$ evaluates to 1 for some $x \in \{0, \ldots, 2^n - 1\}$ if and only if G is satisfiable. Based on S_G, the function $F_G : \mathbb{N} \to [0, 1]$ can be defined as follows:

$$F_G(t) = \begin{cases} \frac{t + S_G(t)}{2^n}, & 0 \leq t \leq 2^n - 1 \\ 1, & t \geq 2^n. \end{cases}$$

It can be observed that F_G represents a cdf because it is monotonically increasing and bounded from above by 1. However, it still has to be shown that a straight-line program can adequately describe F_G. Consider the formula

$$c_G(t) = t + S_G(t) \cdot (1 - b_{n+1}) + b_{n+1} \cdot 2^n, \tag{3.1}$$

where b_{n+1} is the $(n+1)$-st bit of the binary representation of t. All components of c_G can be written as a straight-line program, which can be evaluated in polynomial-time. In addition, $c_G(t)/2^n$ coincides with $F_G(t)$ on the set $\{0, \ldots, 2^{n+1} - 1\}$. As a consequence, a suitable representation has been identified.

In the next step, the behavior of an unsatisfiable CNF formula U is studied. On the set $\{0, \ldots, 2^n - 1\}$, the cdf F_U is then given by $F_U(t) = t/2^n$. Examining the upper bound $t/F_U(t)$, we notice that

$$t/F_U(t) = 2^n$$

holds for all $t \in \{0, \ldots, 2^n\}$ (by defining $t/t = 1$ for $t = 0$). Conversely, let C be a satisfiable CNF formula. Then there exists an $x \in \{0, \ldots, 2^n - 1\}$ such that the

corresponding straight-line program S_C evaluates to 1, i. e., $S_C(x) = 1$. Thus the following applies:

$$\frac{x}{F_S(x)} = \frac{x}{x+1}2^n < 2^n.$$

As a summary, it can be concluded that a CNF formula G is satisfiable only if there is a $t \in \{0, \ldots, 2^n - 1\}$ with $t/F_G(t) < 2^n$. Consequently, RESTART-UPPER-BOUND-CDF is NP-hard w. r. t. polynomial-time reductions. □

Naturally, one question is whether this problem is in NP. In general, there are straight-line programs that cannot be evaluated in polynomial time (an example is given in the proof of Theorem 3.6); thus, one faces the difficulty that the upper bound $t/F(t)$ is generally not computable in polynomial-time. On the other hand, the straight-line program constructed for the reduction from SAT can be evaluated in polynomial-time for all t in the set $\{0, \ldots, 2^{n+1} - 1\}$. Thus, it can be inferred that strengthening the promise yields an NP-complete problem. The resulting problem is called RESTART-UPPER-BOUND-CDF-POLYNOMIAL.

RESTART-UPPER-BOUND-CDF-POLYNOMIAL

Input: A straight-line program c, an integer k, and an integer M.

Promise: The formula $F(i) = \frac{c(i)}{M}$ describes an (improper) cdf on the set $\{0, 1, \ldots, k\}$. Furthermore, $c(i)$ can be evaluated in polynomial-time for all $i \in \{0, 1, \ldots, k\}$.

Question: Is there a restart time t such that $\frac{t}{F(t)} < k$?

Corollary 3.4. RESTART-UPPER-BOUND-CDF-POLYNOMIAL *is* NP*-complete w. r. t. polynomial-time reductions.*

Proof. As stated in the promise, the function F is used in this proof, which is defined by $F(i) := \frac{c(i)}{M}$. The inequality $\frac{i}{F(i)} < k$ can be transformed into $i \cdot M < k \cdot c(i)$. Given that, by assumption, the function c may be evaluated in polynomial-time, it can be concluded that the problem is in NP. The NP-hardness follows directly from Theorem 3.3, and therefore the problem is NP-complete. □

Seeing that the results shown above apply to the upper bound of the expected value under restarts, the question arises whether the exact version is also computationally difficult. At first, we are going to modify the problem specification slightly: The normalization constant M may now be a constant straight-line program.

RESTART-EXACT-CDF

Input: A straight-line program c, an integer k, and a constant straight-line program M.

Promise: The formula $F(i) = \frac{c(i)}{M}$ describes an (improper) cdf on the set $\{0, 1, \ldots, 2k\}$.

Question: Is there a restart time $t \in \{1, \ldots, 2k\}$ such that $E[X_t] < k$? Here, X_t is the random variable induced by the cdf F and the fixed-cutoff strategy defined by t.

For the following polynomial-time reduction, it is necessary to demonstrate how exponentiations of the form 2^K with $K \in \mathbb{N}$ can be represented as a straight-line program.

Lemma 3.5. *Let K and ℓ be natural numbers with $K < 2^\ell$. There is a straight-line program M_ℓ calculating 2^K on input K. The number of instructions in M_ℓ is polynomial in ℓ.*

Proof. This lemma is implicitly given by combining the bit accessing technique used in Remark 3.2 (compare [29]) with the well-known binary exponentiation algorithm [101]. To be precise, the straight-line program M_ℓ emulates the binary exponentiation algorithm's behavior in the following manner.

1. The binary representation $b_\ell \ldots b_1$ of K is calculated as described in Remark 3.2.
2. The variable r is initialized with 1.
3. For each $i \in \{\ell, \ldots, 1\}$ the following is part of M_ℓ:

$$r = r \cdot r$$
$$x = 1 + b_i$$
$$r = r \cdot x$$

In essence, step 3 corresponds to unrolling the binary exponentiation algorithm and replacing its `if`-condition with an arithmetic expression. The correctness of this straight-line program, therefore, stems from that algorithm.

Determining the binary representation as well as calculating the exponential number 2^K requires $O(\ell)$ steps in the straight-line program. Thus the total length of M_ℓ is given by $O(\ell)$. □

This result allows us to address the complexity of RESTART-EXACT-CDF.

Theorem 3.6. *The problem* RESTART-EXACT-CDF *is* NP*-hard w. r. t. polynomial-time reductions.*

Proof. Initially, we proceed analogously to Theorem 3.3. That is, our starting point is a CNF formula G with n variables and the corresponding straight-line program S_G (see Remark 3.2). Let G be any CNF formula with n variables. Then, we can define a function F_G:

$$F_G(t) = \begin{cases} 0, & t = 0 \\ 1 - 2^{-t-S_G(t-1)}, & 1 \leq t \leq 2^n \\ 1 - 2^{-t}, & t > 2^n. \end{cases} \tag{3.2}$$

The function F_G clearly describes a cdf. It is monotonically increasing and bounded from above by 1. However, it is necessary to establish whether an appropriate straight-line program can represent F_G. As a first step, the normalization is discussed. For this purpose, we use a constant straight-line program M which evaluates to $2^{2^{n+1}}$. In accordance with Lemma 3.5, there is a straight-line program M_ℓ, which calculates $2^{2^{n+1}}$ if 2^{n+1} is used as input. A suitable choice of ℓ is, for example, $2n$, in which case the length of M_ℓ is given by $O(n)$. One can then obtain a constant straight-line program M by fixing 2^{n+1} as input.

An important remark is that the program M only has to be specified (and not evaluated) for the reduction. As the length of M is linear in n, and all required information may be derived in polynomial-time, this part of the reduction is doable in polynomial-time. Additionally, the cdf F_G can also be described by $c_G(t)/2^{2^{n+1}}$ in case the function c_G is defined as follows:

$$c_G(t) = \begin{cases} 0, & t < 1 \\ 2^{2^{n+1}} - 2^{2^{n+1}-t-S_G(t-1)}, & 1 \leq t \leq 2^n \\ 2^{2^{n+1}} - 2^{2^{n+1}-t}, & t > 2^n. \end{cases}$$

Now, we consider how to express the term $2^{2^{n+1}-t-S_G(t-1)}$ as a straight-line program. For a fixed n, there is a straight-line program $\mathcal{A}$ computing the binary representation of $2^{n+1} - t - S_G(t-1)$ on input t. Due to Lemma 3.5 and Remark 3.2, the program $\mathcal{A}$ can be designed such that its length is polynomial in n. Since

$2^{n+1} - t - S_G(t-1) < 2^{n+2}$ is valid, Lemma 3.5 can be applied one last time to obtain a straight-line program $\mathcal{B}$, which has polynomial length in n and calculates $2^{2^{n+1}-t-S_G(t-1)}$ on input t. The same reasoning provides a straight-line program $\mathcal{B}'$ of polynomial length calculating $2^{2^{n+1}-t}$ on input t.

With the case distinction technique applied in Equation (3.1), an appropriate straight-line program $\mathcal{A}$ can be constructed such that $\mathcal{A}$ coincides with c_G on the set $\{0, \dots, 2^{n+1}\}$. All necessary components are readily calculable, and thus the program $\mathcal{A}$ may be derived in polynomial-time from the CNF formula G.

Finally, we turn to the expected value under restarts. Consider an unsatisfiable CNF formula U and let F_U be the corresponding cdf as described above. By its definition, F_U resembles the cdf of a geometric distribution [94]. Memorylessness [55] is a fundamental property of geometric distributions. This characteristic already implies that the expected value is exactly 2 for all t in this case. For the sake of completeness, we show this identity without invoking memorylessness.

We use a random variable X_t induced by the cdf F_U and restarts after t steps. This way, for $t \geq 1$, the expected value $E[X_t]$ can be characterized as in Corollary 2.48:

$$\begin{aligned} E[X_t] &= \frac{1}{F(t)} \left(t - \sum_{x=0}^{t-1} F(x) \right) = \frac{1}{1-2^{-t}} \left(t - \sum_{x=0}^{t-1} \left(1 - 2^{-x}\right) \right) \\ &= \frac{1}{1-2^{-t}} \sum_{x=0}^{t-1} 2^{-x} = \frac{2 - 2^{-t+1}}{1-2^{-t}} = 2. \end{aligned} \tag{3.3}$$

On the other hand, consider a satisfiable CNF S and let $t \in \{1, \dots, 2^n\}$ be a natural number with $S_G(t-1) = 1$. Additionally, let F_S be the cdf associated with S as defined in Equation (3.2). Again, the random variable X_t is induced by the cdf F_S and the restart time t. By performing similar calculations as in Equation (3.3), we can bound the expected value $E[X_t]$ from above:

$$E[X_t] \leq 2 \cdot \frac{1-2^{-t}}{1-2^{-t-1}} < 2.$$

In summary, it can be concluded that a CNF formula is satisfiable exactly if there is a restart time $t \in \{1, \dots, 2^{n+1}\}$ with $E[X_t] < 2$. Also, all necessary straight-line programs can be constructed in polynomial-time, making the problem NP-hard. □

In the previous problem, RESTART-UPPER-BOUND-CDF, an additional restriction of the promise implied NP-completeness. This naturally raises the question of whether a similar approach is also applicable to RESTART-EXACT-CDF. However, this seems

unrealistic because the straight-line program (or the respective cdf) constructed for the reduction from SAT cannot be evaluated in polynomial-time. Nevertheless, even if this problem is put aside, the expected runtime under restarts would still have to be calculated, or, alternatively, an adequate certificate for the expected value would have to be found. The problem of computing the expected value is further elaborated in Section 3.2.2.

So far, we have focused on the problems in which the cdf is given. Analogously, one can also examine the problems in which a pmf is given. Accordingly, two modified problems, RESTART-UPPER-BOUND-PMF and RESTART-EXACT-PMF, are introduced. As the problem definitions are similar to the respective cdf versions, we only describe the differences. RESTART-UPPER-BOUND-PMF is identical to RESTART-UPPER-BOUND-CDF, with the only difference that $c(i)/M$ now describes a pmf instead of a cdf. Likewise, RESTART-EXACT-PMF is the pmf version of RESTART-EXACT-CDF, where this time $c(i)/M$ represents a pmf instead of a cdf. On the basis of the results so far, the NP-hardness of these two problems follows.

Corollary 3.7. RESTART-UPPER-BOUND-PMF *and* RESTART-EXACT-PMF *are* NP*-hard.*

Proof. Suppose c is a straight-line program and M is either a constant or a constant straight-line program such that $F(i) := c(i)/M$ is a cdf on a set $\{0, \dots, L\}$. In this case, the function

$$p(i) := \begin{cases} F(0), & i = 0 \\ F(t) - F(t-1), & i > 0 \end{cases}$$

describes a pmf on the set $\{0, \dots, L\}$. This function can be expressed as a straight-line program by using the binary representation of i and the technique from Equation (3.1). Moreover, this program is constructible in polynomial-time, meaning that the problems RESTART-UPPER-BOUND-CDF and RESTART-EXACT-CDF are reducible to RESTART-UPPER-BOUND-PMF and RESTART-EXACT-PMF, respectively. Hence, the NP-hardness follows from Theorem 3.3 and Theorem 3.6. □

3.2.2 *Calculating the Speedup*

While Section 3.2.1 dealt with the question of whether restarts are advisable, this section is about the quality of a restart strategy. For a meaningful assessment of

the quality of restart strategies in terms of the runtime behavior, it is necessary to determine the expected runtime. The primary assumption is that we have a given probability distribution and a fixed-cutoff restart strategy, characterized by the restart time t. Then the question is: What is the expected runtime with restarts? Similar to the previous section, we limit ourselves to a finite set. Equivalently, the partial expectation is used. At the end of the section, we examine the relationship between the partial expectation and expected runtime under restarts more closely.

Definition 3.8 (cf. [179])**.** Let X be a random variable defined over the natural numbers and let p be the pmf of X. For a natural number k, the k-th **partial moment** $\mu_X^{(k)}$ of X is a function such that

$$\mu_X^{(k)}(N) = \sum_{i=0}^{N} i^k p(i).$$

holds for all $N \in \mathbb{N}$. The first partial moment μ_X is also called **partial expectation**.

As in the preceding section, we assume that functions have the form $F(i) = \frac{c(i)}{M}$, where M is a constant natural number and c can be described by a straight-line program. Suppose X is a random variable with cdf F having the structure as described above. In this case, the partial expected value μ_X is a rational number. In order to model the problem using a traditional computational framework, we instead consider $M \cdot \mu_X$ because then the result is a natural number. The corresponding problem can be defined as follows:

PARTIAL-EXPECTATION-CDF

Input: An integer N, a straight-line program c, and an integer M.

Promise: The formula $F(i) = \frac{c(i)}{M}$ describes an (improper) cdf on the set $\{0, 1, \ldots, N\}$.

Question: What is the partial expectation $M\mu_X(N)$, where F defines X?

We now prove the problem to be #P-hard.

Theorem 3.9. PARTIAL-EXPECTATION-CDF *is* #P*-hard with respect to weakly parsimonious polynomial-time reductions.*

Proof. Initially, a characterization of the partial expectation in terms of the cdf is presented. The respective transformations are based on a similar property of the expected value (see, e. g., [145]). Consider a random variable X with cdf F and pmf p,

such that F can be represented by a straight-line program c and a natural number M in accordance with the problem definition. Moreover, let N be an arbitrary natural number. The partial expectation $\mu_x(N)$ can be determined as follows:

$$\begin{aligned}\mu_x(N) &= \sum_{i=0}^{N} i \cdot p(i) = \sum_{i=1}^{N} i \cdot p(i) = \sum_{i=1}^{N}\sum_{k=1}^{i} p(i) = \sum_{k=1}^{N}\sum_{i=k}^{N} p(i) \\ &= \sum_{k=1}^{N} F(N) - F(k-1) = \sum_{k=1}^{N} \frac{c(N) - c(k-1)}{M}. \end{aligned} \tag{3.4}$$

Since both $c(N)$ and $c(k-1)$ are natural numbers, this demonstrates that $M \cdot \mu_X$ is a natural number as well. This equation is a useful tool for reducing #SAT to PARTIAL-EXPECTATION-CDF. Consider a CNF formula G with n variables together with the associated straight-line program S_G. Then a function F_G can be defined as follows:

$$F_G(t) = \begin{cases} 0 & t = 0 \\ \frac{(t-1)+S_G(t-1)}{2^n}, & 1 \leq t \leq 2^n \\ 1, & t > 2^n. \end{cases}$$

Given the fact that F_G is both monotonically increasing and maps to the bounded interval $[0,1]$, F_G is a cdf. It also is easy to describe a straight-line program c (see, for example, Equation (3.1)) such that setting $M = 2^n$ yields $F_G(t) = c(t)/M$.

Furthermore, we set $N = 2^n + 1$ and consider $M \cdot \mu_X(N)$, where X is a random variable induced by F_G. The number of satisfying assignments in G will be referred to as #SAT(G) hereafter. Using Equation (3.4) and the fact that $c(0) = 0$, we then obtain the following value:

$$\begin{aligned} 2^n\mu_X &= \sum_{i=1}^{2^n+1} \Big(c(N) - c(i-1)\Big) = \sum_{i=0}^{2^n} \Big(c(N) - c(i)\Big) = \sum_{i=0}^{2^n} 2^n - \sum_{i=0}^{2^n} c(i) \\ &= \sum_{i=0}^{2^n} 2^n - \sum_{i=1}^{2^n} c(i) = \sum_{i=0}^{2^n} 2^n - \sum_{i=1}^{2^n} \Big((i-1) + S_G(i-1)\Big) \\ &= 2^n(2^n+1) - (2^n-1)2^{n-1} - \sum_{i=1}^{2^n} S_G(i-1) \\ &= 2^n(2^n+1) - (2^n-1)2^{n-1} - \#\mathrm{SAT}(G). \end{aligned} \tag{3.5}$$

To summarize, it is possible to map a CNF formula G to a cdf F_G in polynomial-time such that the partial expectation $\mu_X(N)$ is given by Equation (3.5). By computing $-\left(2^n\mu_X(N) - 2^{2n} - 2^n + 2^{2n-1} - 2^{n-1}\right)$ the number of satisfying assignments in G can be determined. Naturally, this calculation can also be performed in polynomial-time. Consequently, what we described is a weakly parsimonious reduction from #SAT to PARTIAL-EXPECTATION-CDF and, therefore, the #P-hardness is shown. □

If the promise in PARTIAL-EXPECTATION-CDF is reinforced such that only straight-line programs are allowed, which can be evaluated in polynomial-time for all inputs from $\{0, \ldots, N\}$, then the resulting problem is, in fact, #P-complete. The strengthened problem is called PARTIAL-EXPECTATION-CDF-POLY.

Corollary 3.10. PARTIAL-EXPECTATION-CDF-POLY *is* #P*-complete.*

Proof. The proof of the #P-hardness is identical to Theorem 3.9. Hence, it only needs to be shown that the problem is in #P. To characterize $M \cdot \mu_X(N)$, we consider Equation (3.4) once more:

$$M \cdot \mu_X(N) = \sum_{k=1}^{N} c(N) - c(k-1). \tag{3.6}$$

Observe that $c(N) - c(k-1)$ is a positive natural number for all $k \in \{1, \ldots, N\}$. From this equation, we can derive a polynomial-time algorithm $\mathcal{A}$. This algorithm $\mathcal{A}$ uses a straight-line program c, which can be evaluated in polynomial-time, and a natural number N as input. Furthermore, there are two additional inputs k and L, each of which are natural numbers.

If $1 \leq k \leq N$, the algorithm calculates the value $Z := c(N) - c(k-1)$ and accepts precisely if $L < Z$. For all other values of k, the algorithm never accepts. Since c can be evaluated in polynomial-time, $\mathcal{A}$ is a polynomial-time algorithm. Moreover, one can easily verify that the number of accepted words by $\mathcal{A}$ is equal to Equation (3.6).

As a consequence, $M \cdot \mu_X(N)$ conforms to Definition 2.14 and is therefore in #P. Since the problem is also #P-hard, this implies #P-completeness. □

Next, we move on to the pmf version of the problem.

PARTIAL-EXPECTATION-PMF

Input: An integer N, a straight-line program c, a straight-line program d, and an integer M.

Promise: The formula $p(i) \coloneqq \frac{c(i)}{d(i)}$ describes an (improper) pmf on the set $\{0, 1, \ldots, N\}$, i. e., $\sum_{i=0}^{x} p(i)$ is an (improper) cdf.

Question: What is the partial expectation $M\mu_X(N)$, where F defines X.

Note that in contrast to the cdf version PARTIAL-EXPECTATION-CDF, we allow the pmf to consist of the division of two functions. The #P-hardness could be derived from a reduction similar to Corollary 3.7. Nevertheless, we propose a different reduction, which is parsimonious and therefore employs a stronger notion of reducibility.

Theorem 3.11. PARTIAL-EXPECTATION-PMF *is* #P*-hard with respect to parsimonious polynomial-time reductions.*

Proof. Consider a CNF formula G with n variables together with the associated straight line program S_G. Then a function p_G can be defined as follows:

$$p_G(t) = \begin{cases} \frac{S_G(t-1)}{2^n \cdot t}, & 1 \leq t \leq 2^n \\ 0, & \text{else.} \end{cases} \tag{3.7}$$

It can be argued that p_G is indeed a pmf by the observation that $\sum_{\ell=1}^{t} \frac{S_G(\ell-1)}{2^n \cdot \ell}$ is monotonically increasing in t and bounded from above by 1. On the set $\{1, \ldots, 2^n\}$, p_G can evidently be decomposed into the functions $S_G(t-1)$ and $2^n \cdot t$, each of which can be represented by a straight-line program. Furthermore, the partial expectation $2^n \mu_X(2^n)$ is given by:

$$2^n \mu_X(2^n) = 2^n \sum_{i=1}^{2^n} i \cdot \frac{S_G(i-1)}{2^n \cdot i} = \sum_{i=1}^{2^n} S_G(i-1) = \#\mathrm{SAT}(G),$$

where $\#\mathrm{SAT}(G)$ denotes the number of satisfying assignments in G. Consequently, what is described here represents a parsimonious polynomial-time reduction from #SAT to PARTIAL-EXPECTATION-PMF, and therefore the proof is complete. □

The problem definition can easily be extended to cover higher partial moments. A minor adjustment in the above proof yields #P-hardness in each case. If the k-th

partial moment μ_X^k is to be determined, the pmf specified in Equation (3.7) can be adjusted like this:

$$p_G(t) = \begin{cases} \frac{S_G(t-1)}{2^n \cdot t^k}, & 1 \leq t \leq 2^n \\ 0, & \text{else.} \end{cases}$$

The remainder of the proof is unaltered. Thus, the succeeding corollary follows.

Corollary 3.12. *Given the pmf, the computation of the k-th partial moment is* #P*-hard with respect to parsimonious polynomial-time reductions.*

We also discuss under which conditions the problem is in #P. This simply requires that $M \cdot i \cdot c(i)/d(i)$ is a natural number and can be evaluated in polynomial-time. The version of PARTIAL-EXPECTATION-PMF in which these two conditions are added to the promise is called PARTIAL-EXPECTATION-PMF-POLY.

Corollary 3.13. PARTIAL-EXPECTATION-PMF-POLY *is* #P*-complete.*

Proof. This proof is similar to Corollary 3.10. We derive a polynomial-time algorithm $\mathcal{A}$ using two straight-line programs c and d, both of which can be evaluated in polynomial-time, and two natural numbers N and M as input. Two further inputs i and L, which are both natural numbers, are also provided.

Using these inputs, $\mathcal{A}$ calculates the value $Z := M \cdot i \cdot \frac{c(i)}{d(i)}$. In case $i \leq N$, the algorithm will accept the input if and only if $L < Z$, otherwise the input is rejected. The promise guarantees that the result Z is a natural number. Furthermore, the promise ensures that Z can be calculated in polynomial-time. Thus, the partial expectation fulfills Definition 2.14 and is therefore in #P. The #P-hardness follows from Theorem 3.11. □

The link between the partial expectation and the expected runtime under restarts is illustrated in more detail in the following. The calculation of the expected runtime is therefore regarded as a computational problem.

RESTART-EXPECTATION-CDF

Input: A straight-line program c, an integer M and an integer t.

Promise: The formula $F(i) = \frac{c(i)}{M}$ describes an (improper) cdf on the set $\{0, 1, \ldots, t\}$.

Question: What is the expected runtime $E[X_t]$ under restarts, where X_t is defined by the cdf F and the restart time t.

By combining Theorem 2.47 and the definition of the truncated expectation (Definition 2.38), the expected runtime $E[X_t]$ can be represented as follows:

$$E[X_t] = \frac{1 - F(t)}{F(t)} t + E[X \mid X \leq t] = \frac{1 - F(t)}{F(t)} t + \frac{\mu_X(t)}{F(t)}. \tag{3.8}$$

As a consequence, if the cdf F and the partial expected value $\mu_X(t)$ are known, the expected runtime under restarts $E[X_t]$ can be easily calculated. Conversely, Equation (3.8) can also be transformed to $\mu_X(t) = F(t) \cdot E[X_t] - \Big(1 - F(t)\Big) \cdot t$.

As a result, the partial expectation can also readily be reconstructed from the knowledge of the cdf F and the expected runtime. Essentially, we described a kind of reduction[2], which shows that RESTART-EXPECTATION-CDF is as difficult as PARTIAL-EXPECTATION-CDF. A similar kind of reduction can be found for the pmf version of RESTART-EXPECTATION-CDF by converting the cdf F into a pmf p, as in Corollary 3.7.

In conclusion, the calculations needed to assess a restart strategy based on its expected runtime are #P-hard. For this reason, it seems unlikely that there is an effective algorithm for this type of problem.

3.2.3 *Approximating the Restart Time*

Section 3.2.1 analyzed the complexity of decision problems in relation to restarts. Naturally, each of these decision problems has an associated optimization problem. Fundamentally, these problems involve finding a restart time t minimizing either the expected runtime $E[X_t]$ or the corresponding upper bound $t/F(t)$. Furthermore, there is a cdf and pmf version of these problems.

As the related decision problems are NP-hard, the existence of efficient algorithms capable of exactly solving the problems is unlikely. This, however, prompts the question of whether efficient approximation algorithms exist. To be more precise, this section focuses on approximating the **optimal restart time**, meaning the time t that minimizes either the expected runtime $E[X_t]$ or the upper bound $t/F(t)$. At first, we focus on the pmf version and present negative results.

Theorem 3.14. *Let X be a discrete random variable having the (possibly improper) pmf p and additionally consider an arbitrary constant $c > 1$. Unless* P = NP, *there is no polynomial-time algorithm $\mathcal{A}$ printing a restart time t with $E[X_t] < c \cdot \inf_x E[X_x]$ if p is provided as input.*

2 To be precise, this corresponds to a type of reduction called metric reduction, as defined in [106].

Proof. We shall demonstrate that the existence of such an approximation algorithm would imply P = NP. For this purpose, we construct a pmf from a given CNF formula such that one could tell from the result of the algorithm whether a satisfiable or an unsatisfiable formula was given.

Therefore, it is assumed that an algorithm $\mathcal{A}$ with properties according to the theorem exists. For a given CNF formula G and the corresponding straight-line program S_G the following formula $p_G : \mathbb{N} \to [0,1]$ is defined:

$$p_G(i) = \begin{cases} \frac{S_G(i-1)}{2^{n+1}}, & 1 \leq i \leq 2^n \\ \frac{1}{2^{n+1}}, & i = \lceil c \cdot 2^{2n+3} \rceil \\ 0, & \text{else.} \end{cases} \tag{3.9}$$

The function p_G represents a pmf because the associated function $F(x) = \sum_{i=0}^{x} p_G(i)$ is monotonically increasing and maps to the bounded interval $[0,1]$. The following conventions are made for the remainder of this proof. We define $C := \lceil c \cdot 2^{2n+3} \rceil$ and $\beta := \inf_x E[X_x]$.

At first, the case in which an unsatisfiable CNF formula U is given is examined, and we observe its implications for the optimal restart time. Thus, let p_U be the pmf associated with U as defined in Equation (3.9). Put another way, p_U only assumes a non-zero value for C. We denote the associated cdf with F_U, so we have $F_U(x) = \sum_{i=0}^{x} p_U(i)$. The values of F_U can be determined from the definition of the pmf p_U and the property that U is unsatisfiable:

$$F_U(i) = \begin{cases} 0, & 1 \leq i < C \\ \frac{1}{2^{n+1}}, & i \geq C. \end{cases}$$

Apply Corollary 2.48 to determine the expected value under restarts $E[X_t]$. For $t = C$, the expectation assumes the following value:

$$E[X_t] = \frac{1}{F_U(t)} \left(t - \sum_{x<t} F_U(x) \right) = 2^{n+1} \cdot C.$$

It can be easily verified that this is also the optimal expected value or, equivalently, the optimal restart time is $t = C$. On the other hand, in case $t < C$, the expected

value is not well-defined. Consequently, algorithm $\mathcal{A}$ always produces a restart time greater or equal to C if it receives p_U as input.

Secondly, a satisfiable CNF formula H is considered. Again, let p_H be the associated pmf as specified in Equation (3.9), while F_H is the corresponding cdf. We begin by studying the case $t = 2^n$ in more detail. Due to H having at least one satisfying assignment, $F_H(t)$ is at least $1/2^{n+1}$. As in particular $\beta \leq E[X_t]$ holds, the following observation can be deduced:

$$\beta \leq E[X_t] = \frac{1}{F_H(t)} \left(t - \sum_{x<t} F_H(x) \right) \leq 2^{n+1} \left(2^n - \sum_{x<2^n} F_H(x) \right) \leq 2^{2n+1}. \quad (3.10)$$

In the next step, we consider the case $t \geq C$ and examine the respective expectation under restarts $E[X_t]$. More specifically, the objective is finding a lower bound for this expected value.

To this end, it is worth estimating the value of the cdf F_H. The cdf assumes its largest possible value, precisely if H is a tautology. Accordingly, $F_H(t) \leq \frac{2^n+1}{2^{n+1}}$ is valid regardless of the choice of H.

Furthermore, it is also worthwhile to inspect the term $t - \sum_{x<t} F_H(x)$ in more detail. This term is once again at its minimum whenever H is a tautology. For an estimate of the minimum, we decompose the sum $\sum_{x<C} F_H(x)$ into the two partial sums $L_1 := \sum_{x=1}^{2^n} F_H(x)$ and $L_2 := \sum_{x=2^n+1}^{C-1} F_H(x)$. If H is a tautology, then the summand $F_H(x)$ in L_1 is equal to $x/2^{n+1}$ for all x. The result of this arithmetic series can be easily calculated and therefore yields an upper bound $L_1 \leq \frac{1}{4}(2^n + 1)$. On the other hand, in the case of a tautology, the summand in L_2 is constantly $\frac{1}{2}$, and the result is bounded by $L_2 \leq \frac{1}{2}(C - 2^n - 1)$.

As a final preliminary observation, we note that $E[X_t]$ is strictly increasing in t for $t \geq C$. For this reason, $E[X_t] \geq E[X_C]$ holds. Using these preliminary considerations, we are now ready to estimate the expected value $E[X_t]$:

$$\begin{aligned} E[X_t] \geq E[X_C] &= \frac{1}{F_H(C)} \left(C - \sum_{x<C} F_H(x) \right) \geq \frac{2^{n+1}}{2^n+1} \left(C - \sum_{x<C} F_H(x) \right) \\ &\geq \frac{2^{n+1}}{2^n+1} (C - L_1 - L_2) \\ &\geq \frac{2^{n+1}}{2^n+1} \left(C - \frac{2^n+1}{4} - \frac{C - 2^n - 1}{2} \right) \\ &= \frac{2^{n+1}}{2^n+1} \left(\frac{\lceil c \cdot 2^{2n+3} \rceil}{2} + \frac{2^n+1}{4} \right) \geq c \cdot 2^{2n+2}. \end{aligned} \quad (3.11)$$

By applying Equation (3.10) to Equation (3.11), we also obtain $E[X_t] > c \cdot \beta$. Particularly, this means that on input of p_H, algorithm $\mathcal{A}$ prints a restart time t with $t < C$.

To summarize, it follows that SAT can be decided with the assistance of $\mathcal{A}$ in the following manner. First, we construct a pmf as in Equation (3.9) from a given CNF formula G. This pmf serves as input for $\mathcal{A}$. Should the output of $\mathcal{A}$ be greater than $\lceil c \cdot 2^{2n+3} \rceil$, then G is unsatisfiable, otherwise G is satisfiable.

It remains to be stated that constructing the appropriate pmf can be performed in polynomial-time. Such constructions are presented and discussed especially in Section 3.2.1. Since additionally, $\mathcal{A}$ is by assumption a polynomial-time algorithm; it follows that SAT can also be decided in polynomial-time. This is only valid if P = NP. □

Therefore, we demonstrated the existence of such an approximation algorithm for the optimal restart time to be unlikely. One might wonder whether a reason for this is that the function to be optimized is the expected value. However, there is probably also no such approximation algorithm for the upper bound $t/F(t)$, as we shall see.

For this purpose, the following lemma is used, which has been introduced by Luby et al. for discrete distributions. At a later stage, in Lemma 5.6, this result is extended to continuous distributions.

Lemma 3.15 ([123]). *Consider a discrete random variable X defined on the natural numbers together with the corresponding cdf F. Then the following inequality holds:*

$$\inf_{x \in \mathbb{R}_+} \frac{x}{F(x)} \leq 4 \inf_{t \in \mathbb{R}_+} E[X_t].$$

Essentially, this lemma characterizes the quality of the upper bound $t/F(t)$. The minimal value of $t/F(t)$ is, therefore, at most four times as big as the minimal expected runtime under restarts. We now prove the following corollary by means of this property.

Corollary 3.16. *Consider a discrete random variable X together with its pmf p. Define $\alpha := \inf_{t \in \mathbb{R}_+} \frac{t}{F(t)}$ as the minimal value of the upper bound, whereby F corresponds to the respective cdf. Also, let $c > 1$ be an arbitrary constant.*

Unless P = NP, *there is no polynomial-time algorithm $\mathcal{A}$ producing a restart time t with $\frac{t}{F(t)} < c \cdot \alpha$ when p is given as input.*

Proof. Suppose there is an approximation algorithm $\mathcal{A}$ in accordance with the requirements of the corollary. Algorithm $\mathcal{A}$ returns a restart time t in polynomial-time. By definition of $\mathcal{A}$ and by applying Lemma 3.15, the following statement follows:

$$\frac{t}{F(t)} < c \cdot \alpha \leq 4c \inf_{t \in \mathbb{R}_+} E[X_t].$$

This inequality suggests, however, that $\mathcal{A}$ is also a $4c$-approximation algorithm for the optimal restart time with respect to the expected runtime. According to Theorem 3.14, such an algorithm can only exist if P = NP. □

Having discussed the subject for the case of a given pmf thoroughly, we shall now turn our attention to the case where a cdf is provided. In contrast to the previously discussed issues, here efficient, i. e., polynomial-time, algorithms are presented.

We start with the somewhat more restricted problem of finding the optimal restart time w. r. t. the upper bound $t/F(t)$ on a fixed set $\{1, 2, \ldots, k\}$. A viable approach is proposed in Algorithm 3.1. Here and in the following, it is always assumed, as a promise, that the input F is a cdf. Furthermore, here we do not limit ourselves to the assumption that F must be given by a constant and a straight-line program with addition, subtraction, multiplication, and integer division (as discussed at the beginning of this chapter). Instead, F may be from a more powerful model; for example, the straight-line program may allow proper divisions. In other words, the model does not have much influence on the following results.

Algorithm 3.1 An approximation algorithm for the upper bound.

Require: F is a cdf.

```
1:  ApproxRestartTime(F, ε, k):
2:      x ← 1
3:      min_x ← 1
4:      min_value ← x/F(x)
5:      while x < k do
6:          x ← (1 + ε) ·x
7:          if x/F(x) < min_value then
8:              min_x ← x
9:              min_value ← x/F(x)
10:     return min_x
```

The aim is, provided F can be evaluated in polynomial-time, to demonstrate that ApproxRestartTime is an FPTAS. For this purpose, we first analyze the quality of the output in terms of the upper bound.

Theorem 3.17. *For every $\varepsilon > 0$, ApproxRestartTime(F, ε, k) produces a restart time x with $\frac{x}{F(x)} < (1+\varepsilon)\frac{t^*}{F(t^*)}$ where $t^* = \arg\min_{1 \le t < k} \frac{t}{F(t)}$.*

Proof. Examine the loop variable x. At some point during the execution of the algorithm $t^* \le x < (1+\varepsilon)\, t^*$ is valid. As per definition, the cdf F is monotonically increasing, and thus $F(x) \ge F(t^*)$ holds. As a result, the subsequent inequality follows:

$$\frac{x}{F(x)} < (1+\varepsilon)\,\frac{t^*}{F(x)} \le (1+\varepsilon)\,\frac{t^*}{F(t^*)}.$$

This already proves the statement. □

The next step examines the runtime of the algorithm in detail.

Theorem 3.18. ApproxRestartTime *terminates after $\lceil \frac{\ln k}{\ln(1+\varepsilon)} \rceil$ iterations of the loop. Accordingly, it is an FPTAS if F can be evaluated in polynomial-time.*

Proof. Let ℓ be the smallest integer with $(1+\varepsilon)^\ell \ge k$. Via elementary arithmetic, we then obtain $\ell \ge \frac{\ln k}{\ln(1+\varepsilon)}$, and by the definition of ℓ, we have $\ell = \lceil \frac{\ln k}{\ln(1+\varepsilon)} \rceil$. After ℓ iterations, the loop variable x assumes the value $(1+\varepsilon)^\ell \ge k$ and therefore ApproxRestartTime halts. By means of the well-known inequality $\frac{2x}{2+x} \le \ln(1+x)$ (see [120]), the number of iterations can be estimated.

$$\left\lceil \frac{\ln k}{\ln(1+\varepsilon)} \right\rceil \le \left\lceil \left(\frac{1}{\varepsilon} + \frac{1}{2}\right) \ln k \right\rceil.$$

Altogether, the algorithm requires $O(\frac{1}{\varepsilon} \ln k)$ loop cycles, which is linear in both $\frac{1}{\varepsilon}$ as well as in the length of k. Consequently, when restricted to cdfs evaluable in polynomial-time, ApproxRestartTime is an FPTAS. □

ApproxRestartTime thus finds an arbitrarily good approximation of the optimal restart time on $\{1, \ldots, k-1\}$ w. r. t. $\frac{t}{F(t)}$ within a reasonable runtime. Nevertheless, this immediately gives rise to two potential points of criticism or, rather, possibilities for improvement, which are subsequently addressed.

A limitation is, of course, that the algorithm can only find an approximation of the optimal restart time within a specified set. Besides, the estimation of the restart time's quality only applies to the upper bound $t/F(t)$. However, for practical applications, it is desirable for practical applications to obtain a restart time with an expected runtime $E[X_t]$ as short as possible. As a matter of fact, these two restrictions are related and can be resolved simultaneously.

To begin with, suppose $\varepsilon > 0$ and y is a restart time for which

$$\frac{y}{F(y)} \leq (1+\varepsilon) \inf_t \frac{t}{F(t)}$$

is valid. By applying Lemma 3.15, an estimate of the quality of the restart time y w. r. t. the expected runtime is immediately obtained:

$$E[X_y] \leq \frac{y}{F(y)} \leq (1+\varepsilon) \inf_t \frac{t}{F(t)} \leq 4 \cdot (1+\varepsilon) \inf_t E[X_t].$$

Consequently, if a good approximation y to the globally best restart time in terms of $t/F(t)$ is found, then y is also a good approximation for the best restart time with regard to the expected runtime. Moreover, it is evident that ApproxRestartTime finds an approximation to the globally best restart time regarding the upper bound $t/F(t)$ if k is sufficiently large.

In many cases, a priori knowledge of an appropriate value for k is not a reasonable assumption. Hence, the question arises whether another suitable condition can replace the knowledge of k. Note that the value of k is only used for the termination condition in the loop of ApproxRestartTime. Modifying this condition yields the following slightly altered algorithm ApproxRestartTimeExact.

Algorithm 3.2 An approximation algorithm for the expected runtime.

Require: F is a cdf.

```
1:  ApproxRestartTimeExact(F, ε):
2:      x ← 1
3:      min_x ← 1
4:      min_value ← x/F(x)
5:      while x ≤ min_value do
6:          x ← (1 + ε) ·x
7:          if x/F(x) < min_value then
8:              min_x ← x
9:              min_value ← x/F(x)
10:     return min_x
```

We are now able to demonstrate that ApproxRestartTimeExact is a suitable approximation algorithm for the restart time with regard to the expected runtime.

Corollary 3.19. *Let F be a cdf and suppose $y := \arg\min_x \frac{x}{F(x)}$ is an optimal restart time with respect to the upper bound. Moreover, assume the length of the binary representation of $F(y)$ to be bounded from above by $M \in \mathbb{N}$. Then,* ApproxRestartTimeExact(F, ε) *produces a restart time t with $E[X_t] \leq (4+\varepsilon) \inf_x E[X_x]$ for every $\varepsilon > 0$. The algorithm terminates after $O\left(\frac{M+\log_2 y}{\log_2(1+\varepsilon)}\right)$ iterations.*

Proof. First, it is shown that ApproxRestartTimeExact stops after a finite number of loop cycles. To do this, consider the case $x > \min_{\text{value}}$, then

$$\frac{x}{F(x)} > \frac{\min_{\text{value}}}{F(x)} \geq \min_{\text{value}}$$

holds, since the cdf F is bounded by 1. To paraphrase, all $x > \min_{\text{value}}$ are worse restart times (in terms of the upper bound) than $\min_x$. Hence, the algorithm eventually halts. In particular, this also indicates that the reasoning from the proof of Theorem 3.17 is applicable. To reiterate, at some point during the execution of the algorithm $y \leq x \leq (1+\varepsilon) \cdot y$ is valid and therefore

$$\min_{\text{value}} \leq \frac{x}{F(x)} \leq (1+\varepsilon) \cdot \frac{y}{F(y)}$$

follows. As already discussed above, the approximation factor of ApproxRestartTimeExact is then derived from the application of Theorem 3.17 and Lemma 3.15.

It remains to be shown how many iterations the algorithm needs at most. By assumption, the length of the binary representation of $F(y)$ is bounded by M. As a result, we can infer that the value of $F(y)$ is at least 2^{-M} and therefore

$$\frac{y}{F(y)} \leq y2^M$$

is valid. In the first part of the proof, the approximation factor of ApproxRestartTimeExact is shown. From this, it also follows that at some point during the algorithm $\min_{\text{value}} \leq (1+\varepsilon) \cdot y2^M$ holds. The algorithm stops when x exceeds the value of $\min_{\text{value}}$.

Therefore, let $\ell + 1$ be the smallest integer with $(1+\varepsilon)^{\ell+1} > (1+\varepsilon) \cdot y2^M$. Put differently, ℓ is the number of iterations until ApproxRestartTimeExact terminates. Thus the subsequent inequalities follow:

$$\begin{aligned} x = (1+\varepsilon)^{\ell+1} &\geq (1+\varepsilon)y2^M \\ \Leftrightarrow \quad (1+\varepsilon)^{\ell} &\geq y2^M \\ \Leftrightarrow \quad \ell &\geq \frac{M + \log_2 y}{\log_2(1+\varepsilon)}. \end{aligned} \tag{3.12}$$

Since, by assumption, ℓ is the smallest integer for which Equation (3.12) is valid, the algorithm stops after at most $O\left(\frac{M+\log_2 y}{\log_2(1+\varepsilon)}\right)$ loop cycles. □

Notably, Corollary 3.19 is applicable if F is evaluable in polynomial-time. This is because the binary representation of $F(x)$ is then also limited by a polynomial for all x. Moreover, in this particular case, the runtime of ApproxRestartTimeExact is polynomial in both $1/\varepsilon$ as well as in the length of y.

3.3 THE COMPLEXITY OF THE PROMISES

It has been repeatedly assumed that the given functions represent a cdf or a pmf. For a complete and self-contained treatment of the complexity-theoretical aspects of restarts, the complexity of these promises should also be examined in greater detail. We shall formulate both of these preconditions as a decision problem and analyze their respective complexity in detail. As before, the input is assumed to be given by a constant M and a straight-line program c with addition, subtraction, multiplication

and integer division. We then investigate the function F with $F(i) := \frac{c(i)}{M}$ on a set $\{1, \ldots, k\}$. At first, the pmf version of the problem is presented.

PMF

Input: A straight-line program c, an integer M, and an integer ℓ.

Question: Is $\frac{c(i)}{M}$ an (improper) pmf on $\{1, \ldots, \ell\}$?

It is possible to demonstrate the $\mathrm{P}^{\#\mathrm{P}}$-hardness of PMF.

Theorem 3.20. PMF *is* $\mathrm{P}^{\#\mathrm{P}}$*-hard with respect to Cook reductions.*

Proof. Recall that LEXICAL K-TH SAT (see Section 2.2, page 20) is $\mathrm{P}^{\#\mathrm{P}}$-hard in terms of Cook reductions. The statement of the theorem is thus proven by reducing LEXICAL K-TH SAT to PMF. As before, consider a CNF formula G with n variables $x_1, \ldots, x_n$ together with the corresponding straight-line program S_G (see Remark 3.2). For a fixed integer $k \in \mathbb{N}$, the function p_G is defined as follows:

$$p_G(t) = \begin{cases} \frac{1}{k}, & t = 1 \\ \frac{S_G(t-2)}{k}, & t \in \{2, \ldots, 2^n + 1\}. \end{cases} \tag{3.13}$$

Due to the construction of p_G, we find $p_G(t) \geq 0$ holds for all $t \in \mathbb{N}$. Each integer in $\{2, \ldots, 2^n + 1\}$ is in a one-to-one relationship to a complete assignment for G. It follows that p_G constitutes a pmf on $\{1, \ldots, \ell\}$ precisely if there are at most $k - 1$ integers in $\{2, \ldots, \ell\}$ representing a satisfying assignment of G. By means of binary search, the minimal ℓ with $\ell \leq 2^n$ can be found, such that p_G is not a pmf on $\{1, \ldots, \ell\}$.

In this case, the following property is valid for p_G on the set $\{1, \ldots, \ell\}$:

$$\begin{aligned} & \sum_{i=1}^{\ell} p_G(t) > 1 \\ \Leftrightarrow\ & \frac{1}{k} + \sum_{i=2}^{\ell} \frac{S_G(i-2)}{k} > 1 \\ \Leftrightarrow\ & \sum_{i=2}^{\ell} S_G(i-2) > k - 1. \end{aligned}$$

As ℓ is minimal, it can be concluded that $\sum_{i=2}^{\ell} S_G(i-2) = k$ and ℓ corresponds to the k-th satisfying assignment of G. The binary representation of ℓ then determines

if $x_n = 1$. This procedure defines a Cook reduction from LEXICAL K-TH SAT to PMF. PMF is, therefore, #P-hard regarding Cook reductions. □

Suppose we only allow straight-line programs evaluable in polynomial-time and call this version of the problem PMF POLY. This restricted version of the problem is in $\mathrm{P}^{\#\mathrm{P}}$. Moreover, since the reduction uses a straight-line program evaluable in polynomial-time, the $\mathrm{P}^{\#\mathrm{P}}$-completeness of PMF POLY ensues.

Corollary 3.21. PMF POLY *is* $\mathrm{P}^{\#\mathrm{P}}$*-complete with respect to Cook reductions.*

Proof. The $\mathrm{P}^{\#\mathrm{P}}$-hardness is due to Theorem 3.20 because the straight-line program S_G used in Equation (3.13) is evaluable in polynomial-time. All that remains to be demonstrated is that PMF POLY is in $\mathrm{P}^{\#\mathrm{P}}$.

For this purpose, consider a straight-line program c and any fixed, positive integer M. In the following, we use $p(i)$ as shorthand for $p(i) := \frac{c(i)}{M}$. The function p is a pmf if and only if the associated function F with $F(x) = \sum_{i=1}^{x} p(i)$ is a cdf. In order for this to be true, $p(i) \geq 0$, or equivalently $c(i) \geq 0$, needs to be valid for all $i \in \{1, \ldots, \ell\}$. This assertion conforms to the definition of a coNP language, and since $\mathrm{coNP} \subseteq \mathrm{P}^{\#\mathrm{P}}$ (see [166]) holds, this part of the claim is shown. Secondly, F must be bounded from above by 1 on the set $\{1, \ldots, \ell\}$. One may equivalently express this problem by asking whether $\sum_{i=1}^{\ell} c(i) \leq M$ is valid. The required computation can be performed in #P. For this purpose, a construction similar to Corollary 3.10 may be applied.

In summary, a function p is unambiguously identified as a pmf if the associated aggregate function $F(x) = \sum_{i=1}^{x} p(i)$ is monotonically increasing and limited by 1. All necessary operations can be performed in $\mathrm{P}^{\#\mathrm{P}}$, and thus PMF POLY is in $\mathrm{P}^{\#\mathrm{P}}$. This concludes the proof. □

As in the previous sections, we also investigate the cdf version of the problem.

CDF

Input: A straight-line program c, an integer M, and an integer ℓ.

Question: Is F a cdf on $\{1, \ldots, \ell\}$, where $F(i) := \frac{c(i)}{M}$?

Theorem 3.22. CDF *is* coNP*-hard with respect to polynomial-time reductions.*

Proof. We show coNP-hardness by reducing UNSAT to CDF. Assume a CNF formula G with n variables is given. The following function F_G is derived from G and the corresponding straight-line program S_G:

$$F_G(t) = \begin{cases} \frac{1}{2^n+1}, & t = 1 \\ \frac{t}{2^n+1}\Big(1 - S_G(t-2)\Big), & t \in \{2, \dots, 2^n+1\} \\ 1, & t > 2^n + 1. \end{cases}$$

If G is unsatisfiable, then F_G represents the cdf of a uniform distribution on the set $\{1, \dots, 2^n + 1\}$. Otherwise, G is satisfiable. Let $x \in \{0, \dots, 2^n - 1\}$ be the minimal number with $S_G(x) = 1$. By definition, this leads to $F_G(x+2) = 0$ and $F_G(x+1) = \frac{x+1}{2^n+1}$. In particular, $F_G(x+2) < F_G(x+1)$ holds true, but since cdfs must be monotonically increasing, F_G is not a cdf. To summarize, F_G is a cdf precisely if G is unsatisfiable. This completes the proof. □

Similar to the pmf case, the result can be extended by restricting ourselves to straight-line programs, which can be evaluated in polynomial time. The version of CDF with this additional restriction is referred to as CDF POLY.

Corollary 3.23. CDF POLY *is* coNP*-complete with respect to polynomial-time reductions.*

Proof. Let c be a straight-line program together with natural numbers M and ℓ according to the problem definition of CDF POLY. The function F refers to $F(i) := \frac{c(i)}{M}$. For F to be a cdf, it has to be monotonically increasing. This assertion is logically equivalent to $\forall i \in \{2, \dots, \ell\} : c(i) - c(i-1) \geq 0$. Since c can be evaluated in polynomial-time by assumption, this statement can be verified by a coNP machine. Secondly, F must be limited by 1 on the set $\{1, \dots, \ell\}$. A coNP machine can also verify this statement by checking $\forall i \in \{1, \dots, \ell\} : c(i) \leq M$.

It is thus shown that CDF POLY is located in coNP. The coNP-hardness follows from Theorem 3.22 since the straight-line program used for the reduction can be evaluated in polynomial-time. This concludes the proof. □

3.4 SUMMARY

In this chapter, we have analyzed the complexity of issues related to restarts. For this purpose, we assumed two separate models: In one version, the pmf is given; in the second version, the cdf is provided.

In Section 3.2.1, it is investigated whether restarts are advisable. However, since both variations of the problem, i. e., both the cdf and the pmf version, are NP-hard, the existence of an effective decision algorithm seems unlikely. Even if a restart strategy is fixed, the calculations required to determine the expected runtime are expensive. This problem is explored in more detail in Section 3.2.2, and it is demonstrated that the necessary computations are #P-hard. Furthermore, the possibility of the existence of an approximation algorithm for the optimal restart time is discussed in Section 3.2.3. Regarding this topic, a significant distinction between the pmf and the cdf model emerges. While the existence of an efficient approximation algorithm for the pmf version would imply P = NP, for the cdf version, such an algorithm is constructed. One reason for this difference is that the cdf is inherently more structured. In particular, the cdf is monotonically increasing, which is ultimately the property exploited for the construction of the approximation algorithm. In all these problems, it was always assumed that the input is actually a pmf or a cdf. However, as we demonstrated in Section 3.3, these are far from trivial promises, which deserve attention as well.

An unresolved research question is the extent to which the chosen model influences the complexity. Recall that straight-line programs with addition, subtraction, multiplication, and integer division are allowed as input. Furthermore, the straight-line programs should be limited to one input variable each. It appears plausible that many of the problems become considerably easier if integer division is not allowed as long as one keeps the limitation to one input variable.

Another interesting phenomenon is that the pmf version of a problem is often more difficult than the cdf version. Conversely, there are no problems for which the pmf version is (significantly) easier than the cdf version. If F is a cdf, then a reduction to the pmf problem is immediately obtained by $F(t) - F(t-1)$.

Another direction in which one could further develop this issue is generalizing the results to continuous distributions. However, since the usual computational models are inherently discrete, one would have to resort to a model such as the one proposed by Ko [102] for complexity-theoretical considerations. Furthermore, the complexity theoretical investigation of other properties of probability distributions might also be of interest.

One could also consider these issues from the perspective of property testing (e. g., [23, 33]). In contrast to our model, the pmf p and cdf are unknown, instead one has sample access to the underlying distribution. In other words, there is an oracle that returns an element x with probability $p(x)$. For example, in such a framework,

it may be asked whether the underlying pmf is unimodal [34]. In our context, an interesting question would be whether restarts are useful for the underlying distribution.

4

RUNTIME DISTRIBUTIONS AND CRITERIA FOR RESTARTS

Parts of this chapter appeared in the following publications:

Jan-Hendrik Lorenz.
Runtime Distributions and Criteria for Restarts.
In: *arXiv preprint arXiv:1709.10405* (2017).

Jan-Hendrik Lorenz.
Runtime Distributions and Criteria for Restarts.
In: *SOFSEM 2018: Theory and Practice of Computer Science*. Springer International Publishing, 2018, pp. 493–507.
This work received the best student paper award of the conference.

Jan-Hendrik Lorenz.
Restart Strategies in a Continuous Setting.
In: *Theory of Computing Systems* (2021), pp. 1–22.

Florian Wörz and **Jan-Hendrik Lorenz**.
Evidence for Long-Tails in SLS Algorithms.
In: *29th Annual European Symposium on Algorithms (ESA 2021)*. 2021, 82:1–82:16.
This work received the best student paper award of the conference.

The results obtained in Chapter 3 demonstrated that determining an optimal restart strategy is generally challenging. We now move on to investigate cases in which calculating the optimal restart time is feasible. Random variables are frequently associated with a well-established family of random distributions, such as Weibull or lognormal distributions. In such a case, the corresponding cdf depends on only a few parameters, and thus, the NP-hardness results are not a major limitation here.

The findings so far have been obtained for discrete random variables. From now on, we want to focus on another scenario, namely the case where a continuous

distribution is given. The objective is to identify efficient methods for finding the optimal restart strategy.

Modeling the process as a continuous random variable is relevant when, for example, confronted with a scenario in which a computer monitors a real-time system, say transmission times in a network. Such a situation is typical for, e. g., distributed computations. From a theoretical perspective, this setting could also be significant for understanding computational models capable of (at least partially) representing real numbers. Examples of such computational models include timed automatons [3] and general purpose analog computers [158]. The analysis of real-valued restart strategies could also be of interest for the complexity of real functions. For an overview of real-time computational models, we refer to Bournez [30], and for an introduction to the complexity of real functions, we refer to Ko [102].

Restart strategies are also studied in disciplines other than computer science. In this case, continuous processes are typically considered. For example, the manner in which enzymes conduct their search has been investigated [52]. In the examined scenario, enzymes scan a DNA strand for a target. One of the strategies is a relocalization (i. e. a restart) to a different position in the DNA strand. General diffusion processes were also analyzed in the context of restarts [53]. It was demonstrated that random restarts have a positive effect on the expected value under some circumstances.

The primary focus in the works mentioned above, and also in the thesis at hand, is the effect of restarts on the expected waiting time. However, in general, restarts can also be applied to minimize other continuous quantities such as money, gasoline, measurement errors, velocity, and so forth.

Because of the reasons mentioned above, the analysis of restart strategies is also of interest in a continuous model. This chapter is structured as follows. Section 4.1 introduces a necessary and sufficient condition for continuous distributions to decide whether restarts are useful. A condition to identify optimal restart times is then introduced in Section 4.2. The effect of location and scale parameters on the restart properties of distributions is analyzed in Section 4.3. Lastly, Section 4.4 analyzes some typical families of random distributions with respect to restarts.

4.1 EFFECTIVE RESTARTS

The first step before applying a restart strategy is checking under which conditions restarts are useful. For a formal explanation of the usefulness of restarts, refer to Definition 2.51. Moorsel and Wolter [129] developed a condition for the usefulness of restarts (cf. Theorem 2.52). This condition is necessary and sufficient if the expected values $E[X]$ and $E[X - t \mid X > t]$ exist. In this section, another condition for the usefulness of restarts is given, which is also applicable in cases where $E[X]$ or $E[X - t \mid X > t]$ do not exist.

There are several compact descriptions for a dataset. Among the most commonly used measures are the median and the mean. They both indicate a characteristic value for this dataset. If the mean value is considerably greater than the median, this indicates (positively) skewed data. Outliers affect the mean value linearly, whereas the median is robust towards outliers. A substantial gap between median and mean value is therefore explained either by many outliers or by a few but extreme outliers. Restarts are an effective technique for reducing outliers in either case. To make this intuition more precise, we introduce the notion of the quantile function.

Definition 4.1 ([95])**.** Let X be a random variable together with its cumulative distribution function $F_X : \mathbb{R} \to [0, 1]$. Then, the **quantile function** $Q_X : (0, 1) \to \mathbb{R}$ is given by

$$Q_X(p) = \inf\{x \in \mathbb{R} \mid F(x) \geq p\}.$$

In this chapter, we mainly consider such random variables X, such that their pdf f_X is continuous and non-zero on (a, ∞) for some $a \in \mathbb{R}$. As an immediate consequence, the cdf F_X is continuous as well and is furthermore monotonically increasing. In this case, the quantile function Q_X is the inverse of F_X; in other words, $F_X\big(Q_X(p)\big) = p$ holds. If nothing else is stated, we are working with these assumptions. Furthermore, similar to the cdf and pdf, we omit the quantile function's subscript if the meaning is clear from the context.

Consider a positive random variable X having quantile function Q. Then, the median of X is given by $Q(0.5)$. A simple test for the effectiveness of restarts can be conducted by comparing the median to the expected value: If $\frac{Q(0.5)}{E[X]} < 1/2$, then restarts have a positive effect on the expected runtime. For this purpose, recall that $t/F(t)$ is an upper bound for the expected value $E[X_t]$. Choosing $t = Q(0.5)$ yields $E[X_{Q(0.5)}] \leq Q(0.5)/0.5 < E[X]$. In other words, choosing the median as restart time

improves the expected value. It is straightforward to generalize this concept. If the expected value is large due to a small number of disproportionately long runs, but there are also many short runs, restarts improve the performance.

In general, this can be understood as an imbalance between short and long runs. The study of this disparity is a well-known discipline in economics, known as income inequality measurement. An example of this type of measure is the so-called Gini coefficient [100], which summarizes the extent of (income) inequality in one single indicator. Another metric used in the field of income inequality is the Lorenz curve, which will prove helpful for our purposes.

Definition 4.2 ([118])**.** Let X be a real-valued random variable with quantile function Q. Then, the **Lorenz curve** $L : [0,1] \to [0,1]$ is given by:

$$L(p) = \frac{\int_0^p Q(u)\,\mathrm{d}u}{E[X]}.$$

The derivative L' of L is given by $L'(p) = \frac{Q(p)}{E[X]}$. Before we are able to show the necessary and sufficient condition for the usefulness of Restart, we first need to present a characterization of the expected runtime under restarts using the quantile function.

Lemma 4.3. *Let X be a positive, real-valued random variable having quantile function Q. For $p \in (0,1)$, the expected value $E[X_{Q(p)}]$ with restarts after $Q(p)$ time units is given by:*

$$E[X_{Q(p)}] = \frac{1-p}{p} \cdot Q(p) + \frac{\int_0^p Q(u)\,\mathrm{d}u}{p}.$$

Proof. Let F be the cdf and f the pdf of X. According to Theorem 2.47, the expected runtime $E[X_{Q(p)}]$ is given by:

$$E[X_{Q(p)}] = \frac{1-p}{p} Q(p) + E[X \mid X \leq Q(p)].$$

As per Definition 2.38, the truncated expected value $E[X \mid X \leq Q(p)]$ is:

$$E[X \mid X \leq Q(p)] = \frac{\int_0^{Q(p)} x \cdot f(x)\,\mathrm{d}x}{p}.$$

Applying the substitution $u = F(x)$ leads to $\frac{\mathrm{d}u}{\mathrm{d}x} = f(x)$ and $x = Q(u)$. Consequently, the desired result follows via integration by substitution:

$$\begin{aligned} E[X_{Q(p)}] =& \frac{1-p}{p} \cdot Q(p) + E[X \mid X \leq Q(p)] \\ =& \frac{1-p}{p} \cdot Q(p) + \frac{\int_0^{Q(p)} x \cdot f(x)\,\mathrm{d}x}{p} \\ =& \frac{1-p}{p} \cdot Q(p) + \frac{\int_0^{p} Q(u)\,\mathrm{d}u}{p}. \end{aligned} \tag{4.1}$$

□

By means of this lemma, we can now state a necessary and sufficient condition for the usefulness of restarts.

Theorem 4.4. *Let X be a positive, real-valued random variable, then restarts are useful if and only if there is a quantile $p \in (0,1)$ such that*

$$(1-p)L'(p) + L(p) < p.$$

Proof. We start by examining the expected runtime $E[X_{Q(p)}]$ with restarts after $Q(p)$ time units. According to Lemma 4.3, the expectation $E[X_{Q(p)}]$ is given by:

$$E[X_{Q(p)}] = \frac{1-p}{p} \cdot Q(p) + \frac{\int_0^{p} Q(u)\,\mathrm{d}u}{p}. \tag{4.2}$$

This means that restarts are useful precisely if there is a $p \in (0,1)$, such that $E[X_{Q(p)}] < E[X]$ holds (see Definition 2.51). Inserting Equation (4.2) into this inequality yields:

$$\begin{aligned} & \frac{1-p}{p} \cdot Q(p) + \frac{\int_0^p Q(u)\,\mathrm{d}u}{p} < E[X] \\ \Leftrightarrow\ & \frac{1-p}{E[X]} \cdot Q(p) + \frac{\int_0^p Q(u)\,\mathrm{d}u}{E[X]} < p \\ \Leftrightarrow\ & (1-p)L'(p) + L(p) < p. \end{aligned} \tag{4.3}$$

Definition 4.2 is applied in the last line. Inequality (4.3) conforms to the desired result. □

As mentioned at the beginning of this section, Moorsel and Wolter [129] also provided a condition for the usefulness of restarts (see Theorem 2.52). A major difference is that the condition from Theorem 4.4 utilizes quantiles, so only the range $(0,1)$ is of interest. In contrast, the condition from [129] is defined on a, typically, unbounded support. The search for a suitable restart time is thus algorithmically easier to implement with the condition from Theorem 4.4.

It should also be emphasized that the condition from Theorem 4.4 is stronger. This is because the condition from [129] implicitly assumes $E[X] < \infty$, whereas Theorem 4.4 is also applicable in the case $E[X] = \infty$. The following lemma addresses this precise situation.

Lemma 4.5. *Let X be a positive, real-valued random variable together with its quantile function Q, such that $E[X] = \infty$. In this case,*

$$(1-p)L'(p) + L(p) = 0$$

holds for all $p \in (0,1)$ with $Q(p) < \infty$. Here, L is the Lorenz curve, and L' is its derivative.

Proof. Consider an arbitrary $p \in (0,1)$ with $Q(p) < \infty$. As $E[X] = \infty$ is valid by assumption, $L'(p) = 0$ follows immediately. Furthermore, $\int_0^p Q(u)\,\mathrm{d}u \leq p \cdot Q(p) < \infty$ holds true. Thus, $L(p) = 0$ is also valid, which already proves the lemma. □

Lemma 4.5 is, for instance, applicable for certain power-law distributions[1]; for example, the generalized Pareto distribution (Definition 2.43) has this property for some parameter combinations. It is undoubtedly of interest which kind of distributions can benefit from restarts. The exponential distribution represents a boundary case: Wolter [180] showed that the exponential distribution is the only continuous distribution for which restarts are neither useful nor harmful.

In the following, we analyze which classes of distributions satisfy the condition of Theorem 4.4. It is demonstrated that restarts are useful for distributions having the so-called long-tail property.

Definition 4.6 ([61])**.** A positive, real-valued random variable X is **long-tailed**, if and only if

$$\forall x \in \mathbb{R}_+ : \Pr(X > x) > 0 \qquad \text{and} \qquad \lim_{x\to\infty} \frac{\Pr(X > x + y)}{\Pr(X > x)} = 1$$

holds for all $y > 0$.

To prove the desired result, some auxiliary statements are shown. First, a property is specified, implying that the expected value is infinite.

Lemma 4.7 ([61])**.** *Let X be a positive, real-valued random variable and let f be its pdf. If*

$$\liminf_{t\to\infty} t^2 \cdot f(t) > 0 \tag{4.4}$$

holds, then $E[X] = \infty$.

Proof. This statement is implicitly given in [61]. For the purpose of completeness, a proof is presented here. Inequality (4.4) implies $f \in \Omega(t^{-2})$ or equivalently:

$$\exists c > 0\, \exists t_0 > 0\, \forall t > t_0 : c \cdot t^{-2} \leq f(t).$$

This property can be used to estimate the expected value:

$$E[X] = \int_0^\infty x \cdot f(x)\,\mathrm{d}x \geq \int_{t_0}^\infty x \cdot f(x)\,\mathrm{d}x \geq \int_{t_0}^\infty c \cdot x^{-1}\,\mathrm{d}x = \infty.$$

This completes the proof. □

1 For a definition of power-laws, see [128].

Next, a lemma is presented, which essentially states that restarts have neither a positive nor a negative effect if the restart time approaches infinity.

Lemma 4.8. *Consider a positive, real-valued random variable X with pdf f and $E[X] < \infty$. Also, assume that L and L' are well-defined and the limit $\lim_{t\to\infty} t^2 \cdot f(t)$ exists. Then,*

$$\lim_{p\to 1} \Big((1-p) \cdot L'(p) + L(p) \Big) = 1$$

holds.

Proof. First, we consider the limiting value of

$$\lim_{p\to 1} L(p) = \lim_{p\to 1} \frac{\int_0^p Q(u)\,\mathrm{d}u}{E[X]}.$$

By reversing the last step in Equation (4.1) from the proof of Lemma 4.3, we get

$$\lim_{p\to 1} L(p) = \lim_{p\to 1} \frac{\int_0^{Q(p)} x \cdot f(x)\,\mathrm{d}x}{E[X]} = 1$$

because the integral in the numerator represents the expected value. We now consider the limiting value of the term $(1-p) \cdot L'(p) = (1-p)Q(p)/E[X]$. For this purpose, let F be the cdf of X. If X has finite support, i. e., if there is an $x \in \mathbb{R}$ with $F(x) = 1$, then $\lim_{p\to 1}(1-p)Q(p) = 0$ follows immediately. We, therefore, assume that $F(x) < 1$ holds for all $x \in \mathbb{R}$, in other words, $\lim_{p\to 1} Q(p) = \infty$. As $(1-p)$ is approaching 0 for $p \to 1$, the limiting value of $(1-p) \cdot L'(p)$ must be examined more closely. As a preliminary consideration, we note that the derivative Q' of Q is given by $Q'(p) = 1/f\big(Q(p)\big)$. This is because Q is the inverse function to F; the derivative then follows by means of the inverse function theorem [153]. In the following, L'Hospital's rule is applied:

$$\lim_{p\to 1}(1-p) \cdot Q(p) = \lim_{p\to 1} \frac{1-p}{\frac{1}{Q(p)}} = \lim_{p\to 1} Q(p)^2 \cdot f\big(Q(p)\big).$$

We now substitute $p = F(t)$ and apply the definition $Q\big(F(t)\big) = t$:

$$\lim_{p\to 1}(1-p)\cdot Q(p) = \lim_{t\to\infty} Q\big(F(t)\big)^2 \cdot f\big(Q(F(t))\big) = \lim_{t\to\infty} t^2 \cdot f(t).$$

Lemma 4.7 states that $\liminf_{t\to\infty} t^2 \cdot f(t) > 0$ implies that the expected value is infinite. This would contradict the assumption and therefore $\liminf_{t\to\infty} t^2 f(t) = 0$ follows. Since, in addition, the limiting value of $t^2 \cdot f(t)$ exists by assumption,

$$\lim_{t\to\infty} t^2 \cdot f(t) = \limsup_{t\to\infty} t^2 \cdot f(t) = \liminf_{t\to\infty} t^2 \cdot f(t) = 0$$

must be valid. This completes the proof. □

Apart from the pdf and the cdf, the hazard rate function is also frequently used to investigate a random variable's behavior.

Definition 4.9 ([147])**.** Let X be a positive, real-valued random variable having cdf F and pdf f. The **hazard rate function** $r : \mathbb{R}_+ \to \mathbb{R}_+$ of X is given by:

$$r(t) = \frac{f(t)}{1 - F(t)}.$$

There is an intriguing connection between the hazard rate function and the cdf.

Lemma 4.10 ([147])**.** *Consider a positive, real-valued random variable X with hazard rate function r. Then, the cdf F of X can be expressed as follows:*

$$1 - F(t) = \exp\left(-\int_0^t r(u)\,\mathrm{d}u\right).$$

The following lemma characterizes the relationship between the hazard rate function and long-tailed distributions.

Lemma 4.11 ([134])**.** *Let X be a positive, real-valued random variable with hazard rate function r such that the limit $\lim_{t\to\infty} r(t)$ exists. The following three statements are equivalent:*

1. *X is long-tailed .*
2. $\lim_{x\to\infty} \int_x^{x+y} r(t)\,\mathrm{d}t = 0, \forall y > 0.$
3. $\lim_{t\to\infty} r(t) = 0.$

Proof. The proof can be found in [134]. Since the manuscript in question had not been published at the time of writing, we provide a proof (adopted from [134]).

For this proof, let $y \in \mathbb{R}$ be arbitrary with $y > 0$. First, the equivalence of 1 and 2 is shown. In accordance with the definition of long-tail distributions (see Definition 4.6), we investigate the quotient $\frac{\Pr(X>x+y)}{\Pr(X>x)}$ for an arbitrary $y > 0$ and use Lemma 4.10 to obtain:

$$\frac{\Pr(X > x+y)}{\Pr(X > x)} = \frac{\exp\left(-\int_0^{x+y} r(t)\,\mathrm{d}t\right)}{\exp\left(-\int_0^{x} r(t)\,\mathrm{d}t\right)} = \exp\left(-\int_x^{x+y} r(t)\,\mathrm{d}t\right).$$

As an immediate consequence, $\lim_{x\to\infty} \frac{\Pr(X>x+y)}{\Pr(X>x)} = 1$ holds iff $\lim_{x\to\infty} \int_x^{x+y} r(t)\,\mathrm{d}t = 0$ is valid. The next step shows that 3 is implied by 2. This is demonstrated by logical contraposition. For this purpose, $\liminf_{t\to\infty} r(t) > 0$ is assumed or expressed equivalently:

$$\exists C > 0\, \exists x_0 > 0\, \forall x > x_0 : r(x) \geq C.$$

By using this fact, the integral $\int_x^{x+y} r(t)\,\mathrm{d}t$ can be estimated if x is large enough:

$$\forall x > x_0 : \int_x^{x+y} r(t)\,\mathrm{d}t \geq \int_x^{x+y} C\,\mathrm{d}t = y \cdot C > 0.$$

Consequently, $\liminf_{t\to\infty} r(t) = 0$ must be valid for 2 to be true. Additionally, since the limit $\lim_{t\to\infty} r(t)$ exists by assumption the following is observed:

$$\lim_{t\to\infty} r(t) = \limsup_{t\to\infty} r(t) = \liminf_{t\to\infty} r(t) = 0.$$

Finally, we show that 3 implies 2. Since $r(t) \geq 0$ for all $t \in \mathbb{R}_+$, it is noted that $\lim_{t\to\infty} r(t) = 0$ can also be expressed as

$$\forall C > 0\, \exists x_0 > 0\, \forall x > x_0 : r(x) < C.$$

Similar as argued above, this representation can be used to estimate the integral $\int_x^{x+y} r(t)\,\mathrm{d}t$ if x is large enough:

$$\forall x > x_0 : \int_x^{x+y} r(t)\,\mathrm{d}t < \int_x^{x+y} C\,\mathrm{d}t = y \cdot C.$$

As this statement applies to all $C > 0$, we have $\lim_{x\to\infty} \int_x^{x+y} r(t)\,\mathrm{d}t = 0$ as a consequence. This completes the proof. □

We are now ready to prove that restarts are useful for long-tail distributions with reasonable constraints by using these preliminary considerations.

Theorem 4.12. *Consider a positive, long-tailed random variable X with pdf f and hazard rate function r. Also assume that either $E[X] = \infty$ holds or the limits $\lim_{t\to\infty} r(t)$ and $\lim_{t\to\infty} t^2 \cdot f(t)$ both exist. In both cases, restarts are useful for X.*

Proof. In the following, let Q be the quantile function of X and Q' be the derivative of the quantile function Q. Furthermore, the cdf of X is referred to as F. First, we consider the case $E[X] = \infty$. According to Lemma 4.5, $(1-p)\cdot L'(p) + L(p) = 0$ is valid for all $p \in (0,1)$ with $Q(p) < \infty$. This immediately yields $(1-p)\cdot L'(p) + L(p) < p$ for all $p \in (0,1)$ with $Q(p) < \infty$ and thus, following Theorem 4.4, restarts are useful.

We now move on to the second case and assume $E[X] < \infty$ and the existence of the two limits $\lim_{t\to\infty} r(t)$ and $\lim_{t\to\infty} t^2 \cdot f(t)$. First, the derivative of $(1-p)L'(p) + L(p) - p$ is examined more closely. It is worth considering the derivative L'' of L' in advance. Since $L'(p) = Q(p)/E[X]$, and Q is the inverse function to F, this leads to $L''(p) = Q'(p)/E[X] = 1/(f(Q(p)) \cdot E[X])$. This is employed in the following calculation of the derivative:

$$\frac{\mathrm{d}}{\mathrm{d}p}\Big((1-p)L'(p) + L(p) - p\Big) = \frac{1-p}{E[X]\cdot f\big(Q(p)\big)} - 1.$$

The limiting value of this derivative is now considered for $p \to 1$. The substitution $p = F(t)$ can be used; then the corresponding limiting value is obtained for $t \to \infty$:

$$\lim_{p\to 1} \frac{1-p}{E[X]\cdot f\big(Q(p)\big)} - 1 = \lim_{t\to\infty} \frac{1-F(t)}{E[X]\cdot f(t)} - 1 = \lim_{t\to\infty} \frac{1}{E[X]\cdot r(t)} - 1.$$

As X is long-tailed and the limit $\lim_{t\to\infty} r(t)$ exists, Lemma 4.11 may be applied, and therefore $\lim_{t\to\infty} r(t) = 0$ follows. Together with $E[X] < \infty$ this yields:

$$\lim_{p\to 1} \frac{1-p}{E[X]\cdot f\big(Q(p)\big)} - 1 = \lim_{t\to\infty} \frac{1}{E[X]\cdot r(t)} - 1 = \infty. \tag{4.5}$$

According to Theorem 4.4, restarts are useful precisely when

$$(1-p)\cdot L'(p) + L(p) - p < 0 \tag{4.6}$$

is valid. A consequence of Lemma 4.8 is that the left side of this inequality approaches 0 as p approaches 1. In addition, as written at the beginning of the chapter, f is continuous and non-zero. As a consequence, Q is continuous, and thus the functions L and L', which depend on Q, are also continuous. Therefore, the entire left-hand side of Inequality (4.6) is continuous as well.

Simultaneously, according to Equation (4.5), the derivative of the left side of Inequality (4.6) is approaching infinity for $p \to 1$. From this observation and the continuity, we can conclude that there is a $p \in (0,1)$ satisfying the condition from Theorem 4.4. Consequently, restarts are useful for X. □

It should be noted that requiring the existence of the two limit values is not a real limitation. All naturally occurring long-tail distributions either have an infinite expected value, or both limits exist, cf. [134]. To provide context for the theorem, it should be mentioned that this is a sufficient but not a necessary condition. A type of distribution for which restarts are useful, which is not long-tailed, is the hyper-exponential distribution [170].

Thus, when presented with a non-long-tail distribution, Theorem 4.4 must be applied and its condition carefully analyzed. Since the class of long-tailed distributions contains many important distributions (such as Burr [95], lognormal [60], and Lévy distributions [60]), Theorem 4.12 provides a powerful tool for making an immediate statement about the usefulness of restarts.

4.2 OPTIMAL RESTARTS

The previous section established conditions for the usefulness of restarts. Naturally, it is of importance which restart times yield an improvement and, especially, which restart times are optimal. This question is explored in this section. A connection

between optimal restart times and the inverse hazard rate function is presented in [180]. Here, we examine this issue in terms of the optimal restart quantile.

Definition 4.13. Let X be any positive continuous random variable having quantile function Q. A real number $p^* \in (0,1)$ is an **optimal restart quantile** if

$$E[X_{Q(p^*)}] \leq E[X_{Q(p)}], \forall p \in (0,1).$$

Obviously, there is a connection to optimal restart times (cf. Definition 2.53). If $p^* \in (0,1)$ is an optimal restart quantile, then $t = Q(p^*)$ is an optimal restart time. We propose a model to determine the optimal restart quantile, which can be defined purely on the basis of the quantile function.

Theorem 4.14. *Let X be a positive, real-valued random variable with quantile function Q and its derivative Q'. Then all optimal restart quantiles p^* fulfill:*

$$(p^* - 1)Q(p^*) + p^*(1 - p^*)Q'(p^*) - \int_0^{p^*} Q(u)\,\mathrm{d}u = 0.$$

Proof. Recall that according to Lemma 4.3, the expected value $E[X_{Q(p)}]$ is given by:

$$E[X_{Q(p)}] = \frac{1-p}{p}Q(p) + \frac{1}{p}\int_0^p Q(u)\,\mathrm{d}u.$$

Now the derivative of $E[X_{Q(p)}]$ with respect to p is determined. We refer to this derivative as m':

$$m'(p) = \frac{\mathrm{d}E[X_{Q(p)}]}{\mathrm{d}p} = \frac{(p-1)}{p^2}Q(p) + \frac{(1-p)}{p}Q'(p) - \frac{1}{p^2}\int_0^p Q(u)\,\mathrm{d}u. \tag{4.7}$$

The optima can be found by setting $m'(p)$ equal to zero, thus multiplication by p^2 yields:

$$(p-1)Q(p) + p(1-p)Q'(p) - \int_0^p Q(u)\,\mathrm{d}u = 0,$$

This matches the desired property. □

4.3 ANALYSIS OF LOCATION-SCALE FAMILIES

Let X be a random variable and let $a \in \mathbb{R}_+$ and $b \in \mathbb{R}$ be any two constants. The random variables X and $Y = a \cdot X + b$ are members of the same location-scale family. As shall be seen below, the properties regarding restarts of X and Y are strongly related. This relationship is analyzed in the next two sub-sections by first considering the influence of the scale parameter a, and afterwards, the effect of the location parameter b.

4.3.1 *Scale Parameter*

This section shows that scale parameters (see Definition 2.39) do not affect both the usefulness of restarts and the optimal restart quantile.

Theorem 4.15. *Let X be a positive, real-valued random variable and let $a \in \mathbb{R}_+$ be an arbitrary, positive real number. Also, assume Y is a random variable given by $Y = a \cdot X$. Restarts are useful for Y if and only if restarts are useful for X.*

Proof. Let F_X be the cdf of X. According to Definition 2.39, the cdf F_Y of Y is then given by:

$$F_Y(x) = F_X\left(\frac{x}{a}\right).$$

In addition, assume Q_X is the quantile function of X, then $Q_Y(p) = a \cdot Q_X(p)$ is the quantile function of Y because the following holds true:

$$F_Y(Q_Y(p)) = F_X\left(\frac{a \cdot Q_X(p)}{a}\right) = F_X(Q_X(p)) = p.$$

Next, observe:

$$\int_0^p Q_Y(u)\,\mathrm{d}u = a \cdot \int_0^p Q_X(u)\,\mathrm{d}u.$$

As the expected value is linear (see Lemma 2.35), the following applies to $E[Y]$:

$$E[Y] = E[a \cdot X] = a \cdot E[X].$$

Let L_Y be the Lorenz curve of Y, and L'_Y the derivative of L_Y. Accordingly, assume L_X is the Lorenz curve of X, and L'_X is the derivative of L_X. According to Theorem 4.4, restarts are useful for Y precisely when

$$(1-p)L'_Y(p) + L_Y(p) < p. \tag{4.8}$$

We now apply the aforementioned observations about the relationship between X and Y, together with the definition of the Lorenz curve (Definition 4.2) to the left side of Inequality (4.8):

$$\begin{aligned}(1-p)L'_Y(p) + L_Y(p) &= (1-p)\frac{Q_Y(p)}{E[Y]} + \frac{1}{E[Y]}\int_0^p Q_Y(u)\,\mathrm{d}u \\ =(1-p)\frac{a\cdot Q_X(p)}{a\cdot E[X]} + \frac{a}{a\cdot E[X]}\int_0^p Q_X(u)\,\mathrm{d}u &= (1-p)L'_X(p) + L_X(p).\end{aligned} \tag{4.9}$$

Inserting Equation (4.9) into Inequality (4.8) yields the following result:

$$\begin{aligned}&(1-p)L'_Y(p) + L_Y(p) < p \\ \Leftrightarrow\ &(1-p)L'_X(p) + L_X(p) < p.\end{aligned}$$

Thus, as a result of Theorem 4.4, restarts are useful for Y if and only if restarts are useful for X. □

In fact, scale parameters also do not influence the optimal restart quantile, as is demonstrated in the following theorem.

Theorem 4.16. *Let X be a positive, real-valued random variable with quantile function Q_X and let $a \in \mathbb{R}_+$ be an arbitrary, positive real number. Furthermore, Y is assumed to be a random variable given by $Y = a \cdot X$ and having quantile function Q_Y. For $p \in (0,1)$, $Q_Y(p)$ is an optimal restart time for Y if and only if $Q_X(p)$ is an optimal restart time for X.*

Proof. As already argued in the proof for Theorem 4.15, Q_Y and Q_X have the following relationship:

$$Q_Y(p) = a \cdot Q_X(p). \tag{4.10}$$

For every $p \in (0,1)$, the expected value $E[Y_{Q_Y(p)}]$ can be expressed as follows by applying Lemma 4.3 and Equation (4.10):

$$\begin{aligned} E[Y_{Q_Y(p)}] &= \frac{1-p}{p} \cdot Q_Y(p) + \frac{\int_0^p Q_Y(u)\,\mathrm{d}u}{p} \\ &= a \cdot \frac{1-p}{p} \cdot Q_X(p) + a \cdot \frac{\int_0^p Q_X(u)\,\mathrm{d}u}{p} = a \cdot E[X_{Q_X(p)}] \end{aligned} \tag{4.11}$$

Assume that p^* is an optimal restart quantile for Y; this may be expressed equivalently as:

$$E[Y_{Q_Y(p^*)}] \leq E[Y_{Q_Y(p)}],\ \forall p \in (0,1). \tag{4.12}$$

By applying Equation (4.11) to Inequality (4.12) the following statement is obtained:

$$\begin{aligned} & E[Y_{Q_Y(p^*)}] && \leq E[Y_{Q_Y(p)}], && \forall p \in (0,1) \\ \Leftrightarrow\ & a \cdot E[X_{Q_X(p^*)}] && \leq a \cdot E[X_{Q_X(p)}], && \forall p \in (0,1) \\ \Leftrightarrow\ & E[X_{Q_X(p^*)}] && \leq E[X_{Q_X(p)}], && \forall p \in (0,1) \end{aligned}$$

The last inequality is equivalent to saying that p^* is an optimal restart quantile for X. As only equivalence transformations have been used, the proposition is shown. □

These findings show that scale parameters can be ignored in the analysis for restart times. For several commonly used distributions, the properties from Theorem 4.4 and Theorem 4.14 can be applied.

4.3.2 *Location Parameter*

Location parameters are another frequently used type of parameter. If X is any random variable, then a new random variable is given by $Y = X + b$, where $b \in \mathbb{R}$ is the location parameter (also see Definition 2.39). The parameter b shifts the distribution in a certain direction; if X has a (semi-)bounded distribution, the entire support is shifted.

This section shows that a location parameter only linearly influences the usefulness of restarts. Furthermore, a location parameter only exerts a moderate effect on the optimal restart quantile.

Corollary 4.17. *Let X be a positive, real-valued random variable together with its Lorenz curve L_X and the derivative L'_X. Also, assume Y is a random variable given by $Y = X + b$ for some positive, real number $b \in \mathbb{R}_+$. Then, restarts are useful for Y if and only if*

$$(1-p)L'_X(p) + L_X(p) < p + (p-1)c,$$

where $c = \frac{b}{E[X]}$.

Proof. Let F_X be the cdf of X and F_Y the cdf of Y. According to Definition 2.39 and Lemma 2.35, the cdf F_Y and the expected value $E[Y]$ can also be expressed as follows:

$$F_Y(x) = F_X(x-b),$$
$$E[Y] = E[X+b] = E[X] + b.$$

As a result, the quantile function Q_Y of Y can be represented by the quantile function Q_X of X. More precisely, $Q_Y(p) = Q_X(p) + b$, as can be easily verified:

$$F_Y\big(Q_Y(p)\big) = F_X\big(Q_X(p) + b - b\big) = p.$$

Inserting these observations into Theorem 4.4 yields the following:

$$\begin{aligned}
&(1-p)L_Y(p) + L_Y(p) < p \\
\Leftrightarrow &(1-p)\frac{Q_Y(p)}{E[Y]} + \frac{\int_0^p Q_Y(u)\,\mathrm{d}u}{E[Y]} < p \\
\Leftrightarrow &(1-p)\frac{Q_X(p)+b}{E[X]+b} + \frac{\int_0^p Q_X(u)+b\,\mathrm{d}u}{E[X]+b} < p.
\end{aligned}$$

The integral $\int_0^p Q_X(u) + b \, \mathrm{d}u$ can be decomposed into the two terms $\int_0^p Q_X(u) \, \mathrm{d}u$ and pb. The rest follows by simple arithmetic transformations:

$$\begin{aligned}
&(1-p)L_Y(p) + L_Y(p) < p \\
\Leftrightarrow &(1-p)\big(Q_X(p) + b\big) + pb + \int_0^p Q_X(u) \, \mathrm{d}u < p(E[X] + b) \\
\Leftrightarrow &(1-p)Q_X(p) + \int_0^p Q_X(u) \, \mathrm{d}u < p(E[X] + b) - b \\
\Leftrightarrow &(1-p)\frac{Q_X(p)}{E[X]} + \frac{\int_0^p Q_X(u) \, \mathrm{d}u}{E[X]} < p\left(1 + \frac{b}{E[X]}\right) - \frac{b}{E[X]} \\
\Leftrightarrow &(1-p)L'_X(p) + L_X(p) < p + (p-1)c.
\end{aligned}$$

In each step, equivalence transformations are used; thus, the desired property is shown. □

In the next corollary, the impact of location parameters on the optimal restart quantile is investigated.

Corollary 4.18. *Let X be a positive real-valued random variable having the quantile function Q_X and its derivative Q'_X. Also, assume that Y is a random variable given by $Y = X + b$ for some positive, real number $b \in \mathbb{R}_+$. All optimal restart quantiles p^* for Y satisfy:*

$$(p^* - 1)Q_X(p^*) + p^*(1-p^*)Q'_X(p^*) - \int_0^{p^*} Q_X(u) \, \mathrm{d}u = b.$$

Proof. Let Q_Y be the quantile function of Y and let Q'_Y be the derivative of Q_Y. In the proof of Corollary 4.17, it has already been argued that $Q_Y(p) = Q_X(p) + b$ holds. This can be used to specify the derivative Q'_Y:

$$Q'_Y(p) = \frac{\mathrm{d}}{\mathrm{d}p} Q_Y(p) = \frac{\mathrm{d}}{\mathrm{d}p}\big(Q_X(p) + b\big) = Q'_X(p).$$

Consider an optimal restart quantile p^* for Y. According to Theorem 4.14, p^* satisfies the condition

$$(p^* - 1)Q_Y(p^*) + p^*(1 - p^*)Q'_Y(p^*) - \int_0^{p^*} Q_Y(u)\,\mathrm{d}u = 0.$$

The characterization of Q_Y and Q'_Y can be inserted into the equation. The rest follows by using elementary arithmetic transformations:

$$\begin{aligned}
&(p^* - 1)Q_Y(p^*) + p^*(1 - p^*)Q'_Y(p^*) - \int_0^{p^*} Q_Y(u)\,\mathrm{d}u = 0 \\
\Leftrightarrow &(p^* - 1)\big(Q_X(p^*) + b\big) + p^*(1 - p^*)Q'_X(p^*) - \int_0^{p^*} \big(Q_X(u) + b\big)\,\mathrm{d}u = 0 \\
\Leftrightarrow &(p^* - 1)\big(Q_X(p^*) + b\big) + p^*(1 - p^*)Q'_X(p^*) - bp^* - \int_0^{p^*} Q_X(u)\,\mathrm{d}u = 0 \\
\Leftrightarrow &(p^* - 1)Q_X(p^*) + p^*(1 - p^*)Q'_X(p^*) - \int_0^{p^*} Q_X(u)\,\mathrm{d}u = b
\end{aligned}$$

This completes the proof. □

In the last two sections, scale and location parameters have been investigated. It was shown that scale parameters do not affect the usefulness of restarts as well as the optimal restart quantile, while the effect of location parameters is limited.

These results can be applied even if a scale and a location parameter are present at the same time. For example, assume that the random variable $Z = aX + b$ needs to be analyzed, where X is a random variable whose restart properties are known. In addition, $a \in \mathbb{R}_+$ has the role of a scale parameter and $b \in \mathbb{R}_+$ that of a location parameter. One can then first apply Theorem 4.15 or 4.16 to the auxiliary random variable $Y = aX$ and subsequently apply Corollary 4.17 or 4.18, respectively, to $Z = Y + b$.

With these results, it is reasonable to analyze distributions without location and scale parameters. This simplifies the analysis of whether a restart strategy should be applied and, if so, which restart quantile is optimal.

4.4 EXAMINATION OF DISTRIBUTIONS

In this section, we analyze the properties regarding restarts of typical distributions used to describe algorithms. In particular, the lognormal (Section 4.4.1), the Weibull (Section 4.4.2), and the generalized Pareto distribution (Section 4.4.3) are investigated.

4.4.1 *Lognormal*

Due to the central limit theorem, the lognormal distribution arises from the product of independent, identically distributed random variables.

In addition, the lognormal distribution is often used to describe the behavior of algorithms. Notably, the runtimes of local search algorithms have been reported in terms of the lognormal distribution. Arbelaez et al. [4] examined the runtime distributions of two local search SAT solvers (CCASAT [32] and SPARROW [12]) on randomly generated instances and noted that the lognormal distribution is a good fit. Furthermore, the local search CSP (constraint satisfaction problem, see [152]) solver ADAPTIVE SEARCH [42] also exhibits a lognormal distributed runtime behavior on magic-square-problem instances [167].

The runtimes of several evolutionary algorithms, including genetic programming, particle swarm optimization, and genetic algorithms, have been observed to be lognormal distributed by Barrero et al. [20]. Muñoz et al. [133] have empirically evaluated the runtime distributions of some path planning algorithms, such as A* [154] and THETA* [48], and have discovered that the runtime distribution follows lognormal distributions. According to Frost et al. [64], the runtime of several CSP backtracking algorithms corresponds to lognormal distributions if unsatisfiable binary CSP instances are used as input. Clearly, the lognormal distribution is used for a variety of different algorithmic problems. Accordingly, demonstrating the usefulness of restarts under lognormal distributions could benefit a variety of fields.

We refer to Definition 2.41 for the formal description of lognormal distributed random variables. In what follows, the quantile function of lognormal distributed random variables is of interest.

Lemma 4.19 ([95]). *Let X be a lognormal distributed random variable having parameters $\sigma \in \mathbb{R}_+$ and $\mu \in \mathbb{R}$. The quantile function Q of X and the expected value $E[X]$ are given by*

$$E[X] = e^{\mu+\sigma^2/2},$$
$$Q(p) = e^{\mu+\sigma\sqrt{2}\operatorname{erf}^{-1}(2p-1)},$$

where erf^{-1} *is the inverse of the error function* $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \cdot \int_0^x e^{-t^2} \, \mathrm{d}t$.

Although the inverse error function erf^{-1} cannot be solved analytically, numerical techniques are available. We are now prepared to analyze the usefulness of restarts for lognormal distributions.

Theorem 4.20. *Let X be a lognormal distributed random variable. Then there is a $p \in (0,1)$ such that*

$$E[X_{Q(p)}] < E[X].$$

Proof. Suppose f is the pdf and r is the hazard rate function of a lognormal distributed random variable. It is well-known that lognormal distributions have the long-tail property [61] and that the limit $\lim_{x\to\infty} r(x)$ exists (cf. [134]). The existence of the limit $\lim_{x\to\infty} x^2 \cdot f(x)$ is also easy to prove. The theorem is derived from this information by means of Theorem 4.12. For the purpose of self-containedness, the statement is examined in more detail.

Therefore, consider a lognormal distributed random variable X with arbitrary shape parameter $\sigma \in \mathbb{R}_+$ having pdf f, cdf F, and quantile function Q. According to Theorem 4.15, scale parameters do not affect the usefulness of restarts. Since e^{μ} is the scale parameter of the lognormal distribution, we may therefore assume $\mu = 0$.

In the first step, the existence of the limit $\lim_{x\to\infty} x^2 \cdot f(x)$ is shown:

$$\lim_{x\to\infty} x^2 \cdot f(x) = \lim_{x\to\infty} \frac{x^2 \cdot e^{-\frac{(\ln x-\mu)^2}{2\cdot\sigma^2}}}{x \cdot \sigma \cdot \sqrt{2\cdot\pi}} = \lim_{x\to\infty} \frac{x \cdot e^{-\frac{(\ln x)^2}{2\cdot\sigma^2}}}{\sigma \cdot \sqrt{2\cdot\pi}}.$$

This limit can be evaluated by applying the change of variable method with $x = e^t$:

$$\lim_{x\to\infty} x^2 \cdot f(x) = \lim_{t\to\infty} \frac{e^t \cdot e^{-\frac{\left(\ln e^t\right)^2}{2\cdot\sigma^2}}}{\sigma \cdot \sqrt{2\cdot\pi}} = \lim_{t\to\infty} \frac{\exp\left(t - \frac{t^2}{2\cdot\sigma^2}\right)}{\sigma \cdot \sqrt{2\cdot\pi}} = 0.$$

Next, the limiting value of the hazard rate function $r(x) = f(x)/\big(1 - F(x)\big)$ is considered. Once more, the change of variable method is applied with $x = Q(p)$:

$$\lim_{x\to\infty} \frac{f(x)}{1-F(x)} = \lim_{p\to 1} \frac{f\big(Q(p)\big)}{1-F\big(Q(p)\big)}.$$

We obtain an expression for $f\big(Q(p)\big)$ by applying Definition 2.41 and Lemma 4.19:

$$\begin{aligned} f\big(Q(p)\big) &= \frac{e^{-\frac{(\ln Q(p))^2}{2\cdot\sigma^2}}}{Q(p)\cdot\sigma\cdot\sqrt{2\cdot\pi}} = \frac{e^{-\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)}}{\sigma\cdot\sqrt{2\cdot\pi}} \cdot e^{-\frac{\left(\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)\right)^2}{2\cdot\sigma^2}} \\ &= \frac{e^{-\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)-\left(\mathrm{erf}^{-1}(2p-1)\right)^2}}{\sigma\cdot\sqrt{2\cdot\pi}}. \end{aligned}$$

Also, $F\big(Q(p)\big) = p$ holds as per definition. Thus, the limit of the hazard rate function can be written as follows:

$$\lim_{p\to 1} \frac{f\big(Q(p)\big)}{1-F\big(Q(p)\big)} = \lim_{p\to 1} \frac{e^{-\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)-\left(\mathrm{erf}^{-1}(2p-1)\right)^2}}{(1-p)\cdot\sigma\cdot\sqrt{2\cdot\pi}}. \tag{4.13}$$

An important fact about the inverse error function erf^{-1} is that $\mathrm{erf}^{-1}(u)$ tends towards infinity as u approaches 1 (implied in [163]). Thus, L'Hospital's rule can be utilized to further determine the limiting value from Equation (4.13). For this purpose, we first address the required derivatives. As erf^{-1} is inverse to the error function $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \cdot \int_0^x e^{-t^2}\,\mathrm{d}t$, the derivative of erf^{-1} can be determined by means of the well-known inverse function theorem [153]. In particular, we have:

$$\frac{\mathrm{d}}{\mathrm{d}p}\,\mathrm{erf}^{-1}(2p-1) = \sqrt{\pi}\cdot e^{\left(\mathrm{erf}^{-1}(2p-1)\right)^2}. \tag{4.14}$$

This observation can be used to determine the derivative of the terms in Equation (4.13) that are dependent on erf^{-1}:

$$\frac{\mathrm{d}}{\mathrm{d}p}\, e^{\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)} = \sqrt{2\cdot\pi}\cdot\sigma\cdot e^{\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)+\left(\mathrm{erf}^{-1}(2p-1)\right)^2}, \tag{4.15}$$

$$\frac{\mathrm{d}}{\mathrm{d}p}\, e^{\left(\mathrm{erf}^{-1}(2p-1)\right)^2} = 2\cdot\sqrt{\pi}\cdot e^{2\cdot\left(\mathrm{erf}^{-1}(2p-1)\right)^2}\cdot\mathrm{erf}^{-1}(2p-1). \tag{4.16}$$

Using Equation (4.15) and Equation (4.16), the derivative of $f\big(Q(p)\big)$ is calculated:

$$\frac{\mathrm{d}}{\mathrm{d}p} f\big(Q(p)\big) = -\frac{\sqrt{2}\cdot\sigma + 2\cdot \mathrm{erf}^{-1}(2p-1)}{\sqrt{2}\cdot\sigma\cdot e^{\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)}}. \tag{4.17}$$

Finally, L'Hospital's rule can be applied to Equation (4.13) with the derivative of the numerator being given by Equation (4.17):

$$\lim_{p\to 1}\frac{f\big(Q(p)\big)}{1-p} = \lim_{p\to 1}\frac{\sqrt{2}\cdot\sigma + 2\cdot \mathrm{erf}^{-1}(2p-1)}{\sqrt{2}\cdot\sigma\cdot e^{\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)}}.$$

As can be observed, L'Hospital's rule can be applied a second time. The required identities can be directly taken from Equation (4.14) and Equation (4.15). This yields

$$\begin{aligned}\lim_{p\to 1}\frac{f\big(Q(p)\big)}{1-p} &= \lim_{p\to 1}\frac{e^{\left(\mathrm{erf}^{-1}(2p-1)\right)^2}}{\sigma^2\cdot e^{\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)+\left(\mathrm{erf}^{-1}(2p-1)\right)^2}}\\ &= \lim_{p\to 1}\frac{1}{\sigma^2\cdot e^{\sigma\cdot\sqrt{2}\cdot\mathrm{erf}^{-1}(2p-1)}} = 0\end{aligned}$$

since $\mathrm{erf}^{-1}(2p-1)$ tends towards infinity as $p\to 1$. According to Lemma 4.11, X is long-tailed due to the fact that the hazard rate function is approaching 0 in the limit. Therefore, the conditions of Theorem 4.12 are fulfilled, and we can conclude that restarts are useful for X.

At the beginning of the proof, we set the parameter μ to zero. Having shown the usefulness of restarts for an arbitrary shape parameter $\sigma\in\mathbb{R}_+$. Based on Theorem 4.15, we conclude that restarts are useful for all lognormal distributions. □

As a consequence of this theorem, it can be concluded without further analysis that for lognormal distributed random variables, there are always restart times yielding a benefit in expectation. Nevertheless, it has not yet been established which restart times are useful and, in particular, optimal. This question is addressed in the following. For this purpose, consider a lognormal distributed random variable X

with pdf f and quantile function Q. The derivative Q' of the quantile function, as well as the antiderivative $\mathfrak{Q}$, can then be calculated

$$\mathfrak{Q}(p) = -\frac{1}{2} \cdot e^{\mu+\sigma^2/2} \cdot \operatorname{erf}\left(\frac{\sigma}{\sqrt{2}} - \operatorname{erf}^{-1}(2p-1)\right),$$
$$Q'(p) = \frac{1}{f(Q(p))} = \sigma\sqrt{2\pi} \cdot e^{\mu+\sqrt{2}\cdot\sigma\cdot\operatorname{erf}^{-1}(2p-1)+\left(\operatorname{erf}^{-1}(2p-1)\right)^2},$$

where erf is the error function. Using the three functions Q, Q' and $\mathfrak{Q}$, the conditions for the usefulness (Theorem 4.4) of restarts and the optimality of a restart quantile (Theorem 4.14) can be easily checked for specific $p \in (0,1)$. However, it should be emphasized that numerical approaches are necessary to compute both the error function erf as well as the inverse error function erf^{-1}.

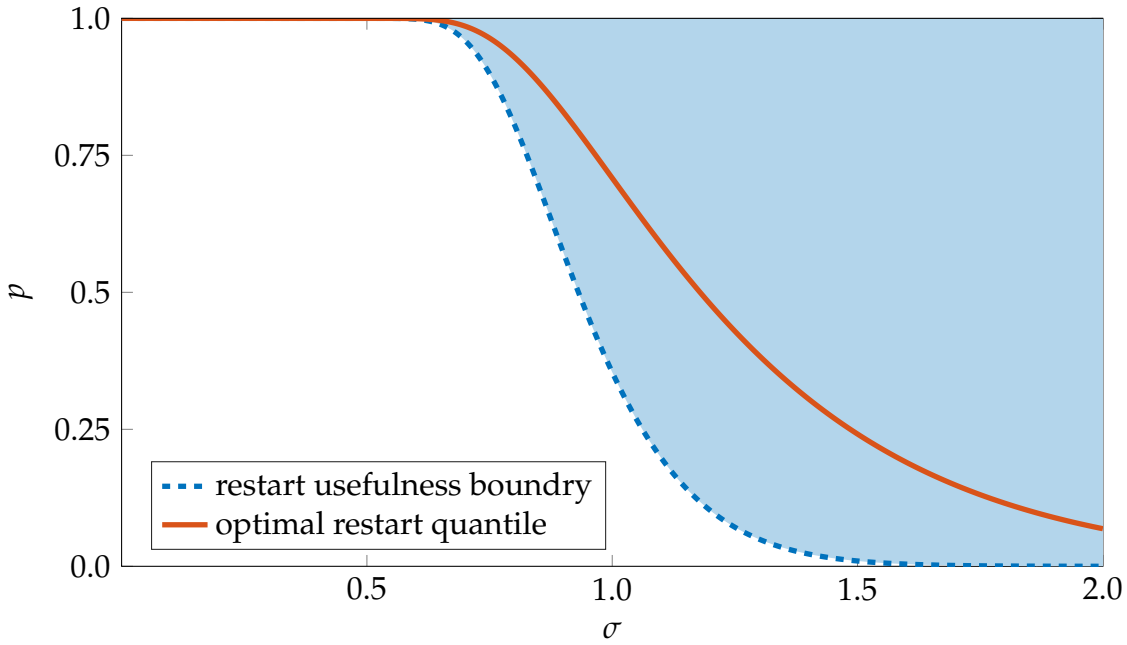


Figure 4.1: This plot visualizes the usefulness of restarts and the optimal restart quantiles for lognormal distributions. The scale parameter μ does not influence this plot's shape (cf. Theorem 4.15 and 4.16). The shape parameter σ can be found on the x-axis, and the quantiles $p \in (0,1)$ are represented on the y-axis. The light blue area corresponds to parameter combinations of σ and p for which restarts are useful in expectation, i. e., satisfying the condition of Theorem 4.4. Accordingly, the dashed blue line denotes parameter combinations for which restarts are neither useful nor harmful. The red line indicates the optimal restart quantiles for a given σ. That is, this quantile minimizes the expected value under restarts. Also, refer to Theorem 4.14. This figure is based on [110].

The area in which restarts are useful, and the optimal restart quantiles are shown in Figure 4.1. Here, Theorem 4.15 and Theorem 4.16 are implicitly used, indicating that the restart properties do not depend on the scale parameter μ. Accordingly, these characteristics can be evaluated purely on the basis of the shape parameter σ. The plot illustrates that for small σ values, only quantiles close to 1 provide an improvement to the expected value. Conversely, in case of high σ values, restarts are useful for almost all quantiles. The behavior of the optimal restart quantile is similar: For high σ values, the optimal restart quantile seems to approach $p \approx 0$.

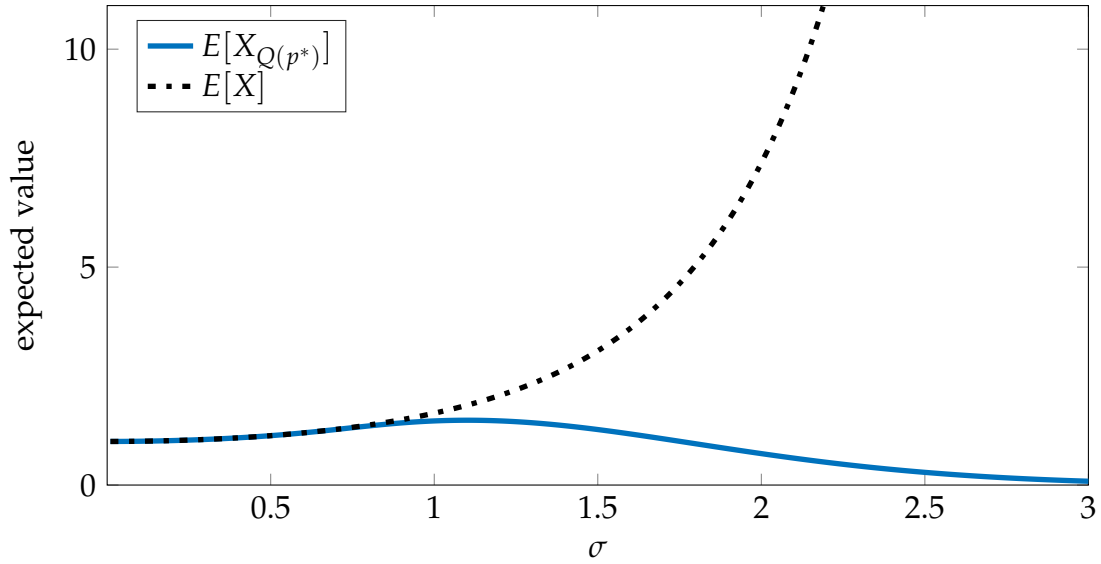


Figure 4.2: This plot compares the expected value without restarts with the expected value using the optimal restart strategy for lognormal distributions. The plot has the same appearance for all values of the scale parameter μ, only the values on the y-axis differ. Therefore, the scale parameter is fixed to $\mu = 0$. The shape parameter σ can be found on the x-axis, and the y-axis indicates the value of the respective expected values. The black dotted line represents the expected value without restarts, and the blue line corresponds to the expected value with restarts at the optimal restart time. This figure is based on [110].

Furthermore, the effect of restarts on the expected value still needs to be clarified. This issue is illustrated in Figure 4.2. It compares the expected value $E[X]$ of a lognormal distributed random variable X with the expected value $E[X_{Q(p^*)}]$ using the optimal restart strategy. The scale parameter μ is again fixed to $\mu = 0$, and therefore the values only depend on the shape parameter σ. For small σ values, the difference between $E[X]$ and $E[X_{Q(p^*)}]$ is negligible. Only starting at about $\sigma \approx 0.9$, a difference between the two values can be detected visually.

On the other hand, the expected value $E[X]$ tends superexponentially w. r. t. σ towards infinity, as can be verified by Lemma 4.19. In contrast, $E[X_{Q(p^*)}]$ starts to decline, beginning at a value of about $\sigma \approx 1.1$ and seemingly approaches 0 for large σ values. Indeed, this property can also be shown theoretically.

Theorem 4.21. *Let $X^{(\mu,\sigma)}$ denote a lognormal distributed random variable with scale $\mu \in \mathbb{R}$ and shape $\sigma \in \mathbb{R}_+$. Also, assume $t^*_{\mu,\sigma} \in \mathbb{R}_+$ to be the optimal restart time associated with $X^{(\mu,\sigma)}$. Then the following holds for the optimal expected value under restarts:*

$$\lim_{\sigma\to\infty} E\left[X^{(\mu,\sigma)}_{t^*_{\mu,\sigma}}\right] = 0.$$

Proof. Let $F_{\mu,\sigma}$ be the cdf associated with $X^{(\mu,\sigma)}$ (see Definition 2.41). We already used $t/F_{\mu,\sigma}(t)$ as an upper bound for $E[X^{(\mu,\sigma)}_t]$ in Chapter 3 when we were concerned with discrete random variables. As is well-known, this is also true if X is a continuous random variable. This can be verified using, for example, Theorem 2.47 by inserting the trivial upper bound $E[X^{(\mu,\sigma)} \mid X^{(\mu,\sigma)} \leq t] \leq t$.

In the subsequent, we use the restart time $t_{\mu,\sigma} = \exp(\mu - \sigma^2)$. Since $t^*_{\mu,\sigma}$ is by definition the optimal restart time, this results in the following estimate for the expected value $E[X^{(\mu,\sigma)}_{t^*_{\mu,\sigma}}]$:

$$\begin{aligned} 0 \leq E[X^{(\mu,\sigma)}_{t^*_{\mu,\sigma}}] \leq E[X^{(\mu,\sigma)}_{t_{\mu,\sigma}}] \leq \frac{t_{\mu,\sigma}}{F_{\mu,\sigma}(t_{\mu,\sigma})} &= \frac{2\cdot t_{\mu,\sigma}}{1+\operatorname{erf}\left(\frac{\ln(t_{\mu,\sigma})-\mu}{\sqrt{2}\cdot\sigma}\right)} \\ &= \frac{2\cdot\exp(\mu-\sigma^2)}{1+\operatorname{erf}\left(-\frac{\sigma}{\sqrt{2}}\right)}. \end{aligned} \tag{4.18}$$

We now analyze the limit of $t_{\mu,\sigma}/F_{\mu,\sigma}(t_{\mu,\sigma})$ as σ approaches infinity. Two necessary details should be clarified beforehand. First, $\lim_{x\to-\infty}\operatorname{erf}(x) = -1$ is valid. Second, the derivative of the error function is given by $\frac{\mathrm{d}}{\mathrm{d}x}\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\exp(-x^2)$. The desired limit can be established by using these facts and applying L'Hospital's rule twice:

$$\begin{aligned} \lim_{\sigma\to\infty}\frac{t_{\mu,\sigma}}{F_{\mu,\sigma}(t_{\mu,\sigma})} &= \lim_{\sigma\to\infty}\frac{2\exp(\mu-\sigma^2)}{1+\operatorname{erf}\left(-\frac{\sigma}{\sqrt{2}}\right)} = \lim_{\sigma\to\infty}\frac{-4\sigma\exp(\mu-\sigma^2)}{-\exp\left(-\frac{\sigma^2}{2}\right)}\sqrt{\frac{\pi}{2}} \\ &= \lim_{\sigma\to\infty}\frac{\sqrt{8\pi}\exp(\mu)\sigma}{\exp\left(\frac{\sigma^2}{2}\right)} = \lim_{\sigma\to\infty}\frac{\sqrt{8\pi}\exp(\mu)}{\sigma\exp\left(\frac{\sigma^2}{2}\right)} = 0. \end{aligned}$$

Since the upper bound $t_{\mu,\sigma}/F_{\mu,\sigma}(t_{\mu,\sigma})$ approaches 0 in the limit, Inequality (4.18) implies $\lim_{\sigma\to\infty} E[X^{(\mu,\sigma)}_{t^*_{\mu,\sigma}}] = 0$. □

The proof of this theorem further suggests that the optimal expected value under restarts is, in fact, approaching 0 superexponentially in σ.

4.4.2 *Weibull*

From a theoretical point of view, the Weibull distribution is of interest because it is one of three limiting distributions of the Fisher-Tippett-Gnedenko theorem (see for example [104]). Informally summarized, this theorem addresses the distribution of minima (respectively maxima) of independently, identically distributed random variables. If the distribution of minima (maxima) converges, then the Weibull distribution is one of the possible limiting distributions. Such a scenario presents itself if the same (randomized) algorithm is started in parallel on n processors and terminates as soon as a solution is found on any processor. The runtime then corresponds to the minimum of n independent[2], identically distributed random variables.

There are also some empirical studies that have investigated the runtime behavior of algorithms and discovered that Weibull distributions are a suitable fit. Frost et al. [64] found that the runtime of multiple CSP solvers on randomly generated, satisfiable instances can be reasonably well approximated by Weibull distributions. Hoos and Stützle [88] investigated the runtime of a SAT solver (GSAT WITH RANDOM WALK [70]) with varying parameter combinations. For non-optimal parameter settings, they discovered that the runtime distribution follows a Weibull distribution. In a paper by Barrero et al. [20], generation-based models without selection pressure are explored. They established that the number of generations to success can be expressed in terms of a Weibull distribution.

For a formal description of Weibull distributed random variables, we refer to Definition 2.42. An important result regarding restarts on Weibull distributed random quantities has already been reported by Wolter [180].

Theorem 4.22 ([180])**.** *Let X be a Weibull distributed random variable with shape parameter $k \in \mathbb{R}_+$. Restarts are useful for X if and only if $k < 1$.*

2 Here, it is assumed that the algorithms do not exchange information.

Therefore, in this section, we only consider the optimal restart quantile and the associated expected value. As we shall see, the restart times should be chosen as small as possible, causing the expected value to approach 0.

Theorem 4.23. *Consider a Weibull distributed random variable X with shape parameter $k \in (0,1)$ and let Q be the quantile function of X. Then, the following is valid for the expected value under restarts $E[X_{Q(p)}]$:*

$$\lim_{p \to 0} E[X_{Q(p)}] = 0.$$

Proof. Assume $a \in \mathbb{R}_+$ to be the scale parameter of the random variable X. As can be readily verified, the quantile function Q is given by

$$Q(p) = a \cdot \left(-\log\left(1-p\right)\right)^{\frac{1}{k}}.$$

According to Lemma 4.3, the expected value $E[X_{Q(p)}]$ is given by:

$$E[X_{Q(p)}] = \frac{1-p}{p} \cdot Q(p) + \frac{\int_0^p Q(u)\,\mathrm{d}u}{p}. \tag{4.19}$$

Both additive terms require attention for the limiting case $p \to 0$. First, we start with the right term. The integral is, of course, approaching 0 for $p \to 0$. Thus, L'Hospital's rule can be applied:

$$\lim_{p \to 0} \frac{\int_0^p Q(u)\,\mathrm{d}u}{p} = \lim_{p \to 0} Q(p) = 0. \tag{4.20}$$

A similar reasoning can be used for the left term in Equation (4.19) as well. As a preliminary consideration, we, therefore, calculate the derivative Q' of Q:

$$Q'(p) = a \cdot \frac{\left(-\log\left(1-p\right)\right)^{\frac{1}{k}-1}}{k \cdot (1-p)}.$$

L'Hospital's rule can thus be applied again, and we obtain:

$$\lim_{p \to 0} \frac{1-p}{p} \cdot Q(p) = \lim_{p \to 0} (1-p) \cdot \lim_{p \to 0} \frac{Q(p)}{p} = \lim_{p \to 0} Q'(p) = 0. \tag{4.21}$$

By combining Equation (4.20) and Equation (4.21), we obtain the desired result:

$$\lim_{p\to 0} E[X_{Q(p)}] = \lim_{p\to 0} \frac{1-p}{p} \cdot Q(p) + \lim_{p\to 0} \frac{\int_0^p Q(u)\,\mathrm{d}u}{p} = 0.$$

This completes the proof. □

This theorem's remarkable by-product is that the optimal expected value under restarts does not depend on any parameter in the limit. Not only that, but the expected value is even approaching zero; in terms of algorithms, this would imply that a result is found practically immediately, without any waiting time. For practical applications, it should therefore be checked thoroughly whether the Weibull distribution is indeed a suitable description, especially for extremely small values. Alternatively, an additional location parameter might be introduced to ensure that the resulting restart strategy is a bit more conservative.

In addition, the case $k > 1$ may also be analyzed. The proof is identical to the above, with the one difference being that $Q'(p)$ approaches infinity for $p \to 0$. Hence, the expected value approaches infinity. For this reason, we do not repeat the proof.

Lemma 4.24. *Consider a Weibull distributed random variable X with shape parameter $k \in (1, \infty)$ and let Q be the quantile function of X. Then, the following is valid for the expected value under restarts $E[X_{Q(p)}]$:*

$$\lim_{p\to 0} E[X_{Q(p)}] = \infty.$$

This statement will prove useful for a later analysis.

4.4.3 *Generalized Pareto*

The motivation for studying the generalized Pareto distribution can be attributed to the Pickands-Balkema-de Haan theorem [18, 143]. For a given random variable X, this theorem is concerned with the limiting distribution of the conditional excess distribution $\Pr(X - u \leq y \mid X > u)$ for $u \to \infty$. In many cases, this excess distribution converges to the generalized Pareto distribution. Therefore, the generalized Pareto distribution is considered a suitable candidate to model extreme events.

It is also a flexible distribution capable of modeling cases with bounded support as well as cases with (one-sided) unbounded support. Furthermore, many other

important distributions, such as the uniform, the exponential, and the Pareto distribution, are all included as special cases. The Pareto distribution, in particular, is widely used to model so-called power-laws. Informally, power-law distributions describe extreme events occurring with a non-negligible probability. Especially the runtimes of combinatorial search algorithms exhibit such a power-law behavior on many domains. Examples include the quasi-group completion problem [75] and graph coloring [93]. Outside of algorithmics, it has also been observed that network transmission times have power-laws [46].

The formal definition of the generalized Pareto distribution is provided in Definition 2.43. Several facts are beneficial in the following.

Lemma 4.25 ([95]). *Let X be a generalized Pareto distributed random variable having scale $a \in \mathbb{R}_+$ and shape $k \in \mathbb{R}$. The quantile function Q of X is given by:*

$$Q(p) = \frac{a}{k}\Big((1-p)^{-k} - 1\Big).$$

The expected value $E[X]$ is specified by:

$$E[X] = \begin{cases} \frac{a}{1-k}, & k < 1 \\ \infty, & k \geq 1. \end{cases}$$

For some of the special cases, the usefulness of restarts has already been analyzed in [180]. In particular, restarts are useful for Pareto distributions, have no influence on exponential distributions, and are harmful for uniform distributions. These findings can be extended using the generalized Pareto distribution.

The generalized Pareto distribution is a long-tail distribution if the shape parameter is chosen appropriately. Consequently, the usefulness of restarts could easily be analyzed using Theorem 4.12. However, this would only prove the existence of a quantile for which restarts are useful. We show a much stronger statement: Depending on the shape parameter, restarts are either useful for all quantiles or none.

Theorem 4.26. *Consider a generalized Pareto distributed random variable X having shape $k \in \mathbb{R}$. For all $p \in (0,1)$, the following statement is valid for the expected value under restarts $E[X_{Q(p)}]$:*

$$E[X_{Q(p)}] \begin{cases} < E[X], & \text{if and only if } k > 0, \\ = E[X], & \text{if and only if } k = 0, \\ > E[X], & \text{if and only if } k < 0. \end{cases}$$

Proof. The theorem's statement can be expressed equivalently by applying the reasoning of the proof of Theorem 4.4. Thus, the following is valid for all $p \in (0,1)$

$$\begin{aligned} &E[X_{Q(p)}] < E[X] \text{ if and only if } (1-p) \cdot L'(p) + L(p) < p, \\ &E[X_{Q(p)}] = E[X] \text{ if and only if } (1-p) \cdot L'(p) + L(p) = p, \\ &E[X_{Q(p)}] > E[X] \text{ if and only if } (1-p) \cdot L'(p) + L(p) > p, \end{aligned} \tag{4.22}$$

where L is the Lorenz curve of X and L' is the derivative of L with respect to p. At first, we examine the scenario in which $k \geq 1$ applies to the shape parameter. In this case, the expected value $E[X]$ is infinite in accordance with Lemma 4.25. This satisfies the conditions of Lemma 4.5 and therefore

$$(1-p) \cdot L'(p) + L(p) = 0 < p$$

holds for all $p \in (0,1)$ and thus fulfills the condition from Inequality (4.22). Hence, the proof for this case is complete.

Next, we consider the case in which the expected value $E[X]$ assumes a finite value, i. e., if $k < 1$. First, the antiderivative $\mathfrak{Q}$ of the quantile function Q is specified:

$$\mathfrak{Q}(p) = -\frac{a}{k} \cdot \left(p + \frac{(1-p)^{1-k}}{1-k} \right). \tag{4.23}$$

Using the antiderivative $\mathfrak{Q}$ and the expected value $E[X]$, the Lorenz curve L can be determined more precisely:

$$\begin{aligned} L(p) &= \frac{\mathfrak{Q}(p) - \mathfrak{Q}(0)}{E[X]} = \frac{-\frac{a}{k} \cdot \left(p + \frac{(1-p)^{1-k}}{1-k}\right) + \frac{a}{k} \cdot \frac{1}{1-k}}{\frac{a}{1-k}} \\ &= \frac{1}{k} \cdot \Big(1 - (1-p)^{1-k} - (1-k) \cdot p\Big). \end{aligned}$$

Furthermore, the derivative L' of the Lorenz curve L can be expressed as follows:

$$L'(p) = \frac{Q(p)}{E[X]} = \frac{\frac{a}{k} \cdot \left((1-p)^{-k} - 1\right)}{\frac{a}{1-k}} = \frac{1-k}{k} \cdot \Big((1-p)^{-k} - 1\Big).$$

The expression $(1-p) \cdot L'(p) + L(p)$ is crucial in this proof. For this reason, we simplify $(1-p) \cdot L'(p) + L(p)$ in the following:

$$\begin{aligned} & (1-p) \cdot L'(p) + L(p) \\ =& \frac{1-k}{k} \cdot \Big((1-p)^{1-k} - (1-p)\Big) + \frac{1}{k} \cdot \Big(1 - (1-p)^{1-k} - (1-k) \cdot p\Big) \\ =& \frac{1}{k} \cdot \Big((1-k) \cdot (1-p)^{1-k} - (1-k) + 1 - (1-p)^{1-k}\Big) \\ =& \frac{1}{k} \cdot \Big(-k \cdot (1-p)^{1-k} - (1-k) + 1\Big) \\ =& \frac{1}{k} \cdot \Big(-k \cdot (1-p)^{1-k} + k\Big) \\ =& -(1-p)^{1-k} + 1. \end{aligned}$$

We exemplarily show the case for $E[X_{Q(p)}] < E[X]$. The other two cases are entirely analogous. The only difference is that the inequality sign is either reversed or is an equals sign in the following:

$$\begin{aligned} & (1-p) \cdot L'(p) + L(p) < p \\ \Leftrightarrow\ & 1 - (1-p)^{1-k} < p \\ \Leftrightarrow\ & \log(1-p) < (1-k) \cdot \log(1-p) \\ \Leftrightarrow\ & k \cdot \log(1-p) < 0. \end{aligned} \tag{4.24}$$

As $\log(1-p)$ is a negative number for all $p \in (0,1)$, this implies that k must be positive to satisfy Inequality (4.24). As already mentioned above, the other two cases follow by analogous reasoning; therefore, the proof is complete. □

In addition, the question of the optimal restart time still needs to be answered. Similar to Weibull distributions, restarts should be carried out as early as possible, as outlined in the next theorem.

Theorem 4.27. *Consider a generalized Pareto distributed random variable X having scale $a \in \mathbb{R}_+$ and shape $k \in (0,\infty)$. Also, assume that Q is the quantile function of X. Then the following observations hold for the expected value under restarts:*

1. $\lim_{q\to 0} E[X_{Q(q)}] = a$,
2. $E[X_{Q(p)}]$ *is strictly monotonically increasing with respect to* p.

Proof. We start by showing proposition 1. The antiderivative $\mathfrak{Q}$ of Q is described in Equation (4.23). Some caution is required since $\mathfrak{Q}$ is discontinuous for $k = 1$. This can, however, be easily remedied. As can be verified, $\mathfrak{Q}$ is then given by:

$$\mathfrak{Q}(p) = \begin{cases} -\frac{a}{k} \cdot \left(p + \frac{(1-p)^{1-k}}{1-k}\right), & k \neq 1, \\ -a \cdot \left(p + \ln(1-p)\right), & k = 1. \end{cases} \tag{4.25}$$

According to Lemma 4.3, the expected value $E[X_{Q(p)}]$ can be calculated as follows:

$$E[X_{Q(p)}] = \frac{1-p}{p} \cdot Q(p) + \frac{\mathfrak{Q}(p) - \mathfrak{Q}(0)}{p}.$$

At first, we analyze the limit $\lim_{q\to 0} E[X_{Q(q)}]$ for the case $k \neq 1$. For this purpose, Equation (4.25) and Lemma 4.25 are applied:

$$\begin{aligned} \lim_{q\to 0} E[X_{Q(q)}] &= \lim_{q\to 0} \frac{1-q}{q} \cdot Q(q) + \frac{\mathfrak{Q}(q) - \mathfrak{Q}(0)}{q} \\ &= \lim_{q\to 0} \frac{a}{k} \cdot \frac{1-q}{q}\left((1-q)^{-k} - 1\right) - \frac{a}{k} \cdot \frac{\left(q + \frac{(1-q)^{1-k}}{1-k} - \frac{1}{1-k}\right)}{q}. \end{aligned} \tag{4.26}$$

In order to handle this limit, we first determine two auxiliary limits. First, we consider $\lim_{q\to 0} \frac{(1-q)^{-k}-1}{q}$ and apply L'Hospital's rule:

$$\lim_{q\to 0} \frac{(1-q)^{-k}-1}{q} = \lim_{q\to 0} k\cdot(1-q)^{-k-1} = k.$$

Next, $\lim_{q\to 0} \frac{q+\frac{(1-q)^{1-k}-1}{1-k}}{q}$ is examined, and once again L'Hospital's rule is used:

$$\lim_{q\to 0} \frac{q+\frac{(1-q)^{1-k}-1}{1-k}}{q} = \lim_{q\to 0} 1-(1-q)^{-k} = 0.$$

By suitable insertion of these two limits in Equation (4.26), the following result is obtained:

$$\lim_{q\to 0} E[X_{Q(q)}] = a.$$

The case $k = 1$ needs to be analyzed separately. As a preliminary consideration, we consider the limit $\ln(1-q)/q$ for $q \to 0$. It is obtained by applying L'Hospital's rule:

$$\lim_{q\to 0} \frac{\ln(1-q)}{q} = \lim_{q\to 0} -\frac{1}{1-q} = -1.$$

The limit $\lim_{q\to 0} E[X_{Q(q)}]$ can now be determined in case $k = 1$:

$$\begin{aligned}
&\lim_{q\to 0} E[X_{Q(q)}] = \lim_{q\to 0} \frac{1-q}{q}\cdot Q(q) + \frac{\mathfrak{Q}(q)-\mathfrak{Q}(0)}{q} \\
=&\lim_{q\to 0} a\cdot\frac{1-q}{q}\left((1-q)^{-1}-1\right) - a\cdot\frac{q+\ln(1-q)}{q} \\
=&\lim_{q\to 0} a\cdot\frac{q}{q} - a\cdot\frac{q}{q} - a\cdot\frac{\ln(1-q)}{q} = a.
\end{aligned}$$

This concludes the proof for the first statement, and we now address proposition 2. For this purpose, let m' be the derivative of $E[X_{Q(p)}]$ with respect to p. The desired

property is shown by proving that $m'(p)$ is positive for all $p \in (0,1)$. The general form of m' is already derived in Equation (4.7):

$$m'(p) = \frac{\mathrm{d}E[X_{Q(p)}]}{\mathrm{d}p} = \frac{(p-1)}{p^2}Q(p) + \frac{(1-p)}{p}Q'(p) - \frac{\mathfrak{Q}(p) - \mathfrak{Q}(0)}{p^2},$$

where Q' is the derivative of the quantile function Q and, in this case, Q' is given by:

$$Q'(p) = a \cdot (1-p)^{-1-k}.$$

As before, we will first examine the case $k \neq 1$:

$$\begin{aligned}
& m'(p) > 0 \\
\Leftrightarrow & \frac{p-1}{p^2} \cdot \frac{a}{k} \cdot \left((1-p)^{-k} - 1\right) + a \cdot \frac{(1-p)^{-k}}{p} + \frac{a}{k} \cdot \frac{p - \frac{1-(1-p)^{1-k}}{1-k}}{p^2} > 0 \\
\Leftrightarrow & \frac{1}{k} \cdot \left[(p-1) \cdot \left((1-p)^{-k} - 1\right) + p - \frac{1-(1-p)^{1-k}}{1-k}\right] + p \cdot (1-p)^{-k} > 0 \\
\Leftrightarrow & \frac{1}{k} \cdot \left[1 - (1-p)^{1-k} - \frac{1-(1-p)^{1-k}}{1-k}\right] + p \cdot (1-p)^{-k} > 0 \\
\Leftrightarrow & 1 - (1-p)^{1-k} - \frac{1-(1-p)^{1-k}}{1-k} + k \cdot p \cdot (1-p)^{-k} > 0.
\end{aligned}$$

At this point, a case distinction is conducted. First, we assume $k \in (0,1)$. Obviously, this implies that $1-k$ is positive. Thus,

$$\begin{aligned}
& m'(p) > 0 \\
\Leftrightarrow & (1-k) \cdot \left(1 - (1-p)^{1-k} + k \cdot p \cdot (1-p)^{-k}\right) - 1 + (1-p)^{1-k} > 0 \qquad (4.27) \\
\Leftrightarrow & (1-k) \cdot k \cdot p \cdot (1-p)^{-k} + k \cdot (1-p)^{1-k} - k > 0 \\
\Leftrightarrow & (1-k) \cdot p \cdot (1-p)^{-k} + (1-p)^{1-k} > 1 \\
\Leftrightarrow & (1-p)^{-k} \cdot \left((1-k) \cdot p \cdot + (1-p)\right) > 1 \\
\Leftrightarrow & (1-p)^{-k} \cdot \left(1 - k \cdot p\right) > 1. \qquad (4.28)
\end{aligned}$$

Observe the left-hand side of Inequality (4.28) for $p = 0$:

$$(1-p)^{-k} \cdot \left(1 - k \cdot p\right) = 1. \qquad (4.29)$$

Additionally, the derivative with respect to p of the left hand-side of Inequality (4.28) is considered:

$$\frac{\mathrm{d}}{\mathrm{d}p}\,(1-p)^{-k}\cdot\left(1-k\cdot p\right)=\frac{(k-1)\cdot k\cdot p}{(1-p)^{k}\cdot(p-1)}.$$

We now proceed by demonstrating that this derivative is positive for all $p\in(0,1)$:

$$\begin{aligned}&\frac{(k-1)\cdot k\cdot p}{(1-p)^{k}\cdot(p-1)}>0\\ \Leftrightarrow\;&(k-1)\cdot k\cdot p<0.\end{aligned}\tag{4.30}$$

Since by assumption $k\in(0,1)$, Inequality (4.30) is valid for all $p\in(0,1)$. This indicates that the left side of Inequality (4.28) is monotonically increasing. Together with Equation (4.29), this means that Inequality (4.28) is also valid for all $p\in(0,1)$. This, in turn, implies that $E[X_{Q(p)}]$ is monotonically increasing. For the case $k\in(1,\infty)$, the proof is similar. The inequality sign in Inequality (4.27), and all following inequalities reversed. Otherwise, the reasoning is identical. Thus, only the special case $k=1$ remains to be considered:

$$\begin{aligned}&m'(p)>0\\ \Leftrightarrow\;&(p-1)\cdot\left(\frac{1}{1-p}-1\right)+p\cdot(1-p)\cdot\frac{1}{(1-p)^{2}}+p+\ln(1-p)>0\\ \Leftrightarrow\;&\frac{p}{1-p}+\ln(1-p)>0.\end{aligned}\tag{4.31}$$

Similar to above, the left side of Inequality (4.31) is examined for $p=0$:

$$\frac{p}{1-p}+\ln(1-p)=0.\tag{4.32}$$

Next, the derivative of the left-hand side of Inequality (4.31) is calculated:

$$\frac{\mathrm{d}}{\mathrm{d}p}\,\frac{p}{1-p}+\ln(1-p)=\frac{p}{(1-p)^{2}}.$$

Now check for which p this derivative is positive:

$$\frac{p}{(1-p)^{2}}>0.$$

This inequality is obviously true for all $p \in (0,1)$. Thus, the left-hand-side of Inequality (4.31) is monotonically increasing, and together with Equation (4.32), it follows that $m'(p) > 0$ is true for all $p \in (0,1)$. This, in turn, implies that $E[X_{Q(p)}]$ is monotonically increasing. All cases have been analyzed, and it is thus proven that proposition 2 applies for all $k \in (0,\infty)$. □

It is remarkable that (in the limit) the optimal expected value under restarts $E[X_{Q(p^*)}] \to a$ does not depend on the shape parameter. By comparing this with the expected value without restarts $E[X] = a/(1-k)$ for $k \in (-\infty, 1)$, one immediately notices that for $k \to 0$ the two expected values $E[X_{Q(p^*)}]$ and $E[X]$ converge towards the same value. This is consistent with the fact that for $k = 0$, the generalized Pareto distribution becomes an exponential distribution. For exponential distributions, restarts have neither a positive nor a negative effect on the expected value [180].

4.5 CONCLUSION AND OUTLOOK

In this chapter, a necessary and sufficient condition for the usefulness of restarts is presented (Theorem 4.4), and furthermore, a necessary condition for the optimality of a restart quantile is developed (Theorem 4.14). For large classes of distributions, it is proven that restarts are useful for them. In particular, Theorem 4.12 shows that restarts are useful for most long-tail distributions. Using this theorem, other important distributions such as the Burr and the Fréchet distribution could also be analyzed.

Furthermore, the influence of scale and location parameters is analyzed. As Theorem 4.15 and 4.16 demonstrate, scale parameters do not influence the usefulness of restarts and also do not influence the optimal restart quantile. Corollary 4.17 and 4.18 illustrate that the impact of location parameters is limited. These results suggest that scale and location parameters can be neglected in the analysis of restart properties, which significantly simplifies such an analysis.

Furthermore, three important types of distributions are analyzed in terms of their restart properties. For the lognormal distribution, it has been established that restarts are useful and that the optimal expected value under restarts is approaching zero as the shape parameter increases.

However, one should note that although there are useful restart times for lognormal distributions, this does not mean that all restart times have a positive effect on the expected value. This is especially relevant if one can only rely on parameter esti-

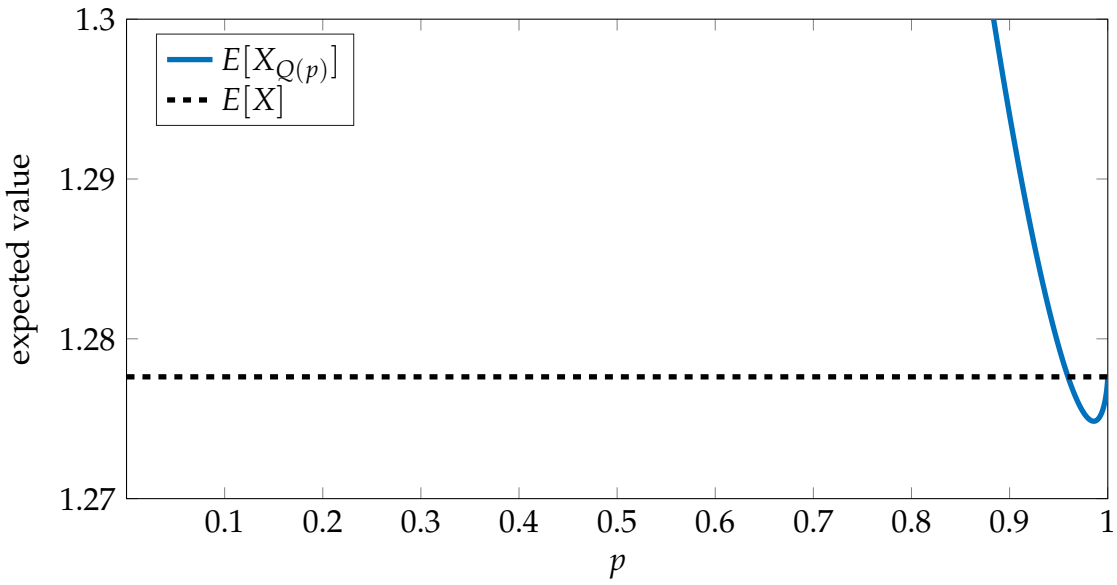


Figure 4.3: This figure depicts the expected runtime of a lognormally distributed random variable X with $\mu = 0$ and $\sigma = 0.7$ as a dashed line. The blue line is the expected runtime with restarts after $Q(p)$ steps. This figure is based on [110]

mates, which are inherently inaccurate. When additionally confronted with a small shape parameter σ, say $\sigma < 0.8$, there is a high risk of choosing an unsuitable restart quantile based on the parameter estimation. The effects of such a bad choice are illustrated in Figure 4.3. As can be observed there, $p \approx 0.985$ is a well-suited restart quantile. However, if the restart quantile is chosen too small, for example, $p < 0.96$, restarts can negatively impact the expected value.

Not every distribution requires this much care. An example is the generalized Pareto distribution, for which it is found that if restarts are useful, they are useful for all restart quantiles. In fact, choosing a suitable restart time is also easy: it should be as early as possible. A distribution with a similar property is the Weibull distribution, for which it turns out that when the restart quantiles approach zero, the expected value under restarts also approaches zero.

As mentioned above, restarts have been shown to be useful for almost all long-tail distributions. An open question is whether it is possible to extend this result to all heavy-tail distributions (see [61]).

5

RESTART STRATEGIES IN A CONTINUOUS SETTING

Parts of this chapter appeared in the following publication:

Jan-Hendrik Lorenz.
Restart Strategies in a Continuous Setting.
In: *Theory of Computing Systems* (2021), pp. 1–22.

Although the fixed-cutoff strategy is appealing in theory, it suffers from one substantial shortcoming: Determining the optimal restart time t^* is a non-trivial problem (refer to Chapter 3 in particular). One way to bypass this drawback is to employ a universal restart strategy (see Definition 2.44). Among the most prevalent strategies is the universal strategy by Luby et al. [123], which is both theoretically as well as practically attractive. From a theoretical point of view, a noteworthy property is that Luby's strategy is approximately optimal. This means that any other universal strategy can, in general, be better by at most a constant factor regarding the expected value. An empirical study by Huang [91] established that Luby's strategy performs best in practice. In other settings, e. g., (1+1) evolutionary algorithms, Luby's strategy also proves to be beneficial [62].

When attempting to minimize a continuous quantity, such as the examples discussed at the beginning of Chapter 4, it seems reasonable to consider well-established strategies from the field of algorithmics, such as Luby's strategy. This, nevertheless, poses the fundamental question of whether this is a viable strategy in a continuous environment, given that the performance guarantee has only been demonstrated for the discrete case.

This chapter is devoted to this particular issue. In general, a negative answer is obtained (Section 5.2). The performance guarantees of Luby's strategy do not hold in a continuous setting. Moreover, the result can be extended to the fact that there is no universal strategy with such guarantees. Finally, by restricting the problem scope, we obtain a class of distributions on which there are suitable universal strategies (Section 5.3).

5.1 FOUNDATIONS

In the following, $\alpha_X^* = \inf_{t<\infty} E[X_t]$ denotes the expected runtime using the optimal fixed-cutoff strategy. Generally, the runtime distribution depends on both the algorithm used and the problem instance. However, the runtime distribution is necessary to identify a good restart time, which is why the fixed-cutoff strategy is rarely used in practice. A frequently employed alternative is Luby's strategy. This strategy does not utilize any information about the distribution and is, therefore, a universal strategy. For the definition of universal strategies, refer to Definition 2.44.

Definition 5.1 ([123])**.** The universal restart strategy $L = (t_1, t_2, \dots)$ is called **Luby's strategy** if t_i is of the form

$$t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1, \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases},$$

for all $i \in \mathbb{N}$.

The start of the sequence is therefore $(1, 1, 2, 1, 1, 2, 4, 1, \dots)$. We denote the expected value of a random variable X using Luby's strategy with $E[X_{\text{Luby}}]$. Luby et al. [123] showed that the expected runtime $E[X_{\text{Luby}}]$ can be bounded w. r. t. the best fixed-cutoff strategy α_X^*.

Theorem 5.2 ([123])**.** *Let X be an arbitrary, discrete random variable defined on $\mathbb{N}$. Then the following inequality is valid:*

$$E[X_{\text{Luby}}] \leq 192\alpha_X^*(5 + \log_2 \alpha_X^*),$$

where $\alpha_X^* = \inf_{t<\infty} E[X_t]$.

In their article, Luby et al. [123] presented Luby's strategy for discrete runtime distributions. Moreover, they demonstrated that this strategy is asymptotically optimal. To be precise, they have shown that any universal strategy also involves (at least) the factor $\log \alpha_X^*$ in its associated expected value and thus cannot be significantly better than Luby's strategy.

An important property of restart strategies is that their expected value can be represented as a convex combination of expected values of fixed-cutoff strategies.

This characteristic has already been remarked in the preliminaries, however, we did not provide a proof. First, for the sake of completeness, we review the statement.

Lemma 2.49. *Let $\mathcal{L} = (t_1, t_2, \dots)$ be an arbitrary restart strategy and let X be any continuous or discrete positive random variable. Let $E[X_{t_i}]$ denote the expected value of X with the fixed-cutoff strategy $(t_i, t_i, \dots)$. Then, $E[X_{\mathcal{L}}]$ can be expressed by*

$$E[X_{\mathcal{L}}] = \sum_{k=1}^{\infty} a_k E[X_{t_k}],$$

where $(a_1, a_2, \dots)$ is a sequence with $a_k \in \mathbb{R}_+, \forall k \in \mathbb{N}$ and $\sum_{k=1}^{\infty} a_k = 1$.

Proof. Luby et al. [123] showed this property for discrete distributions. We prove the continuous case. Let $\mathcal{A}$ be some randomized algorithm with associated (continuous) random variable X having pdf f_X and cdf F_X. An arbitrary restart strategy $\mathcal{L} = (t_1, t_2, \dots)$ induces a restarted version of $\mathcal{A}$, which is denoted by $\mathcal{A}_{\mathcal{L}}$. We identify the runtime distribution of $\mathcal{A}_{\mathcal{L}}$ with a random variable $Y := X_{\mathcal{L}}$ having pdf f_Y and cdf F_Y. Define the cumulative restart times $T_i := \sum_{k=1}^{i} t_k$ and set $T_0 := 0$ by convention. The function $S_Y(x)$ is shorthand for $\Pr(Y > x) = 1 - F_Y(x)$ in this proof.

The function S_Y can be written in terms of F_X. Observe the probability $S_Y(T_i + x)$ with $x < t_{i+1}$ and $i \in \mathbb{N}$. The runtime of $\mathcal{A}_{\mathcal{L}}$ is greater than $T_i + x$ if the first i runs all failed, and $\mathcal{A}_{\mathcal{L}}$ also does not find a solution within the first x time-units of the $(i+1)$-st run. All runs are, by definition, independent of each other. Therefore, the probability $S_Y(T_i + x)$ is given by

$$S_Y(T_i + x) = \big(1 - F_X(x)\big) \prod_{k=1}^{i} \big(1 - F_X(t_k)\big). \tag{5.1}$$

The pdf f_Y of Y can readily be calculated by taking the appropriate derivative:

$$f_Y(T_i + x) = f_X(x) \prod_{k=1}^{i} \big(1 - F_X(t_k)\big) = f_X(x) S_Y(T_i). \tag{5.2}$$

We first prove the following auxiliary identity:

$$\sum_{i=0}^{\infty} T_i S_Y(T_i) F_X(t_{i+1}) = \sum_{i=1}^{\infty} t_i S_Y(T_i). \tag{5.3}$$

The first term of the left sum is zero and can, therefore, be omitted. Furthermore, note that $S_Y(T_i)F_X(t_{i+1})$ corresponds to the probability $\Pr(T_i < Y \leq T_{i+1})$. Then, the desired result is achieved with the following transformations:

$$\begin{aligned}
&\sum_{i=0}^{\infty} T_i S_Y(T_i) F_X(t_{i+1}) = \sum_{i=1}^{\infty} T_i S_Y(T_i) F_X(t_{i+1}) \\
&= \sum_{i=1}^{\infty} \sum_{k=1}^{i} t_k S_Y(T_i) F_X(t_{i+1}) = \sum_{k=1}^{\infty} t_k \sum_{i=k}^{\infty} S_Y(T_i) F_X(t_{i+1}) \\
&= \sum_{k=1}^{\infty} t_k \sum_{i=k}^{\infty} \Pr(T_i < Y \leq T_{i+1}) = \sum_{k=1}^{\infty} t_k S_Y(T_k).
\end{aligned}$$

The expected value $E[Y]$ is given by:

$$\begin{aligned}
E[Y] &= \int_0^{\infty} x f_Y(x)\,\mathrm{d}x = \sum_{i=0}^{\infty} \int_{T_i}^{T_{i+1}} x f_Y(x)\,\mathrm{d}x \\
&= \sum_{i=0}^{\infty} \int_0^{t_{i+1}} (T_i + x) f_Y(T_i + x)\,\mathrm{d}x \\
&= \sum_{i=0}^{\infty} T_i S_Y(T_i) \int_0^{t_{i+1}} f_X(x)\,\mathrm{d}x + S_Y(T_i) \int_0^{t_{i+1}} x f_X(x)\,\mathrm{d}x \qquad (5.4) \\
&= \sum_{i=0}^{\infty} T_i S_Y(T_i) F_X(t_{i+1}) + S_Y(T_i) \int_0^{t_{i+1}} x f_X(x)\,\mathrm{d}x. \qquad (5.5)
\end{aligned}$$

To obtain Equation (5.4), the characterization of the pdf f_Y from Equation (5.2) is applied. The left part of Equation (5.5) can be transformed with Equation (5.3). For the right part, we use Theorem 2.47 and Definition 2.38 to obtain the identity

$$\int_0^{t_i} x f_X(x)\,\mathrm{d}x = F_X(t_i) E[X_{t_i}] - t_i\big(1 - F_X(t_i)\big),$$

where $E[X_{t_i}]$ is the expected runtime of $\mathcal{A}$ using the fixed-cutoff strategy $(t_i, t_i, \dots)$. The appropriate transformations yield:

$$\begin{aligned} E[Y] &= \sum_{i=1}^{\infty} t_i S_Y(T_i) + S_Y(T_{i-1})\Big(F_X(t_i)E[X_{t_i}] - t_i(1 - F_X(t_i))\Big) \\ &= \sum_{i=1}^{\infty} t_i S_Y(T_i) + S_Y(T_{i-1})F_X(t_i)E[X_{t_i}] - t_i S_Y(T_i) \\ &= \sum_{i=1}^{\infty} S_Y(T_{i-1})F_X(t_i)E[X_{t_i}] = \sum_{i=1}^{\infty} \Pr\left(T_{i-1} < Y \leq T_i\right)E[X_{t_i}]. \end{aligned}$$

In other words, the expected runtime $E[Y]$ can be expressed as a combination of expected runtimes of fixed-cutoff strategies. Obviously, $\Pr\left(T_{i-1} < Y \leq T_i\right)$ is non-negative and $\sum_{i=1}^{\infty} \Pr\left(T_{i-1} < Y \leq T_i\right)$ sums up to one. This completes the proof. □

We frequently use Weibull distributed random variables for the reasoning in the proofs. A particularly crucial characteristic of Weibull distributions in relation to restarts is that the expected value under restarts is monotone.

Lemma 5.3. *Let X be a Weibull distributed random variable with shape $k \in \mathbb{R}_+$ and arbitrary scale $\lambda > 0$. If $k < 1$, then the expected value $E[X_t]$ with restarts at t increases strictly monotonically for $t > 0$. If $k > 1$, then $E[X_t]$ is strictly monotonically decreasing.*

Proof. The properties of the Weibull distribution with respect to restarts have already been discussed intensively in Chapter 4. Inserting the definition of the Weibull distribution (Definition 2.42) along with the definition of the truncated expectation (Definition 2.38) into the expected value under restarts $E[X_t]$ (Theorem 2.47) yields:

$$E[X_t] = \frac{1}{1 - \exp\left[-(\lambda t)^k\right]} \left(t \exp\left[-(\lambda t)^k\right] + \frac{1}{\lambda}\gamma\left[1 + \frac{1}{k}, (\lambda t)^k\right] \right).$$

Here, $\gamma[s, x] = \int_0^x t^{s-1}e^{-t}\,\mathrm{d}t$ is the lower incomplete gamma function. In the following, we determine the derivative of $E[X_t]$ with respect to t. In order to calculate this, the derivative of the lower incomplete gamma function $\gamma\left[1 + \frac{1}{k}, (\lambda t)^k\right]$ is discussed first. By applying the definition of γ and the chain rule, we obtain:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\gamma\left[1 + \frac{1}{k}, (\lambda t)^k\right] &= k \cdot \lambda \cdot \exp\left[-(\lambda t)^k\right] \cdot \left((\lambda t)^k\right)^{\frac{1}{k}} \cdot (\lambda t)^{k-1} \\ &= k \cdot \lambda \cdot \exp\left[-(\lambda t)^k\right] \cdot (\lambda t)^k. \end{aligned}$$

We are now ready to determine the derivative of $E[X_t]$ with this preliminary consideration. For the remainder of the proof, we write $g(t)$ as an abbreviation for $\frac{\mathrm{d}}{\mathrm{d}t}E[X_t]$. The derivative $g(t)$ is then given by:

$$g(t) = \frac{\exp\left[-(\lambda t)^k\right]}{1-\exp\left[-(\lambda t)^k\right]}\left(1 - \frac{\exp\left[-(\lambda t)^k\right]k(\lambda t)^k + k(\lambda t)^{k-1}\gamma\left[1+\frac{1}{k},\,(\lambda t)^k\right]}{1-\exp\left(-(\lambda t)^k\right)}\right).$$

We will show that $g(t)$ is positive (resp. negative) for all $t > 0$ if $k < 1$ (resp. $k > 1$). In case $k < 1$ the inequality $g(t) > 0$ is considered, which may be equivalently expressed as follows:

$$\frac{1-\exp\left[-(\lambda t)^k\right]}{k} - (\lambda t)^k \exp\left[-(\lambda t)^k\right] - (\lambda t)^{k-1}\gamma\left[1+\frac{1}{k},\,(\lambda t)^k\right] > 0. \tag{5.6}$$

The case $k > 1$ is similar, whereas the >-sign becomes a <-sign in the inequality. We now continue with the case $k < 1$; we briefly discuss the case $k > 1$ at the end of the proof.

First, the limit of Inequality (5.6) for the case $t \to 0$ is examined more closely. Of particular interest is the limit of the term $(\lambda t)^{k-1}\gamma\left[1+\frac{1}{k},\,(\lambda t)^k\right]$. Consequently, the behavior of this term is analyzed in more detail. The limiting value can be derived by applying L'Hospital's rule:

$$\lim_{t\to 0} -\frac{\gamma\left[1+\frac{1}{k},\,(\lambda t)^k\right]}{(\lambda t)^{1-k}} = \lim_{t\to 0} -\frac{\exp\left[-(\lambda t)^k\right]k(\lambda t)^{2k}}{1-k} = 0.$$

The other terms in Inequality (5.6) obviously also approach 0 in the limit, and accordingly, the entire left side of Inequality (5.6) tends towards 0 in the limit. Inequality (5.6) can also be expressed equivalently as follows:

$$\frac{1-\exp\left[-(\lambda t)^k\right]}{k(\lambda t)^{k-1}} - \exp\left[-(\lambda t)^k\right]\lambda t - \gamma\left[1+\frac{1}{k},\,(\lambda t)^k\right] > 0. \tag{5.7}$$

In the following, $h(t)$ represents the derivative of the left side of Inequality (5.7) $\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1-\exp\left[-(\lambda t)^k\right]}{k(\lambda t)^{k-1}} - \exp\left[-(\lambda t)^k\right]\lambda t - \gamma\left[1+\frac{1}{k},\,(\lambda t)^k\right]\right)$. One can readily verify that $h(t)$ is given as follows:

$$h(t) = \frac{\left(1-\exp\left[-(\lambda t)^k\right]\right)(1-k)\lambda}{k(\lambda t)^k}.$$

As can easily be checked, $h(t)$ then assumes the following values for $t > 0$:

$$h(t)\begin{cases} > 0, \text{ for } k < 1, \\ = 0, \text{ for } k = 1, \\ < 0, \text{ for } k > 1. \end{cases}$$

In summary, if $k < 1$, then $h(t)$ is positive and this in turn implies $g(t) > 0$ for all $t > 0$. Hence, in this case, $E[X_t]$ is strictly monotonically increasing. On the other hand, for the case $k > 1$, the left side of Inequality (5.6) approaches 0 in the limit for $t \to 0$. Since $h(t)$ is strictly negative, it follows that $E[X_t]$ has to be strictly monotonically decreasing. □

5.2 NONEXISTENCE RESULTS

Theorem 5.2 provides a powerful performance guarantee for Luby's strategy. This section examines whether this or any other guarantee also applies to continuous scenarios. We obtain a negative result and thereby show that no such bound exists and, moreover, that there is no universal strategy having such a bound.

Theorem 5.4. *There is no universal strategy $L = (t_1, t_2, \dots)$ such that there is a function $f : \mathbb{R}_+ \to \mathbb{R}$ with $\lim_{x\to 0} f(x) < \infty$ and $\exists x \in \mathbb{R}_+ : f(x) < \infty$ such that*

$$E[X_L] \leq \alpha_X^* \cdot f(\alpha_X^*) \tag{5.8}$$

for all runtime distributions X, where $\alpha_X^* = \inf_{t<\infty} E[X_t]$.

Proof. Consider an arbitrary, universal restart strategy $L = (t_1, t_2, \dots)$. Let, furthermore, f be a function in accordance with the assumption of the theorem. We

distinguish between two scenarios: Either all restart times $t_i \in L$ are exactly 0 or at least one restart time $t_j \in L$ is not equal to 0.

We start with the case where there is a non-zero restart time. For this purpose, let $j \in \mathbb{N}$ be a natural number with $t_j > 0$. We also define the cumulative restart times as $T_i := \sum_{k=1}^{i} t_k$ and set $T_0 := 0$ by convention. Furthermore, consider a Weibull distributed random variable X with shape parameter $k < 1$ and arbitrary scale parameter $\lambda \in \mathbb{R}_+$. According to Theorem 4.23, the optimal expected runtime approaches zero, i. e. $\alpha_X^* = 0$. Since, by assumption, $\lim_{x \to 0} f(x) < \infty$ holds, the right side of Inequality (5.8) must assume the value 0, i. e., $E[X_L] \leq 0$.

As per Lemma 5.3, the expected value $E[X_t]$ increases strictly monotonically in t. Since the optimal restart time is approaching zero and t_j is greater than zero, $E[X_{t_j}] > \alpha_X^* = 0$ follows. Following Lemma 2.49, the expected value $E[X_L]$ using the restart strategy L can be expressed as a convex combination $\sum_{k=1}^{\infty} a_k E[X_{t_k}]$. In particular, we can use this result to bound $E[X_L]$ by $E[X_L] \geq a_j E[X_{t_j}]$. Furthermore, by following the proof of Lemma 2.49, we also obtain $a_j = \Pr\left(T_{j-1} < X_L \leq T_j\right)$. As the cdf of the Weibull distribution is strictly monotonically increasing, $a_j > 0$ follows. Since both a_j and $E[X_{t_j}]$ are strictly greater than zero, the following observation holds:

$$E[X_L] \geq a_j E[X_{t_j}] > \alpha_X^* \cdot f(\alpha_X^*) = 0.$$

In other words, Inequality (5.8) does not hold.

In the second case, all restart times are assumed to be constantly zero, so $t = 0$ holds $\forall t \in L$. However, it is straightforward to find a random variable Y causing the expected value $E[Y_L]$ to diverge towards infinity. For instance, consider a Weibull distribution with shape parameter $k = 2$. As already shown in Lemma 4.24, the expected value $E[Y_L]$ approaches infinity. According to Wolter [180], restarts are not beneficial for Y, i. e., $\alpha_Y^* = \inf_{t<\infty} E[Y_t]$ assumes the expected value $E[Y]$. The expected value of a Weibull distribution is given by $\lambda \cdot \Gamma(1 + \frac{1}{k})$, where $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} \, \mathrm{d}t$ is the gamma function [95]. In our case, we, therefore, have $\alpha_Y^* = \lambda \cdot \Gamma(1.5)$. By assumption, there is at least one $x \in \mathbb{R}_+$ with $f(x) < \infty$. We choose λ such that $\lambda \cdot \Gamma(1.5) = x$ holds. Then, $\alpha_Y^* \cdot f(\alpha_Y^*)$ is finite, and consequently, the following is valid:

$$E[Y_L] > \alpha_Y^* \cdot f(\alpha_Y^*)$$

Therefore, Inequality (5.8) does not hold in this case either. □

This result is immediately applicable to Luby's strategy because the logarithm approaches minus infinity as its argument approaches zero. Hence, the conditions of Theorem 5.4 are met, and this implies that the limit in Theorem 5.2, shown by Luby et al. [123], is not valid in a continuous context. Naturally, this statement can be extended for all bounds in accordance with Theorem 5.4. As Luby's strategy is of major importance in practice, we state this finding as a corollary.

Corollary 5.5. *There is no function* $f : \mathbb{R}_+ \to \mathbb{R}$ *with the following two properties* (i) $\exists x \in \mathbb{R}_+ : f(x) < \infty$ *and* (ii) $\lim_{x \to 0} f(x) < \infty$ *such that*

$$E[X_{\text{Luby}}] \leq \alpha_X^* \cdot f(\alpha_X^*)$$

for all random variables X *defined on* $\mathbb{R}_+$*, where* $\alpha_X^* = \inf_{t<\infty} E[X_t]$.

Proof. This result is implied by Theorem 5.4. □

It should also be noted that computers are inherently discrete systems operating with finite precision. Nevertheless, in some circumstances, the results described above are also relevant for computers. The restart strategies representable by a computer as, for example, floating-point numbers are obviously a proper subset of real-valued restart strategies. Since Theorem 5.4 provides a statement about all universal restart strategies, the realizable universal restart strategies are also included. Therefore, the results described above are relevant, for instance, when a computer monitors a continuous quantity, such as power consumption or network transmission times. Under such circumstances, there is no restart strategy realizable by a computer with a performance guarantee as per Theorem 5.4.

5.3 EXISTENCE RESULTS

The previous section demonstrated that, in general, there is no universal strategy with a performance guarantee for real-valued random variables. For the proof, Weibull distributions are used for which, with appropriate parameter choices, both the optimal restart time and the expected value under restarts approaches 0. However, many distributions do not have such behavior, and therefore we investigate which classes of distributions admit universal strategies with performance guarantees. We base our arguments on the reasoning presented in [103] and adapt it to

the continuous case. We primarily improve the analysis of the expected value in Theorem 5.7 and thereby obtain, to the best of our knowledge, the currently best known upper bound for the expected runtime when using Luby's strategy. In the following, β_X^* denotes $\min_{t\in\mathbb{R}_+} \frac{t}{F(t)}$, and $k^* \in \mathbb{R}_+$ is a real number with $\frac{k^*}{F(k^*)} = \beta_X^*$.

Lemma 5.6 ([103, 123]). *The inequality*

$$\beta_X^* \leq 4\alpha_X^*$$

holds for all positive random variables X, where $\alpha_X^* = \inf_{t<\infty} E[X_t]$.

Proof. Luby et al. [123] report this property for discrete probability distributions. The proof, however, is omitted in their work. A slight modification of the proof discussed in [103] also shows this property for continuous distributions.

Let F be the cdf of X and let $t \in \mathbb{R}_+$ be an optimal restart time, i. e., $E[X_t] = \alpha_X^*$. In the following, we distinguish between two cases and begin with the assumption $E[X_t] < \frac{t}{2F(t)}$. Then $\Pr\big(X \leq 2E[X_t]F(t)\big) \geq 1/2$ holds for all random variables X with this property. To demonstrate this, an auxiliary random variable $Y := \min(X, t)$ is defined. The expected values $E[Y]$ and $E[X_t]$ can be observed to be closely related:

$$E[Y] = F(t) \cdot E[X \mid X \leq t] + \big(1 - F(t)\big) \cdot t = F(t) \cdot E[X_t].$$

The last transformation follows due to Theorem 2.47. In the next step, the probability $\Pr\big(X > 2E[X_t]F(t)\big)$ is examined more closely. Since $2E[X_t]F(t) < t$ holds,

$$\Pr\big(X > 2E[X_t]F(t)\big) = \Pr\big(Y > 2E[X_t]F(t)\big) = \Pr\big(Y > 2E[Y]\big) \leq \frac{1}{2} \tag{5.9}$$

follows by means of Markov's inequality (see, e. g., [151]). Also, Inequality (5.9) indicates $\Pr\big(X \leq 2E[X_t]F(t)\big) \geq 1/2$. By using this property, β_X^* can be bounded:

$$\beta_X^* = \min_x \frac{x}{F(x)} \leq \frac{2E[X_t]F(t)}{\Pr\big(X \leq 2E[X_t]F(t)\big)} \leq 4E[X_t]F(t) \leq 4E[X_t] = 4\alpha_X^*.$$

We now move on to the second case $E[X_t] \geq \frac{t}{2F(t)}$. This property can directly be utilized to estimate α_X^*:

$$\alpha_X^* = E[X_t] \geq \frac{t}{2F(t)} \geq \frac{1}{2} \min_x \frac{x}{F(x)} = \frac{\beta_X^*}{2}.$$

So one can conclude that $\beta_X^* \leq 4\alpha_X^*$ is valid for all positive distributions. □

With the help of this lemma, it can be shown that for probability distributions for which the optimal restart time does not approach 0, there are universal strategies having a useful performance guarantee.

Theorem 5.7. *Let X be a positive real-valued random variable having cdf F such that there is a k^* with $\frac{k^*}{F(k^*)} = \beta_X^*$ and $k^* > 2^{-d-1}$ for some $d \in \mathbb{N}_0$. Then,*

$$E[X_{\text{Luby}}] \leq 2^{d+7}\alpha_X^*(d + 4 + \log_2 \alpha_X^*)$$

holds, where $\alpha_X^* = \inf_{t<\infty} E[X_t]$.

Proof. For the proof, we apply the techniques described in [103]. A contribution made here is improving the analysis of the expected value, which results in the current best estimate of Luby's strategy's expected runtime to the best of our knowledge.

For the remainder of the proof, Y is used as an abbreviation for the random variable X_{Luby}, which results from applying Luby's strategy to X. The shortest run, such that its length is at least k^*, is often considered for the reasoning. The length of all runs in Luby's strategy are natural numbers and powers of two. Since, by assumption, k^* is greater than 2^{-d-1}, it follows that $2^{\lceil \log_2 k^* \rceil + d}$ meets the desired property. We, therefore, use the two auxiliary values $j_0 := \lceil \log_2 k^* \rceil + d$ and $n_0 := \lceil -\log_2 F(k^*) \rceil$ in the following.

First, the case $n_0 = 0$ or, equivalently, $F(k^*) = 1$ is considered. Furthermore, consider any two natural numbers $m \in \mathbb{N}$ and $j \in \mathbb{N}_0$ with $j < m$. If $Y > m2^{m-1}$, then there are at least 2^{m-1-j} runs of length 2^j, as can be readily verified by using the definition of Luby's strategy. Following this argument, for $Y > (j_0 + 1)2^{j_0}$, at least one run of length 2^{j_0} is performed. Since $F(k^*) = 1$ holds, $\Pr(Y > (j_0 + 1)2^{j_0}) = 0$ follows. Thus, the expected value $E[Y]$ can be estimated:

$$\begin{aligned} E[Y] &\leq (\lceil \log_2 k^* \rceil + d + 1)2^{\lceil \log_2 k^* \rceil + d} \leq k^*(\log_2 k^* + d + 2)2^{d+1} \\ &= \beta_X^*(\log_2 \beta_X^* + d + 2)2^{d+1} \leq \alpha^*(\log_2 \alpha_X^* + d + 4)2^{d+3}. \end{aligned} \tag{5.10}$$

In Equation (5.10), first $\beta_X^* = k^*/F(k^*)$ and $F(k^*) = 1$ is used, then Lemma 5.6 is applied. The last inequality satisfies the statement of the theorem.

We now proceed to the case $F(k^*) < 1$, then $n_0 \geq 1$ is valid. Furthermore, $j_0 \geq 0$ holds due to the definition of j_0. As already argued, if $Y > m2^{m-1}$, there

are at least 2^{m-1-j} runs of length 2^j. This allows an estimation of the probability $\Pr(Y > m2^{m-1})$:

$$\Pr(Y > m2^{m-1}) \leq \Pr(X > 2^j)^{2^{m-1-j}}. \tag{5.11}$$

Inequality (5.11) can be used to estimate the tail probability of Y. Thus, for $\ell \in \mathbb{N}_0$, the following holds true:

$$\begin{aligned}\Pr\left(Y > (j_0 + n_0 + \ell + 1)2^{j_0+n_0+\ell}\right) &\leq \Pr(X > 2^{j_0})^{2^{n_0+\ell}} \leq \Pr(X > 2^{j_0})^{\frac{2^\ell}{F(k^*)}} \\ &\leq \left(1 - F(k^*)\right)^{\frac{2^\ell}{F(k^*)}} \leq \exp\left(-2^\ell\right). \end{aligned} \tag{5.12}$$

By appropriately utilizing Inequality (5.12), the expected value $E[Y]$ can be bounded from above. For this purpose, we exploit a well-known characterization (see, for example, [128]) of the expected value:

$$E[Y] = \int_0^\infty \left(1 - F_Y(x)\right) \mathrm{d}x.$$

Let $d(\ell) := (j_0 + n_0 + \ell + 1)2^{j_0+n_0+\ell}$ for $\ell \geq 0$ and $d(-1) := 0$. We then obtain the following bound for $E[Y]$ by applying Inequality (5.12):

$$\begin{aligned} E[Y] &= \sum_{\ell=0}^\infty \int_{d(\ell-1)}^{d(\ell)} \left(1 - F_Y(x)\right) \mathrm{d}x \\ &\leq (j_0 + n_0 + 1)2^{j_0+n_0} + \sum_{\ell=0}^\infty (j_0 + n_0 + \ell + 3)2^{j_0+n_0+\ell} \exp\left(-2^\ell\right) \\ &\leq 2(j_0 + n_0)2^{j_0+n_0} + \sum_{\ell=0}^\infty (j_0 + n_0)(\ell + 4)2^{j_0+n_0+\ell} \exp\left(-2^\ell\right) \qquad (5.13) \\ &\leq (j_0 + n_0)2^{j_0+n_0} \left(2 + \sum_{\ell=0}^\infty (\ell + 4)\left(\frac{2}{\exp(2)}\right)^\ell\right). \qquad (5.14) \end{aligned}$$

In (5.13), the inequality $j_0 + n_0 \geq 1$ is implicitly used. The series $\sum_{\ell=0}^{\infty} (\ell+4) \left(\frac{2}{\exp(2)}\right)^{\ell}$ is analyzed in the next step:

$$\begin{aligned}
&\sum_{\ell=0}^{\infty} (\ell+4) \left(\frac{2}{\exp(2)}\right)^{\ell} = \sum_{\ell=0}^{\infty} (\ell+1) \left(\frac{2}{\exp(2)}\right)^{\ell} + 3 \sum_{\ell=0}^{\infty} \left(\frac{2}{\exp(2)}\right)^{\ell} \\
&= \sum_{\ell=1}^{\infty} \ell \left(\frac{2}{\exp(2)}\right)^{\ell-1} + 3 \sum_{\ell=0}^{\infty} \left(\frac{2}{\exp(2)}\right)^{\ell} \\
&= \left(\frac{e^2}{e^2-2}\right)^2 + 3\frac{e^2}{e^2-2} \leq 6. \qquad (5.15)
\end{aligned}$$

Hence, inserting Inequality (5.15) in Inequality (5.14) simplifies the estimate:

$$E[Y] \leq 8(j_0 + n_0) 2^{j_0+n_0}. \qquad (5.16)$$

The definitions of j_0 and n_0 yield:

$$2^{j_0+n_0} \leq 2^{2+d+\log_2 k^* - \log_2 F(k^*)} = 2^{d+2} \frac{k^*}{F(k^*)} = 2^{d+2} \beta_X^*. \qquad (5.17)$$

Finally, Inequality (5.17) has to be inserted into Inequality (5.16) and Lemma 5.6 is applied:

$$E[Y] \leq 2^{d+5} \beta_X^* (d + 2 + \log_2 \beta_X^*) \leq 2^{d+7} \alpha_X^* (d + 4 + \log_2 \alpha_X^*).$$

This estimate conforms to the statement of the theorem, and thus the proof is complete. □

It is worth noting that for $d = 0$, the upper bound of the expected runtime is $128\alpha_X^*(4 + \log_2 \alpha_X^*)$, which is a considerable improvement over the previously known bound for Luby's strategy[1]. The arguments used in the reasoning of Theorem 5.7 can be transferred back to the discrete case, thus implicitly improving the bound in that case as well. Furthermore, Theorem 7 in [123] limits any universal strategy's performance from below by a bound of the form $\alpha_X^* \log(\alpha_X^*)$. While the proof is given for discrete probability distributions, the arguments can directly be applied to the case discussed here. This implies that Luby's strategy is asymptotically optimal for the class of distributions investigated here. Moreover, Theorem 7 in [123] also suggests that only the constants in the upper bound can be further improved.

1 The previously sharpest bound was: $180\alpha_X^*(4 + \log_2 \alpha_X^*)$ (see [103]).

5.4 CONCLUSION AND OUTLOOK

Several well-established algorithms (e.g. [62, 75]) employ restart strategies to optimize their expected runtimes. A ubiquitous strategy among these algorithms is Luby's universal strategy. It is a slowly exponentially growing sequence of restart times. Luby et al. [123] demonstrated their universal strategy to be asymptotically optimal in the worst case within their work.

Previously, these properties have only been shown for discrete distributions. This work examines Luby's strategy and general universal strategies in a continuous setting. Theorem 5.4 states that, in general, there are no universal strategies with reasonable worst-case bounds in a continuous environment. In other words, any universal strategy (including Luby's strategy) is arbitrarily worse than an optimal strategy for some distributions.

More precisely, for a universal restart strategy L, there is no estimate of the form $E[X_L] \leq \alpha_X^* \cdot g(\alpha_X^*)$ with $\lim_{t\to 0} g(t) < \infty$. The only remaining possibility is providing a runtime guarantee for universal strategies using a function g with $\lim_{t\to 0} g(t) = \infty$. The expected runtime $E[X_x]$ is known to converge to $1/f(x)$ when x approaches zero, where f is the density function. Among the critical distributions is the Weibull distribution, which has the property $f(x) \sim x^{k-1}$ for $x \to 0$ and $k \in (0, 1)$. One can therefore infer that every possible bound function g has to approach infinity at least as fast as $1/x$ for $x \to 0$.

Nevertheless, universal strategies find wide acceptance among practitioners because they eliminate the need for knowledge about the underlying distribution. Often some details about the distribution are available, however. For this reason, we investigate conditions admitting worst-case bounds for Luby's strategy. Theorem 5.7 states that there are meaningful worst-case bounds if the optimal restart time is not too small. In addition, we improved the original bounds by Luby et al. [123].

In summary, two extreme cases can be distinguished: Firstly, if the underlying distribution is fully known, an optimal restart time can be computed, resulting in an optimal strategy. The other extreme case is when the distribution is entirely unknown, in which case only universal strategies are available. For future research, it would be of interest whether partial knowledge of the distribution can be utilized to create an efficient intermediate restart strategy admitting better bounds than Luby's strategy.

6

THE POTENTIAL OF RESTARTS FOR PROBSAT

Parts of this chapter appeared in the following publications:

Jan-Hendrik Lorenz and **Julian Nickerl**.
The Potential of Restarts for ProbSAT.
In: *International Conference on Computer Aided Systems Theory*. Springer, 2019, pp. 352–360.

Jan-Hendrik Lorenz and **Julian Nickerl**.
The Potential of Restarts for ProbSAT.
In: *arXiv preprint arXiv:1904.11757* (2019).

In Chapter 4, a technique for calculating the optimal restart quantile (and thus the optimal restart time) for continuous random distributions has been introduced. For these techniques to be applied in practice, the probability distribution has to be known in advance. This is, however, usually not the case in the field of algorithms. Although the probability distribution can be empirically approximated by repeatedly solving the instance at hand, in practical applications, most instances are no longer of interest once they have been solved once. Over the last few years, there have been exciting new results (for instance, [6, 51]) deploying machine learning approaches to predict the runtime distribution of previously unseen instances.

The purpose of this chapter is to analyze if and which probability distributions are adequate to describe the runtime behavior of the local search SAT solver PROBSAT [17]. Another question is whether machine learning techniques can be utilized to predict the runtime distribution and derive a useful restart strategy. One of the main reasons for considering PROBSAT is that it has been a highly successful solver in recent years; its core concepts are used in numerous other solvers. This is further elaborated upon in Section 6.4.

Two domains are considered. First, uniformly, randomly generated 3-SAT instances having a clause-to-variable ratio close to the satisfiability threshold (≈ 4.267, according to [126]) are examined. Secondly, randomly generated instances with a

hidden solution are studied. The majority of the latter type of instances are difficult to solve for local search solvers such as PROBSAT.

This chapter demonstrates that a machine learning pipeline can be constructed on both domains. This pipeline predicts runtime distributions accurately enough to infer a restart strategy from the estimated distributions, which statistically significantly improves the performance. For the uniform instances, the average speedup factor is 1.06, and for particularly difficult instances, the speedup factor is 1.21. While a factor of 1.21 is already a considerable result, the speedup on instances with hidden solutions even reaches an average of 50.95, clearly demonstrating the viability and relevance of this approach.

A comparable work that should be mentioned is from Haim and Walsh [80]. In their work, they present a portfolio SAT solver utilizing machine learning techniques to choose between different restart strategies. However, only a fixed number of preconfigured restart strategies are selectable; furthermore, each of these restart strategies is independent of the given instance. In contrast, our approach involves computing a restart strategy from the predicted parameters; hence, we use restart strategies dependent on the given instance. Therefore, the approaches presented in [80] and in this chapter differ substantially.

6.1 OVERVIEW

Figure 6.1 illustrates a machine learning pipeline at an abstract level. The first step involves choosing a suitable random distribution to describe the runtime distribution for a given instance. During the second step, the parameters for the selected distribution type are estimated.

The underlying idea is to observe the behavior of PROBSAT on multiple instances in order to draw conclusions about which distribution types are suitable to describe the runtime behavior. In addition, the a-priori unknown parameters have to be determined for these distribution types on the basis of the observations. Section 6.2 presents the theoretical methods used for estimating these parameters, as well as evaluating whether the distribution type is adequate.

For an unseen instance, we decide which distribution type and which parameters are best suited based on the previous observations. For determining the distribution type, we employ random forests, and for estimating the parameters, we use neural networks as well as random forests. These methods are introduced in Section 6.3.

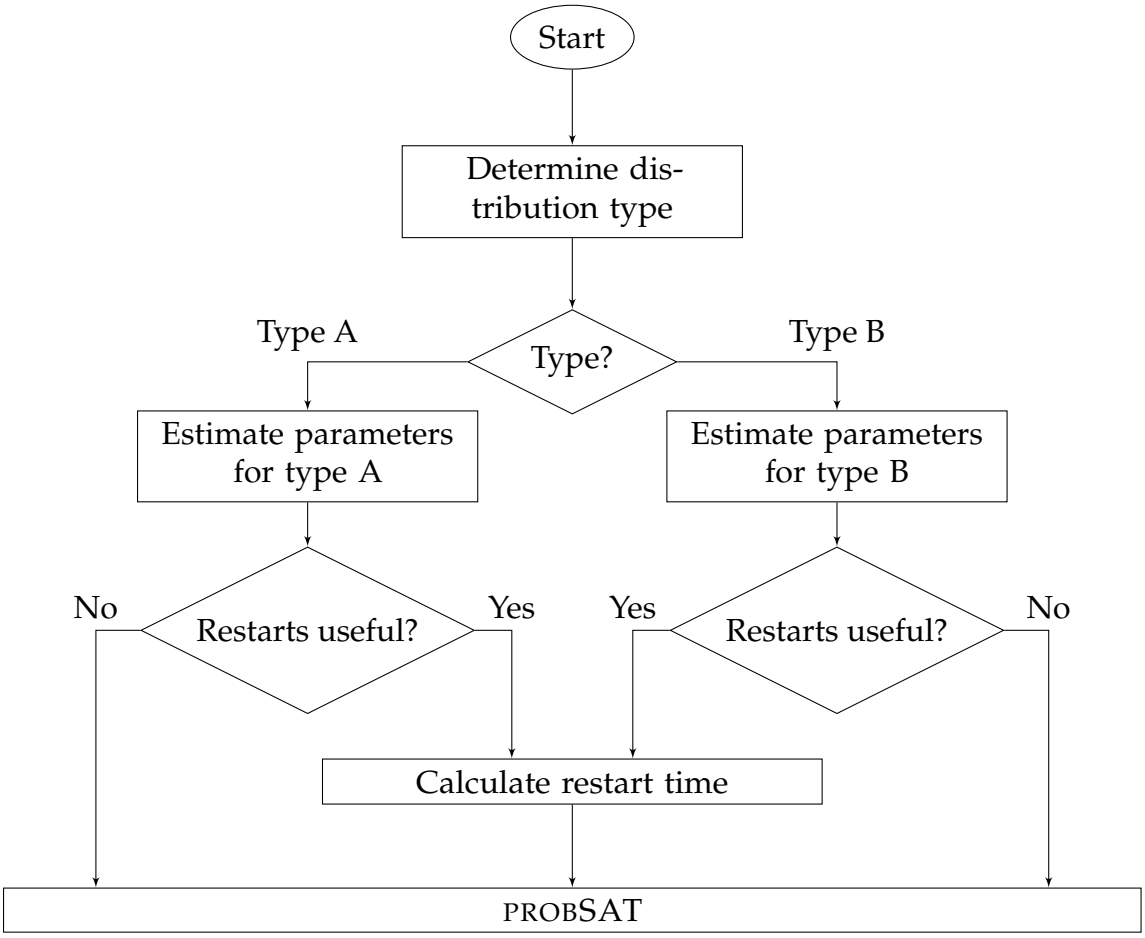


Figure 6.1: An example of a machine learning pipeline on an abstract level.

After the first two steps in the pipeline, the distribution type, and the associated parameters have been established. This information can then be utilized to assess whether restarts should be performed and if so, an optimal restart quantile can be calculated. The methodology used for these steps has already been addressed in Chapter 4. In particular, we refer to Section 4.1 for deciding whether restarts are useful and to Section 4.2 for computing the optimal restart quantile.

In the last step, PROBSAT is invoked, with or without restarts, depending on the previous choices. In Section 6.4, PROBSAT and its related concepts are illustrated. As mentioned before, two different domains are considered, namely uniform instances as well as instances with a hidden solution. In Section 6.5, these two domains are formally introduced.

We then consider the behavior of PROBSAT on uniform instances in Section 6.6, followed by an analysis of PROBSAT on instances with a hidden solution in Section 6.7. In both cases, we examine which distributions are suitable for describing the runtime behavior and develop a machine learning model to predict the runtime distribution of previously unseen instances. Both approaches are also evaluated experimentally.

Finally, Section 6.7.5 verifies, on the SAT Competition 2018 [84] instances, whether the presented techniques are suitable to make PROBSAT competitive with the current state-of-the-art solvers.

6.2 ESTIMATION AND EVALUATION OF RUNTIME DISTRIBUTIONS

In this chapter, we often have some independent, identically distributed observations of a random variable X. Suppose it is known that X belongs to a certain distribution family having parameter space Θ; however, the specific parameters $\theta \in \Theta$ are unknown. The parameters can be approximated by using the maximum likelihood approach.

Definition 6.1 ([177])**.** Suppose that $X_1, X_2, \ldots, X_n$ are independent, identically distributed random variables having pdf $f^{(\theta)}$, which depends on the (unknown) parameters θ. Also, assume that $x_1, \ldots, x_n$ are realizations of the random variables $X_1, \ldots, X_n$. The **likelihood function** $\mathcal{L}_n$ is given by:

$$\mathcal{L}_n(\theta) = \prod_{i=1}^{n} f^{(\theta)}(x_i).$$

An estimate of the parameters can be obtained by maximizing the likelihood function $\mathcal{L}_n$.

Definition 6.2 ([177])**.** Let $\mathcal{L}_n$ be a likelihood function. The **maximum likelihood estimator** $\hat{\theta}$ is the value of θ that maximizes $\mathcal{L}_n(\theta)$.

The maximum likelihood estimator has several appealing properties; details on these characteristics are, however, beyond the scope of this work. We refer to [177] for further reading. In another scenario, the underlying cdf of a sample is unknown. In this case, the empirical cdf can often be used.

Definition 6.3 ([72])**.** Consider n samples $(x_1, \ldots, x_n)$ that have been sampled from the independent, identically distributed random variables $X_1, \ldots, X_n$. We use $x_{(i)}$ to denote the i-th smallest sample. That is:

$$x_{(1)} < x_{(2)} < \cdots < x_{(n)}.$$

The **empirical cumulative distribution function (ecdf)** $\hat{F}_n$ is given by:

$$\hat{F}_n(x) = \begin{cases} 0, & x < x_{(1)} \\ \frac{i}{n}, & x_{(i)} \leq x < x_{(i+1)}, \text{ for } i \in \{1, \ldots, n-1\} \\ 1, & x \geq x_{(n)}. \end{cases}$$

In some sections, we use the KS test or the KS test statistic, defined in the following. For a comprehensive discussion of the KS test and its properties, we refer to [72].

Definition 6.4 ([72])**.** Let $\hat{F}_n$ be an ecdf for some $n \in \mathbb{N}$ and let X be a continuous random variable with cdf F_X. The **Kolmogorov-Smirnov test statistic** D_n (also called **KS test statistic**) is given by:

$$D_n = \sup_{x \in \mathbb{R}} |\hat{F}_n(x) - F_X(x)|.$$

Typically, KS test statistics are used for refuting that two probability distributions are identical. Assume there are n independent and identically distributed samples of a random variable X with unknown cdf F_X available. The null hypothesis H_0 states

that the unknown cdf F_X corresponds to a given cdf F_0. Put differently, the following applies:

$$H_0 : F_X(x) = F_0(x), \forall x \in \mathbb{R}.$$

The KS test statistic D_n is then calculated. In the next step, working under the assumption that the null hypothesis H_0 is true, the probability of observing a KS test statistic at least as extreme as D_n is calculated. This probability is called the p-value. If p is small enough, then the null hypothesis is rejected. One may, therefore, assume that the two cdfs F_X and F_0 are not identical. Typical p-values beyond which the null hypothesis is rejected are 0.05 and 0.01. This threshold is called the level of significance and is denoted by α. We do not go into detail about the calculation of the p-value here. However, it is worth mentioning that most common statistical software packages are capable of calculating the p-value. The whole procedure is called KS test.

For the most part, we use the KS test for a different purpose. Usually, the aim is to refute that the unknown cdf F_X matches the given cdf F_0, whereas we are interested in whether F_0 is a reasonably accurate description of F_X. We use the KS test statistic as a measure of how well a distribution describes the observed data. We also calculate the p-value; if it is above the level of significance $\alpha = 0.05$, then we consider F_0 to be an adequate fit. We wish to stress that this should by no means be regarded as proof of $F_X = F_0$. The purpose is solely to measure the quality of the fit.

6.3 USED MACHINE LEARNING TECHNIQUES

This section first introduces the fundamental setting. In the remainder of this section, two specific methods are presented: random forests and neural networks.

The introduction to the problem is based on [92]. Suppose there are p input variables $X = (X_1, \ldots, X_p)$ (**features**) and another variable of interest Y (**label**). The goal is to use the features X as a means to make statements about the behavior of Y or, in some contexts, to predict the value of Y. Generally, the aim is to define a function f so that Y is described as accurately as possible by $f(X)$. There are two types of problems: If Y is a qualitative quantity, it is a classification problem, whereas if Y is a quantitative quantity, then a regression problem arises.

For example, suppose that a CNF formula is given. From this instance, one can easily infer some features, such as the number of variables and the number of clauses. An associated classification problem is whether the given instance is satisfiable.

Practically speaking, previous observations are used to determine the function f. To indicate concrete observations, lower case letters are used for the observed values in the following. Thus, $x_i = (x_{i,1}, \ldots, x_{i,p})$ denotes the i-th observation of the features and y_i represents the i-th observation of the label variable. The observations can be split into a training and a test set. The **training set** is used to determine the function f, while the **test set** is utilized to evaluate the quality of the obtained function f. The quality is evaluated by the so-called **loss function** L. For example, in regression problems, the mean squared error is often used as loss function:

$$L(f) = \frac{1}{N} \sum_{i=1}^{N} \big(y_i - f(x_i)\big)^2,$$

where N is the number of observations in the test set. In general, other loss functions may also be used. We explicitly specify the loss function if it does not correspond to the mean squared error. The loss function can also be used to minimize the error in determining the function f. In this case, the training set is used for the loss function instead of the test set. Up to now, only a rather abstract description of how the function f is constructed has been given. In the next two sections, two specific methods are presented, which are used in this chapter.

6.3.1 *Random Forests*

The principle of random forests is introduced in [31]. In order to describe random forests, it is best to first illustrate the concept of a decision tree. A decision tree is a directed tree. Each internal node is labeled with a rule concerning a feature X_i. Accordingly, the edges are labeled with the possible results of this rule. The leaf nodes are then assigned a realization of the label variable y. Broadly speaking, decision trees are constructed in a greedy recursive manner. The rule that improves the loss function the most is chosen for each node. Appendix A discusses this approach in more detail.

The following paragraph is based on [81]. Random forests are a collection of decision trees. Suppose that the training set consists of N observations composed of p features. Each decision tree is obtained by first sampling N observations (with

replacement) from the training set. For each split, that is, each time a rule is chosen for an inner node, the number of available features is limited to $q \leq p$. The q available features are also determined randomly. The construction of the decision tree is then conducted as described in Appendix A. If a random forest is designed for a classification problem, typically $q = \lfloor \sqrt{p} \rfloor$ is chosen. For regression problems, [81] suggests the choice of $q = \lfloor \frac{p}{3} \rfloor$, but in [71] it is empirically shown that $q = p$ is often a better option.

Finally, the output of a random forest has to be defined. In the case of a classification problem, the output class is determined by a majority vote of the decision trees. For a regression problem, the output is the arithmetic mean of the output of the trees.

6.3.2 *Neural Networks*

We also use what is known as feedforward neural networks. Since we only use this kind of network, we implicitly mean a feedforward neural network whenever we refer to a neural network.

Definition 6.5 ([107])**.** A **neural network** is a directed, acyclic graph $G = (V, E)$ such that its nodes V (also called neurons) can be partitioned into three disjoint subsets V_{in}, V_{hidden}, and V_{out}. The nodes in V_{in} are called input neurons, the nodes in V_{hidden} are hidden neurons, and those in V_{out} are referred to as output neurons.

Each edge $(u, v) \in E$ (also called connection) is assigned a weight $w_{uv} \in \mathbb{R}$. In addition, each non-input neuron $v \in V_{\text{hidden}} \cup V_{\text{out}}$ carries an activation function $f_v : \mathbb{R} \to \mathbb{R}$ and a bias $b_v \in \mathbb{R}$.

In general, the activation functions should be non-linear. Furthermore, it is possible to restrict oneself to neural networks having a layered structure. The layers can be numbered, with the input layer often not being counted.

Definition 6.6 ([107])**.** An ***r*-layer neural network** is a neural network having the graph $G = (V, E)$, allowing the hidden neurons V_{hidden} to be partitioned into $r - 1$ disjoint subsets $V_{\text{hidden}}^{(1)}, V_{\text{hidden}}^{(2)}, \ldots, V_{\text{hidden}}^{(r-1)}$. We refer to V_{in} as the 0-th layer, $V_{\text{hidden}}^{(i)}$

as the i-th layer, and V_{out} as the r-th layer, furthermore r is also called the **depth** of the network. In addition, the edges E have to meet the following property:

$$E \subseteq \left(V_{\text{in}} \times V_{\text{hidden}}^{(1)}\right) \cup \left(\bigcup_{i=1}^{r-2} V_{\text{hidden}}^{(i)} \times V_{\text{hidden}}^{(i+1)}\right) \cup \left(V_{\text{hidden}}^{(r-1)} \times V_{\text{out}}\right). \tag{6.1}$$

If the set of edges E equals the right-hand side of Equation (6.1), then the neural network is called **fully connected**.

Unless otherwise stated, it is assumed that the networks are fully connected. These two definitions specify the syntax of neural networks. Next, we discuss a given network's semantics and explain how to calculate the network's output for a given input. A comprehensive description can be found, for example, in [78]. Suppose there are p input neurons and $x = (x_1, \ldots, x_p)$ is the input to the neural network. The output of the network can be constructed inductively by defining the output $h_j^{(k)}$ of the j-th neuron in the k-th layer having activation function f. We define $b_j^{(k)}$ as the bias of the neuron and $w_{ij}^{(k)}$ as the weight of the edge from the i-th neuron in the $(k-1)$-st layer to the j-th neuron in the k-th layer. Then, the output $h_j^{(k)}$ can be calculated as follows:

$$\begin{aligned} h_j^{(0)} &= x_j \\ a_j^{(k)} &= b_j^{(k)} + \sum_{i=1}^{n_{k-1}} w_{ij}^{(k)} h_i^{(k-1)} \\ h_j^{(k)} &= f\left(a_j^{(k)}\right), \end{aligned}$$

where n_{k-1} is the number of nodes in the $(k-1)$-st layer. The network output is then simply the output of the last layer.

Using the output of the network, the associated loss function can also be calculated. Typically, the goal is to optimize a given network's structure such that the loss function is minimized and that the problem at hand is solved as well as possible. A common approach is calculating the gradient of the loss function with respect to all weights and biases and adjusting the network parameters accordingly. We use the learning algorithm ADAM [99], which, broadly speaking, determines the mean value and the second moment of the gradient and applies them to adjust the weights and biases. This algorithm is introduced and described in Appendix B.

Neural networks, in particular, are susceptible to a phenomenon known as overfitting. Overfitting is characterized by a lack of generalization and thus poor performance of the learned model on the test set. One could also say that the model memorized the training data instead of learning the underlying structure. There are several techniques mitigating the risk of overfitting. These methods are briefly described below. For a detailed discussion, we refer to [1].

First of all, regularization can be applied. This is accomplished by adding the term $\lambda \cdot \sum_{i,j,k} \left(w_{ij}^{(k)}\right)^2$ to the loss function L, where $\lambda \in \mathbb{R}_+$ is a parameter for the regularization. This form of regularization is known as L_2-regularization. Since the weights are adjusted by the gradient, this regularization term ensures that the weights usually do not increase too much. Another possibility is to add noise to the data in each layer. Typically, a normally distributed random number is added to the data. When using the dropout method, neurons are (temporarily) omitted for training, i. e., all incoming and outgoing edges of the dropped neurons are ignored.

The last remaining technique is called early stopping. Usually, the weights are adjusted until they converge, thus minimizing the loss function on the training data. However, this does not necessarily imply that the loss function has been minimized on general data. To avoid this drawback, a fraction of the training data is put aside as validation data. The validation data is not used for training. Instead, only the behavior of the loss function, or another metric, is observed. As long as the performance on the validation data improves, the training is continued. If the performance starts to deteriorate, this is an indication of overfitting. The training can then be stopped. The methods mentioned here for reducing overfitting are all applied in the neural networks we use.

6.4 PROBSAT

PROBSAT [14] is a stochastic local search algorithm for solving SAT. The main reason why we investigate PROBSAT is that although the initial concept is already a few years old, PROBSAT-like algorithms are still highly relevant. In 2013 a PROBSAT implementation [15] won the random track, and in 2014 a parallel version of PROBSAT [16] won the parallel random track of the respective SAT competition. The SAT solver DIMETHEUS [68] won the random track in 2014 and 2016, and this solver uses PROBSAT as a core component. In the 2017 SAT competition YALSAT [27], another PROBSAT implementation, won the random track. It was not until 2018 that another

type of solver (SPARROW2RISS [13]) won the SAT competition. However, it was later shown that GAPSAT [116], a version of PROBSAT that modifies the instances, performs significantly better than SPARROW2RISS.

The fundamental structure of PROBSAT is described in Algorithm 6.1.

Algorithm 6.1 PROBSAT. This pseudocode is taken from [14].

```
1:  procedure PROBSAT(formula F, integer max_flips, integer max_tries)
2:      for i = 1 to max_tries do
3:          a ← randomly generated assignment
4:          for j = 1 to max_flips do
5:              if a satisfies F then return a
6:              C_u ← randomly selected unsatisfied clause
7:              for x ∈ C_u do
8:                  compute f(x, a)
9:              var ← random variable x according to probability f(x,a) / Σ_{z∈C_u} f(z,a)
10:             a ← a[var]
11:     return UNKNOWN
```

The algorithm works on complete assignments. Initially, a complete assignment a is chosen randomly, whereby the variables are initialized with `true` independently of each other with probability $1/2$, otherwise they are initialized with `false`.

In each step, an unsatisfied clause $C_u = (v_1 \vee \cdots \vee v_\ell)$ is randomly picked. The assignment of a variable $v_i \in C_u$ is then flipped. A **flip** of v_i refers to the change of the assignment of v_i in a, while the assignment of all other variables remains unchanged. We identify the resulting assignment with $a[v_i]$. In order to decide which variable of C_u is flipped, two essential values have to be calculated for all $v_i \in C_u$. The **break value** $\text{break}(v_i)$ denotes how many clauses are satisfied by a but are unsatisfied by $a[v_i]$. The **make value** $\text{make}(v_i)$, on the other hand, refers to the number of clauses that are unsatisfied under a, but which are satisfied under $a[v_i]$. PROBSAT then flips the variable v_i with probability $\frac{f(v_i)}{\sum_{k=1}^{\ell} f(v_k)}$. If PROBSAT is run on a 3-SAT formula, the parameterized function

$$f(v_i) = \frac{\text{make}(v_i)^{c_m}}{(\varepsilon + \text{break}(v_i))^{c_b}}$$

having parameters c_b, c_m, and ε is typically employed. For k-SAT with $k > 3$, a different function is used, but since this chapter only deals with the application to 3-SAT, this function is not described in detail. In the following, we consider two different parameter configurations. The original recommended parameterization from [14] sets $\varepsilon = 1, c_m = 0$ and $c_b = 2.3$. In the version of PROBSAT that won the SAT competition 2013 [15], the parameters are set as follows: $\varepsilon = 0.9, c_m = 0$, and $c_b = 2.06$. Since both parameterizations specify $c_m = 0$, the make value can be ignored. Also, note that the outer for-loop in Algorithm 6.1 corresponds to restarts since a new, independently chosen assignment is chosen in every iteration. However, in most implementations of PROBSAT restarts are not used, i. e., $max\_flips = \infty$. Unless otherwise stated, we also assume that no restarts are performed.

6.5 THE INSTANCE TYPES

To test PROBSAT, we use two types of instances, which are part of the SAT Competition's random track. First, uniform, random instances are discussed, followed by random instances with a hidden solution. In particular, we focus on 3-SAT instances.

For **uniform, random 3-SAT instances** with n variables and m clauses, each clause is generated by sampling three literals uniformly and independently. In principle, no statement about the satisfiability of a randomly, uniformly generated instance can be made in advance. However, there is a conjecture [157] stating that randomly generated instances with a clause-to-variable ratio m/n below a certain threshold r_s are satisfiable with high probability and otherwise are unsatisfiable with high probability. This value r_s is known as the **satisfiability threshold**. The empirical best approximation is due to [126], they estimated $r_s \approx 4.267$. Random uniformly generated instances included in the SAT Competition's random track are typically generated with this clause-to-variable ratio (see [82]).

A slightly different approach is pursued when generating instances with a **hidden solution** [21]. The concept is based on defining a complete assignment a at the beginning. Only clauses satisfied by a are added to the formula. This ensures the satisfiability of the generated formula. However, not all clauses that are satisfied by a are added. Instead, a clause C is added with a weighted probability p_i depending on the number of correct literals i in C with respect to a. This procedure is presented as pseudocode in Algorithm 6.2.

Algorithm 6.2 Pseudocode to generate a random 3-SAT formula with a hidden solution. This pseudocode has been taken and adapted from [19]. In line 4, similar to uniform instances, a random clause C is generated by uniformly and independently sampling three literals. The method in line 5 counts how many literals are satisfied in C under the hidden solution a. Depending on this number, in line 7, the generated clause C is added to the formula F with a certain probability p_i.

```
1: procedure GENERATE(integer n, assignment a, real numbers r, p1, p2, p3)
2:     F ← ∅
3:     while |F| < r · n do
4:         C ← generateRandom3SatClause(n)
5:         i ← numberOfSatisfiedLiterals(C, a)
6:         if i > 0 then
7:             with probability p_i: F ← F ∪ {C}
8:     return F
```

Depending on the parameter combination of Algorithm 6.2, different types of instances are generated, which are in turn of varying difficulty for PROBSAT. Here, we focus on the configurations used in the SAT Competition 2018 [84]. Three different configurations are identified; we call them `barthel`, `komb` and `qhid`. The parameters of these instance types are listed in Table 6.1.

	p_1	p_2	p_3	r
`barthel`	0.163	0.057	0.221	4.300000
`komb`	0.870	0.479	0.130	5.205555
`qhid`	0.300	0.090	0.027	5.500000

Table 6.1: This table shows three randomly generated instance domains with a hidden solution. The parameters p_1, p_2, p_3 and r refer to the inputs for Algorithm 6.2. The number of variables n per instance ranges from 200 to 400. The parameter configurations are extracted from the instances of the SAT Competition 2018 [84].

6.6 PROBSAT ON UNIFORM INSTANCES

We start by examining PROBSAT on uniform randomly generated instances. For this purpose, we use the PROBSAT implementation provided in [11], supplemented by the possibility to use Luby's strategy (cf. Definition 5.1). In the following, we use $c_m = 0$, $c_b = 2.3$ and $\varepsilon = 1$ for the parameters of PROBSAT as suggested in

the paper that introduced PROBSAT [14]. It should be noted that this parameter configuration is found by tuning the parameters specifically on uniformly generated instances.

We shall see that the Weibull distribution, in particular, is well-suited to describe the runtime behavior of PROBSAT on uniform instances. Indeed, the Weibull distribution is considerably better suited than the lognormal distribution, which was used in [5] to describe the runtime of PROBSAT. In addition, we also investigate whether this version of PROBSAT can be further improved by restarts and find that there is still much potential for improvement by using a suitable restart strategy.

Based on this insight, a machine learning pipeline is designed, which proceeds in two steps. First, a runtime distribution is predicted, and secondly, an optimal restart time is calculated using the techniques from Chapter 4. This approach is evaluated on previously unseen instances. The average speedup factor is 1.06. This result is statistically significant. In the following, the details are described.

6.6.1 *Instance Specification*

First of all, we describe which instances have been used for the subsequent experiments. For this purpose, instances with 1500 to 2500 variables near the satisfiability threshold have been considered. More precisely, the clause-to-variable ratio ranges from 4.204 to 4.272. The instances are uniform, randomly generated instances. The generator KCNFGEN [66] is used for the creation.

As can be inferred from Algorithm 6.1, PROBSAT is an incomplete solver. That means that it can only verify the satisfiability of a given instance. As mentioned before, PROBSAT does not perform restarts by default, thus $max\_flips = \infty$ in Algorithm 6.1. This means PROBSAT never stops on an unsatisfiable instance. For the following experiments, no timeout is used. In other words, the algorithm is allowed to run until a satisfying assignment is detected. For this reason, unsatisfiable instances must be excluded in advance. For this purpose, the SAT solver DIMETHEUS [67, 68] is utilized. At the time of performing the experiments, DIMETHEUS was the best solver for randomly generated instances, as witnessed by the results of the SAT Competition 2016 [83]. Finally, only those instances that DIMETHEUS recognized as satisfiable were used. Note that this can be understood as a filter, but since PROBSAT is an incomplete solver, the only alternative is using a timeout. Altogether, 2400 instances were created, of which 1632 are satisfiable according to DIMETHEUS. The set

of these 1632 instances is denoted by $\mathcal{C}$ in the following. The experiments described below are conducted on $\mathcal{C}$.

6.6.2 *Assessment of the Runtime Distribution*

This section aims to explore which random distributions are capable of describing the behavior of PROBSAT on uniform instances. In Section 4.4.1, 4.4.2 and 4.4.3, we extensively investigated the lognormal, Weibull, and the generalized Pareto distribution in terms of their characteristics with respect to restarts. The introductions of these sections also pointed out that these three distribution families are common candidates to describe algorithms' behavior. Therefore, we also consider these three distribution types in this section. Additionally, it should be emphasized that Arbelaez et al. [5] studied runtime distributions for PROBSAT and identified the lognormal distribution as a suitable distribution.

In the following, we are particularly interested in deducing conclusions regarding restarts based on the probability distributions. Special attention has to be paid to the Weibull and the generalized Pareto distribution since it has been shown by Theorem 4.23 and Theorem 4.27 that the Weibull and generalized Pareto distributions have a special property concerning restarts: If restarts are useful, they should be performed as early as possible. In the case of PROBSAT, this would imply that immediately after randomly generating an assignment, a restart is necessary, and a new assignment has to be generated. This is an unreasonable procedure because it would suggest that choosing an assignment at random has a higher chance of finding a satisfying assignment than the algorithm itself. Hence, we conclude that the two-parameter Weibull and generalized Pareto distribution are inappropriate for our purpose. Instead, we use a three-parameter Weibull and generalized Pareto distribution having an additional location parameter besides the shape and scale parameter.

The influence of location parameters has already been explored in general in Section 4.3.2. In particular, Corollary 4.17 demonstrates that the influence of such a location parameter on the usefulness condition of restarts is at most linear. However, in the case of the Weibull and the generalized Pareto distribution, a much stronger statement can be shown. For these two distributions, the location parameter does not influence the usefulness of restarts.

Corollary 6.7. *Assume X is a Weibull distributed random variable with shape parameter $k < 1$. Also, let $b \in \mathbb{R}_+$ be any positive real number. Then, restarts are useful for $Y = X + b$.*

Proof. According to [61], X has a long-tailed distribution. However, in this case, Y is long-tailed as well (also according to [61]).

Using Definition 2.42 and Definition 2.39, we obtain the pdf f_Y and the cdf F_Y of Y:

$$F_Y(x) = 1 - e^{-(\frac{x-b}{a})^k},$$
$$f_Y(x) = \frac{k}{a} \cdot \left(\frac{x-b}{a}\right)^{k-1} \cdot e^{-(\frac{x-b}{a})^k}.$$

It is obvious, that the limit $\lim_{x\to\infty} x^2 \cdot f_Y(x)$ exists because for $x \to \infty$ the term $e^{-(\frac{x-b}{a})^k}$ approaches zero faster than $x^2 \cdot \left(\frac{x-b}{a}\right)^{k-1}$ approaches infinity. We also consider the limit of the hazard rate function $r_Y(x) = f_Y(x)/\big(1 - F_Y(x)\big)$ (see Definition 4.9):

$$\lim_{x\to\infty} r_Y(x) = \lim_{x\to\infty} \frac{f_Y(x)}{1 - F_Y(x)}$$
$$= \lim_{x\to\infty} \frac{\frac{k}{a} \cdot \left(\frac{x-b}{a}\right)^{k-1} \cdot e^{-(\frac{x-b}{a})^k}}{e^{-(\frac{x-b}{a})^k}} = \lim_{x\to\infty} \frac{k}{a} \cdot \left(\frac{x-b}{a}\right)^{k-1} = 0.$$

The last equation is valid because $k < 1$. All in all, the conditions of Theorem 4.12 are met, and therefore restarts are useful for Y. □

A similar statement also applies to the generalized Pareto distribution.

Corollary 6.8. *Assume X is a generalized Pareto distributed random variable with shape parameter $k > 0$. Furthermore, let $b \in \mathbb{R}_+$ be any positive real number. Then, restarts are useful for $Y = X + b$.*

Proof. The cdf F_Y and pdf f_Y of Y are derived from Definition 2.43 and Definition 2.39:

$$F_Y(x) = 1 - \left(1 + \frac{k \cdot (x-b)}{a}\right)^{-\frac{1}{k}},$$
$$f_Y(x) = \frac{1}{a} \cdot \left(1 + \frac{k \cdot (x-b)}{a}\right)^{-\frac{1}{k}-1}.$$

We start by analyzing the limit of the hazard rate function r_Y (see Definition 4.9):

$$\lim_{x\to\infty} r_Y(x) = \lim_{x\to\infty} \frac{f_Y(x)}{1-F_Y(x)} = \lim_{x\to\infty} \frac{\frac{1}{a}\cdot\left(1+\frac{k\cdot(x-b)}{a}\right)^{-\frac{1}{k}-1}}{\left(1+\frac{k\cdot(x-b)}{a}\right)^{-\frac{1}{k}}}$$

$$= \lim_{x\to\infty} \frac{1}{a}\cdot\left(1+\frac{k\cdot(x-b)}{a}\right)^{-1} = 0$$

This implies that Y is long-tailed as per Lemma 4.11. Furthermore, according to Lemma 4.25, the expected value $E[X]$ and thus $E[Y]$ is infinite, if $k \geq 1$. In this case, the conditions of Theorem 4.12 are met, and the desired property follows.

We, therefore, limit ourselves to $k \in (0,1)$ and consider the limit of $x^2 \cdot f_Y(x)$:

$$\lim_{x\to\infty} x^2\cdot f_Y(x) = \lim_{x\to\infty} \frac{x^2}{a\cdot\left(1+\frac{k\cdot(x-b)}{a}\right)^{1+\frac{1}{k}}}$$

$$= \lim_{x\to\infty} \frac{2\cdot x}{a\cdot(1+k)\cdot\left(1+\frac{k\cdot(x-b)}{a}\right)^{\frac{1}{k}}}$$

$$= \lim_{x\to\infty} \frac{2}{a\cdot(1+k)\cdot\left(1+\frac{k\cdot(x-b)}{a}\right)^{\frac{1}{k}-1}} = 0.$$

Here, L'Hospital's rule is applied twice. The final equality follows since by assumption $k \in (0,1)$. Thus the conditions of Theorem 4.12 are also satisfied in this case, and hence restarts are useful for Y. □

In other words, the usefulness of restarts can be judged purely based on the shape parameter of the Weibull and the generalized Pareto distribution, even if an additional location parameter is employed. The computation of the optimal restart time can then be carried out by using Corollary 4.18.

6.6.3 *Empirical Distributions*

In the following, the runtime distributions of PROBSAT on instances in $\mathcal{C}$ are examined in detail. For this purpose, each instance $i \in \mathcal{C}$ is solved several times. Due to the high computational effort, the calculation is performed in parallel on the machines of the bwUniCluster as well as two internal servers. Owing to this highly

heterogeneous setup, measuring the runtime in seconds is an unreliable measure. This is because the runtime in seconds depends heavily on which computer the instance is solved. To overcome this issue, we instead measure the runtime in terms of the number of performed flips until a satisfying assignment is found.

The distribution of the instances to the available cores is implemented by using SPUTNIK [173]. As a side note, the difficulty of the instances in $\mathcal{C}$ is highly diverse. The easiest instances can be solved in a fraction of a second, while the hardest instances take several days to solve.

We associate the runtime distribution (in flips) of PROBSAT on an instance $i \in \mathcal{C}$ with the random variable X_i. A priori, the exact cdf of X_i is entirely unknown, which is why we employ the empirical cumulative distribution function. More precisely, we execute PROBSAT 300 times on each instance $i \in \mathcal{C}$ and thereby obtain 300 independent, identically distributed observations from which we determine the ecdf $\hat{F}_{300}$ of X_i. We also obtain parameter estimates for the lognormal, Weibull, and generalized Pareto distribution by means of maximum likelihood estimation from these observations. We compare these fitted distributions with the ecdf $\hat{F}_{300}$ using the KS test[1] (see Definition 6.4). If the p-value is greater than 0.05, then we presume that the fitted distribution adequately describes the observations, or alternatively, the fitted distribution passed the KS test. In addition, the winning distribution is the fitted distribution having the highest p-value. This indicates that this distribution reflects the data most accurately.

	Weibull	lognormal	generalized Pareto	none
Passed	1529	1273	1382	10
Won	541	606	475	10

Table 6.2: The number of instances where the maximum likelihood fit of the respective distribution passed the KS test at a significance level of 0.05. A distribution "won" the test, if it passed with the highest p-value.

Table 6.2 illustrates which fitted distributions passed the KS test and which distributions won. Remarkably, only 10 of the 1632 instances in $\mathcal{C}$ cannot be adequately represented by any of the three distribution types. As can be observed, the runtime distributions of 1273 instances are described by the lognormal distribution; this

1 If one were to use the KS test as a standard hypothesis test, a modification would be required at this point. This is because the KS test overestimates the p-value on fitted distributions (compare, e. g., [140]). However, as already mentioned above, we are only interested in whether a fitted distribution represents an adequate fit. For this purpose, the KS test statistics and/or the associated p-value can be used as a measure.

corresponds to 78.0% of the instances. This confirms an observation by Arbelaez et al. [5]. They studied the runtime distributions of PROBSAT on a set of 500 instances. Using the same methodology, i. e., applying the KS test with a significance level of 0.05, the lognormal distribution describes 389 instances (77.8%) of this set.

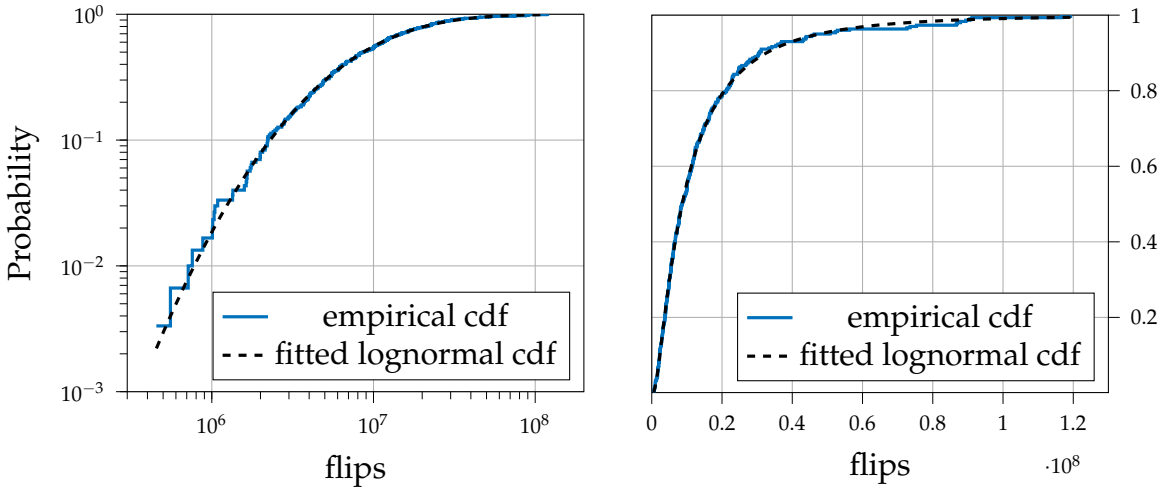
However, as can be inferred from Table 6.2, the Weibull distribution provides a substantial improvement in this respect. Altogether 93.7% of the instances are adequately represented by the Weibull distribution. If one were to lower the significance level to 0.01, then the Weibull distribution could be used to describe as much as 97% of the instances. Nevertheless, the lognormal distribution is the type of distribution, which won most often.

As a side note, the exponential distribution is sometimes employed to describe the runtime distribution of algorithms (e. g., [6, 89]). We, therefore, initially also considered the exponential distribution as a candidate. However, it turned out that the exponential distribution without location parameters only describes 52.7% of the runtime distributions adequately. By adding a location parameter, this percentage can be increased to 67.7%, but this is still substantially less than the other three examined distributions. As a consequence, we discarded the exponential distribution as a candidate.

By means of two diagrams, we illustrate the possible strengths and weaknesses of representing the runtime distributions with the distributions described above.

Figure 6.2 shows a lognormal fit. As indicated in the graph, it is an exceptionally good fit. Even the probabilities for particularly short runs are accurately represented, as shown in Figure 6.2a. Such a representation is beneficial because it implies that restart times can be determined by means of Theorem 4.14 and that these will yield good results. The good capability for our application is reflected in the high p-value.

On the other hand, Figure 6.3 shows a case where the fit can lead to misconceptions about the suitability of restarts. In this case, a Weibull distribution is fitted to the data. As the linear-scale plot (Figure 6.3b) shows, the probability that PROBSAT finds a solution is underestimated almost over the entire interval. While the fit looks acceptable in the left tail at first glance, the log-scale plot (Figure 6.3a) reveals that the probability is misjudged by almost an entire order of magnitude. Generally, the probability for these short runs is not well reflected by the Weibull distribution. Since it is a Weibull distribution with a low shape parameter k, one would use this fitted distribution to choose a very low restart time (compare Theorem 4.23 and Corollary 4.18). However, this would most likely lead to a significant deterioration of PROBSAT's performance, mostly due to the bad reflection of the probability for



(a) log-scale **(b)** linear scale

Figure 6.2: The empirical distribution function and the lognormal fit of a typical instance. The fitted lognormal distribution has shape parameter $\sigma = 1.036$ and scale parameter $\mu = 15.974$. The KS test statistic is $D_{300} = 0.0201$ and has an associated p-value of $p = 0.9996$. This figure is based on [113].

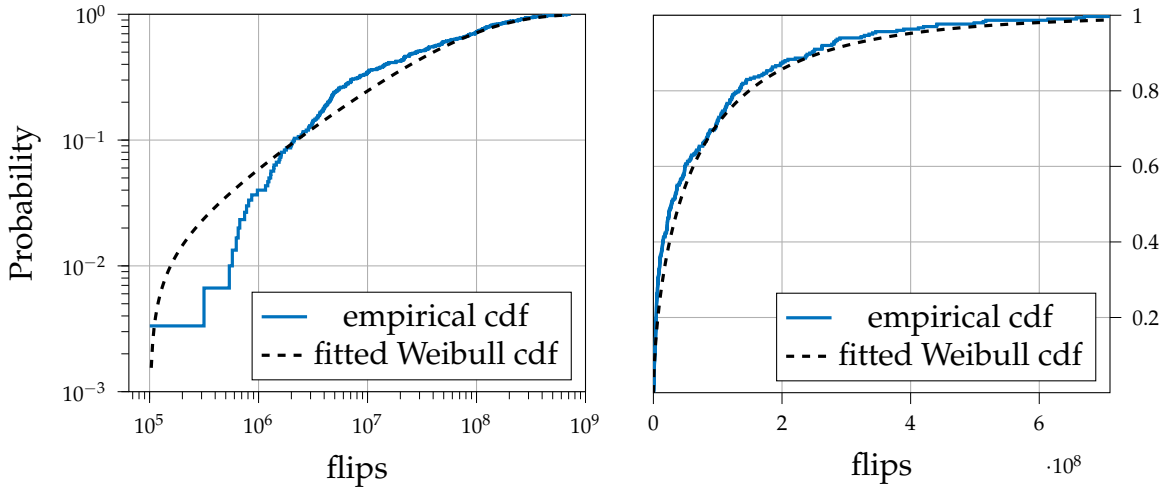
short runs. However, this unsuitability is also expressed by the p-value of the KS test, as p is approximately 0.002. In fact, this is one of the ten instances not adequately described by any of the three distributions; this is also reflected in the respective p-values.

6.6.4 *Estimating the Potential of Restarts*

This section aims to determine if and how much restarts can improve PROBSAT. Secondly, we evaluate how well the random distributions described in Section 6.6.3 are suited to identify an appropriate restart strategy.

The advantage of using a random distribution such as the lognormal or Weibull distribution to calculate the optimal restart time is that two or three parameters define the distribution's complete behavior. Consequently, the NP-hardness of determining an optimal restart time (see Theorem 3.3) is not a concern.

On the other hand, the behavior of an algorithm like PROBSAT can rarely be explained by a two or three-parameter distribution. If a restart time is calculated from such a fitted distribution, the behavior is ultimately not optimal. This implies that just by using a fitted distribution, some potential is already lost. To assess the



(a) log-scale **(b)** linear scale

Figure 6.3: The empirical distribution function and the Weibull fit. The fitted Weibull distribution has shape parameter $k = 0.643$, scale parameter $a = 70949599.692$, and location parameter $b = 100538$. The KS test statistic is $D_{300} = 0.1085$ and has an associated p-value $p = 0.0016$. This figure is based on [113].

distribution's quality, we first identify the empirically best restart time without using a fitted distribution.

For this purpose, assume there are n observed runtimes $(t_{(1)}, t_{(2)}, \ldots, t_{(n)})$. Without loss of generality, suppose that $t_{(i)}$ is the i-th smallest sample, i. e.,

$$t_{(1)} \leq t_{(2)} \leq \cdots \leq t_{(n)}.$$

Furthermore, let $\hat{F}_n$ be the ecdf associated with these observations. Then, an estimate $\hat{x}_t$ for the expected runtime with restarts after t flips can be given:

$$\hat{x}_t = \frac{1 - \hat{F}_n(t)}{\hat{F}_n(t)} \cdot t + \frac{\sum_{i:\, t_{(i)} \leq t} t_{(i)}}{|\{i \mid t_{(i)} \leq t\}|}. \tag{6.2}$$

The minimal $\hat{x}_t$ is denoted $\hat{opt}$. In other words, $\hat{opt}$ is an estimate of the optimal expected runtime under restarts.

The **speedup** is a well-known indicator for comparing the performance of two algorithms. Let A_k be a random variable describing the runtime distribution of an algorithm on an instance k. Likewise, let B_k be a random variable describing another

algorithm's runtime distribution on the same instance. In our case, we define the speedup $s_k^{(A,B)}$ on instance k, as shown below:

$$s_k^{(A,B)} = \frac{E[A_k]}{E[B_k]}. \tag{6.3}$$

We generally use estimates for the expected value. For instance, the effect of restarts on an instance k can be compared by observing the ratio of $o\hat{p}t$ and $\overline{x}$, where $\overline{x}$ is the arithmetic mean of the observed runtimes. Expressed as a formula, the speedup by restarts s_k^{restarts} on instance k is estimated by:

$$s_k^{\text{restarts}} = \frac{\overline{x}}{o\hat{p}t}.$$

However, this only covers the possible speedup on a single instance k. To calculate the average speedup over the entire set $\mathcal{C}$ the geometric mean has to be calculated [58]. The geometric mean $\overline{x}_{\text{geom}}$ of n positive, real numbers $x_1, \ldots, x_n \in \mathbb{R}_+$ is defined as follows:

$$\overline{x}_{\text{geom}} = \left(\prod_{i=1}^{n} x_i \right)^{\frac{1}{n}}.$$

With these definitions, the average speedup due to restarts can be calculated. First, we compare the expected runtimes $o\hat{p}t$ of the optimal restart strategy compared to the strategy without restarts $\overline{x}$. This translates into an average speedup of 1.393. We use this value primarily as a benchmark. That is to say, as an upper bound to determine how much restarts can potentially improve PROBSAT. Nevertheless, this clearly demonstrates that PROBSAT can be considerably improved by using a restart strategy.

Next, we consider the average speedups using the lognormal, Weibull, and generalized Pareto distribution. For this purpose, we proceed as follows: The fitted parameters are used to decide if restarts are useful. To recap: For the lognormal distribution, restarts are always useful (Theorem 4.20), whereas, for the Weibull (Corollary 6.7) and the generalized Pareto distribution (Corollary 6.8), the respective shape parameter has to be suitable.

If restarts are useful, the restart time t^* is calculated using Theorem 4.14 or Corollary 4.18. An estimate of the expected runtime is then calculated by plugging t^* into

Equation (6.2). This allows the calculation of the speedup and the average speedup over all instances.

We also consider strategies that have access to more than one distribution. To calculate the speedup, we either use the distribution having the higher p-value in the KS test or, alternatively, the distribution providing the higher speedup according to Equation (6.2). In both cases, this decision is made on a per-instance basis. We consider all possible combinations of the lognormal, the Weibull, and the generalized Pareto distribution. The findings are shown in Table 6.3. For the sake of brevity, we abbreviate the lognormal distribution with L, the Weibull distribution with W, and the generalized Pareto distribution with GP in the table.

	KS test	speedup
$\emptyset$(baseline)	1.393	1.393
$\{L\}$	1.154	1.154
$\{W\}$	1.174	1.174
$\{GP\}$	1.157	1.157
$\{L, W\}$	1.253	1.284
$\{L, GP\}$	1.162	1.184
$\{W, GP\}$	1.187	1.285
$\{L, W, GP\}$	1.200	1.300

Table 6.3: The calculated average speedups for different sets of probability distribution types. For sets containing several distributions, the distribution with the higher p-value in the KS test is chosen in the middle column (KS test); this distribution is then used to calculate the speedup. Likewise, in the right column (speedup), the distribution with the higher speedup is selected. For the purpose of comparability, the row containing the empty set indicates the speedup that was determined empirically from the data.

As is evident, some of the potential is already lost through the fitting process. However, this behavior was to be expected, as already discussed at the beginning of this section. Nevertheless, a single distribution type is hardly sufficient to determine good restart times. Promising candidates are the sets containing the Weibull distribution. Especially, the combination $\{L, W\}$ is of interest since the results are encouraging in both the KS test column and the speedup column.

6.6.5 *A Machine Learning Pipeline*

The speedup is estimated from data obtained by executing PROBSAT without any restarts in the previous sections. Although this offers insight into the potential of restarts, it is not clarified whether this information can be retrieved in practice. Most notably, it is not apparent how the fitted distributions' parameters can be obtained without first solving the instance.

Arbelaez et al. [4] use machine learning techniques to predict the runtime distribution, but their focus is mainly on predicting the expected runtime if the corresponding algorithm is parallelized. Eggensperger et al. [51] propose a method based on neural networks. These neural networks predict the parameters of the runtime distribution for previously unseen instances. We employ a comparable methodology while extending and refining the design for our purposes. Specifically, we empirically show that the chosen methods are well-suited for computing useful restart times.

The parameters are estimated in two main phases. First, a random forest predicts the type of distribution. Secondly, based on the estimated distribution type, a forecast of the distribution parameters is obtained by a neural network specially designed for this distribution type.

Each candidate machine learning model M is assessed with a test set $\mathcal{D}_{\text{test}}$. For this purpose, the average speedup s_M on $\mathcal{D}_{\text{test}}$ is calculated by using the restart times predicted by M. The value s_M serves as a tool for direct comparison to evaluate the quality of the models.

The test set $\mathcal{D}_{\text{test}}$ consists of several instances from $\mathcal{C}$ and the associated parameter estimates. However, we did not choose the test set uniformly at random because the shape parameter is of particular significance for deciding if and when to restart (compare Chapter 4). Therefore, we deliberately put more emphasis on instances with an especially high or a low shape parameter. The underlying idea is that a model should be capable of accurately determining the shape parameter and that this is ultimately more valuable for the decision to restart than the scale and location parameters. The test set $\mathcal{D}_{\text{test}}$ contains 162 instances from $\mathcal{C}$. For the training, only the remaining 1470 instances are used; we refer to them as the training set $\mathcal{D}_{\text{train}}$.

After trying several combinations of distribution types, we found that the combination of lognormal and Weibull distribution leads to the highest speedup. While the potential speedup due to the generalized Pareto distribution, as shown in Table 6.3,

is decent, neural networks were unable to estimate the parameters of the generalized Pareto distribution accurately. As a result, the addition of the generalized Pareto distribution to the machine learning pipeline deteriorated the results. For this reason, we decided to drop the generalized Pareto distribution from consideration.

In the following, we describe which features are used, as well as the structure of the random forest and the neural networks.

6.6.6 *Feature Extraction*

The features are obtained from the feature extractor SATZILLA [183]. SATZILLA is invoked with the parameters `-base`, `-ls`, and `-lobjois` (as inspired by [6]). A total of 81 features are generated. An exact definition of these features is provided in [182].

In our case, we found that the neural networks benefit noticeably if the features exhibit a minimum degree of variance. In order to draw comparable conclusions about the variance, the following transformation is performed for each observation x of a feature:

$$z = \frac{x - \min\limits_{y \in \mathcal{D}_{\text{train}}} (y)}{\max\limits_{y \in \mathcal{D}_{\text{train}}} (y) - \min\limits_{y \in \mathcal{D}_{\text{train}}} (y)}. \tag{6.4}$$

After this transformation, each feature has a similar scale. We confined ourselves to those features having a sample variance of at least 0.05. In addition, four other handpicked features are used. This reduces the total number of features to 34.

The remaining features concern the formula's structure, for example, how many variables and clauses are present or how close the formula is to a Horn formula. There are also features utilizing a graph representation of the given formula. Finally, the algorithmic behavior on the given formula is examined. Two local search solvers are tested on the formula. The corresponding features mostly concern the maximal number of clauses satisfied by the algorithms and the number of local search steps necessary to reach this maximum. A detailed list of the features can be found in Appendix C. We wish to emphasize that the transformation from Equation (6.4) was performed only for the features' pre-selection. A different type of normalization is applied in the components of the machine learning pipeline.

6.6.7 *Random Forest*

We determine the predicted distribution type using a random forest for classification. As the label, we take the distribution that performed better in the KS test, i. e., the distribution with the higher p-value. The features in the Random Forest are not normalized. Different types of normalization were also tested, but these did not affect the quality of the random forest.

The quality of the random forest is controlled via the potential speedup. For this purpose, the maximum likelihood estimation of the parameters for the predicted distribution type is used. Based on these estimated parameters, the restart time is computed, and thus the potential speedup can be assessed. In order to underline this aspect: The random forest only predicts the distribution type, not the precise parameters. We use the previously calculated maximum likelihood estimate for the predicted distribution type to calculate the potential speedup. The goal is to identify the highest possible speedup based on the random forest. In other words, a misclassification is considered less severe if a good restart time can be calculated even with the wrong distribution.

The average speedup is established on the basis of ten-fold cross-validation. For ten-fold cross-validation, the training set $\mathcal{D}_{\text{train}}$ is split into ten sets of roughly equal size $\{\mathcal{D}_{\text{train}}^{(1)}, \ldots, \mathcal{D}_{\text{train}}^{(10)}\}$. Nine of these sets are then used for training, and the resulting random forest is then evaluated on the remaining set. This procedure is repeated ten times, such that each of the sets $\mathcal{D}_{\text{train}}^{(i)}$ is used exactly once as a validation set. Finally, the result of the cross-validation is the average speedup over the ten iterations.

It became apparent that the method of constructing the random forest only exerts a limited influence on the potential speedup measured by the cross-validation. Finally, we chose to use a random forest consisting of 50 trees. The cross-validation results in an average speedup of 1.218 for this model. This equates to 97.2% of the possible speedup as indicated in the KS test column of the combination $\{L, W\}$ in Table 6.3. Other parameter combinations for the random forest have also been tested. However, as already mentioned earlier, the results regarding the speedup are quite stable. An assessment of the random test with respect to the classification error is given in Appendix C.1.2.

The construction of the random forest is implemented in Python by means of SCIKIT-LEARN [142]. The final random forest, which is subsequently used for the machine learning pipeline, is trained on the entire training set $\mathcal{D}_{\text{train}}$.

6.6.8 *Neural Networks*

The parameters of the Weibull and lognormal distribution are estimated via neural networks. In the case of the lognormal distribution, the scale and shape parameters are predicted, while the Weibull distribution requires an additional location parameter. Both distribution types use a separate neural network.

Normalized features are especially important for the neural networks. To achieve this, the following transformation is applied to each observation x of a feature:

$$z = \frac{x - \overline{x}}{s},$$

where $\overline{x}$ is the sample mean, and s is the sample standard deviation of the associated feature. Due to this transformation, all features have mean 0 and variance 1. The neural networks converge faster using data scaled in this manner.

The quality of the networks is initially monitored by cross-validation, as in the case of the random forests. The potential average speedup is again the metric by which the quality is measured. This allows the comparison of several alternative network designs, including multiple variants of the loss function and the hyperparameters involved in the learning process. Several such variants are tested. The results turned out to be quite sensitive to the choice of the neural network design. In the following, we describe the network's final configuration and how the findings from the design phase influenced it.

Initially, the goal was to predict all parameters, i. e., the shape, the scale, and the location parameter, using the same network. However, we found that using this method causes the estimate of the location parameter to be of poor quality; this prediction is often incorrect by several orders of magnitude. This insight led to the use of an independent network for estimating the location parameter. A common choice for the loss function L is the root mean square error RMSE,

$$\text{RMSE}(\hat{\theta}) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (\hat{\theta}_i - \theta_i)^2},$$

where n is the number of observations, θ_i are the labels and $\hat{\theta}_i$ are the predicted values. For the neural network determining the location parameter, we chose the root mean square error as the loss function. This improved the overall prediction quality of the location parameter. The shape and scale parameters are estimated in a joint network. A loss function based on the root mean square error would not adequately reflect the correlation between these two parameters; therefore, another loss function is needed. One option is the potential speedup, but for technical reasons, it is challenging to realize. To calculate the speedup, a lot of additional data is required leading to an unacceptably slow learning process. Another possibility is the log-likelihood function as proposed by Eggensperger et al. [51]. We tried using this function, but the resulting predictions of the shape parameter were of poor quality in our case. Accordingly, this led to an unsuitable prediction of the restart time and thus to low speedup values.

Finally, we chose a loss function resembling the KS test statistic. Here, we provide an example of the loss function L for the lognormal distribution. We use the following labels for the neural network: A sampled runtime x, the associated value of the ecdf $\hat{F}_{300}(x)$, and the maximum likelihood estimate σ of the shape parameter. The loss function L is defined by

$$L(\hat{\sigma}, \hat{\mu}) = \left| F(x \mid \hat{\sigma}, \hat{\mu}) - \hat{F}_{300}(x) \right| + \left| \sigma - \hat{\sigma} \right|, \tag{6.5}$$

where $\hat{\sigma}, \hat{\mu}$ are the estimates of the shape and scale parameters generated by the neural network. Furthermore, $F(x \mid \hat{\sigma}, \hat{\mu})$ is the value of the cdf at x of a lognormal distributed random variable with shape $\hat{\sigma}$ and scale $\hat{\mu}$.

The second part of Equation (6.5), $|\sigma - \hat{\sigma}|$, substantially improves the prediction quality of the shape parameter at the expense of a slight worsening of the scale parameter. However, as already mentioned, the shape parameter is significantly more important for our purposes. Overall, this leads to better restart times, and therefore, also to a higher speedup.

It is worth mentioning that the maximum likelihood estimate of the scale parameter μ is not used as a label. However, by using $|F(x \mid \hat{\sigma}, \hat{\mu}) - \hat{F}_{300}(x)|$ in the loss function, the neural network is directed towards μ, so it does not have to be used explicitly as a label. We also tried different variations of this loss function, but ultimately the loss function, given in Equation (6.5), yielded the most robust results. It is also worth mentioning that an analogous loss function is used for the Weibull distribution. The tangent hyperbolic $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ is primarily used as

the activation function of the neurons. We employ two hidden layers for each neural network. The number of neurons for the location, Weibull, and lognormal network is listed in Table 6.4.

layer	location	Weibull	lognormal
input	34 neurons	34 neurons	34 neurons
hidden 1	14 neurons	14 neurons	14 neurons
hidden 2	1 neuron	7 neurons	7 neurons
output	1 neuron	2(4) neurons	2(3) neurons

Table 6.4: The number of neurons of the location, Weibull, and lognormal neural network, separated by layer. The reason for the values in brackets is explained in the text.

As can be seen from the table, the neural networks for Weibull and lognormal distribution contain four and three neurons, respectively. This has a purely technical reason: In the loss function for the lognormal distribution, we used three labels, although only two values, the shape and scale parameter, are to be estimated. We implemented this by adding a third neuron in the output layer, but its output is not used. In addition, the maximum likelihood estimate of the location parameter is provided as a label for the Weibull network. Again, the output of the corresponding neuron is not used.

The neural networks are quite susceptible to overfitting. For this reason, countermeasures are necessary for the learning process. We used L_2-regularization, Gaussian noise, and dropout (also see Section 6.3.2). The precise parameters for these countermeasures can be found in Appendix C.1.3.

The final models' average speedups, measured by using cross-validation, are 1.127 for the lognormal model and 1.137 for the Weibull model. However, the speedup of the Weibull model warrants some caution. There are cases where a restart time lower than the shortest observed runtime is predicted. This is because a Weibull distribution having a low shape also implies low restart times. In this case, however, the estimated expected runtime (Equation (6.2)) is undefined. In such a situation, these undefined values are not included in the average speedup.

Additionally, it should be mentioned that the neural networks are implemented in Python using the framework KERAS [40] with TENSORFLOW backend [124].

Up to this point, the quality of the individual components is determined by means of cross-validation. However, the quality of the entire pipeline is still to be assessed. For this purpose, the test set $\mathcal{D}_{\text{test}}$ is used. By successively running the components

of the pipeline, it is possible to decide if and when to restart. The resulting restart strategy has an average speedup of 1.157 on the test set. Therefore, a noticeable speedup on the test set.

6.6.9 *Evaluation of the Pipeline*

At the end of the last section, it is observed that the proposed machine learning pipeline can be employed to predict useful restart times on the test set. However, in each step, the results are evaluated using the data from the original data acquisition. This may lead to distorted findings regarding the speedup. We, therefore, intend to re-evaluate the pipeline by another approach. For this purpose, we first generate 100 new satisfiable instances by the same approach as described in Section 6.6.1. These new instances are denoted by $\mathcal{K}$. We wish to stress that the instances in $\mathcal{K}$ are neither in the training set $\mathcal{D}_{\text{train}}$ nor in the test set $\mathcal{D}_{\text{test}}$. Therefore, the results reported here are independent of the preceding findings.

Two strategies are being compared with each other. First, the original version of PROBSAT is considered; since the original version does not perform restarts, we will call this strategy "no restarts".

Secondly, a restart strategy based on the machine learning pipeline is employed. This is illustrated as a flowchart in Figure 6.4. Again, a random forest is used to decide whether a lognormal or a Weibull distribution is the more appropriate model. Then, the parameters are determined with the corresponding neural network. In the case of a lognormal distribution, restarts are always useful (Theorem 4.20), whereas for Weibull distributions, restarts are only useful if the shape parameter is less than 1.0 (Corollary 6.7). If restarts are useful according to the predicted parameters, the restart time is determined by means of Theorem 4.14. We call this strategy "pipeline restarts".

Each of the two strategies is executed 100 times on all instances in $\mathcal{K}$. Both strategies are compared in terms of their average runtime (still in flips). Each run has a timeout of 10^{11} flips. On our system, this timeout corresponds to an execution time of well over one day per run. Only one of the 100 instances is affected by this timeout. The no restarts strategy was able to solve this instance in 28 cases, while the pipeline restarts strategy found a solution in 72 cases. The following analysis only concerns the 99 instances without a timeout.

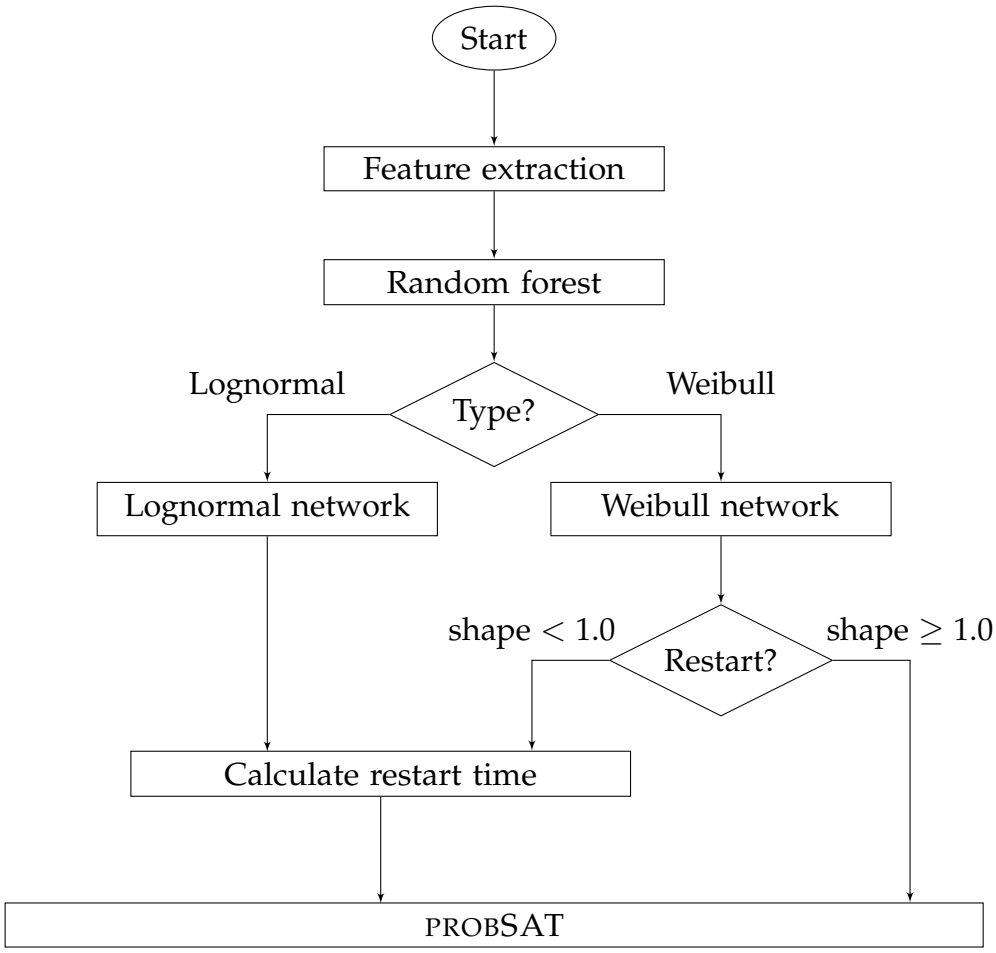


Figure 6.4: The "pipeline restarts" strategy. The random forest and the neural network are described in Section 6.6.7 and Section 6.6.8.

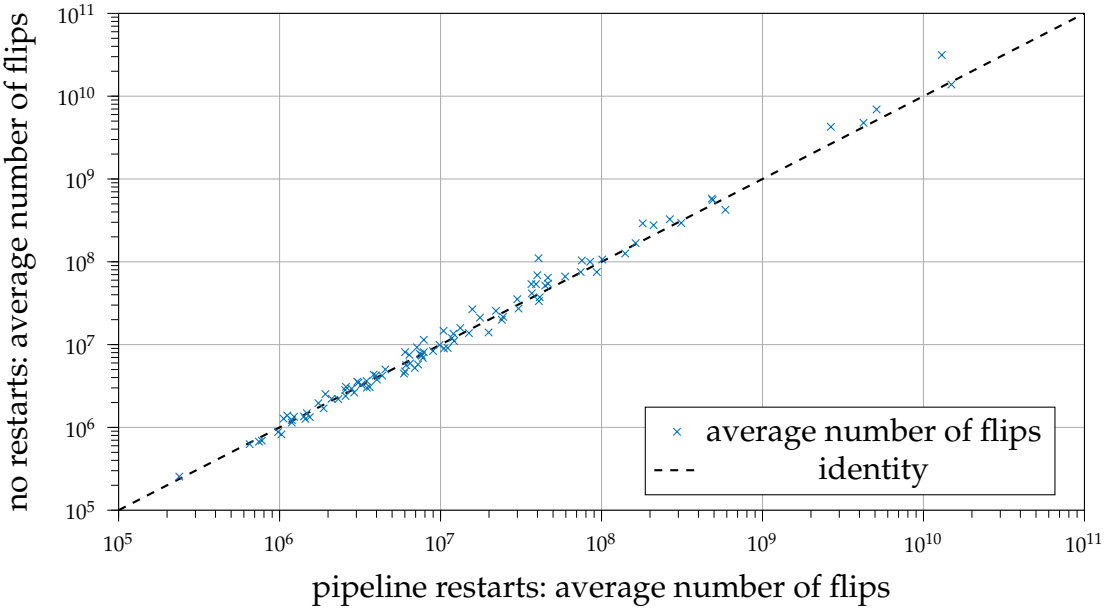


Figure 6.5: The pipeline restarts strategy is compared with the no restarts strategy. The dots are the logarithmically scaled average runtimes. Each marker indicates the average runtimes on a single instance. The average runtimes of the pipeline restarts strategy are shown on the x-axis; those of the no restarts strategy are on the y-axis. The pipeline restarts strategy is more efficient if the marker lies above the dashed line; otherwise, its competitor is more efficient. This figure is based on [112].

The comparison between the two strategies is illustrated in Figure 6.5. Upon initial inspection, there is no obvious difference between the two strategies. Especially in the range up to about 10^7 flips, the points seem to cluster around the identity line. There are two main explanations for this. First, the parameters of PROBSAT are tuned without restarts on uniform instances. For this tuning process, short timeouts between 10 and 100 seconds were used. Consequently, these parameters work particularly well for instances that can be solved in this relatively short time. Secondly, if the Weibull distribution is used and a shape parameter greater than 1.0 is predicted, the pipeline restarts strategy will not perform restarts. As a result, the no restarts and pipeline restarts strategies do not differ on these instances.

An impression of the effect of these reasons is gained by considering the hard instances for which the pipeline restarts strategy performs restarts. Figure 6.6 depicts the runtimes of the hardest 33 instances for which restarts are performed. Clearly, the majority of points are above the identity line, and therefore the pipeline strategy is more efficient on the corresponding instances.

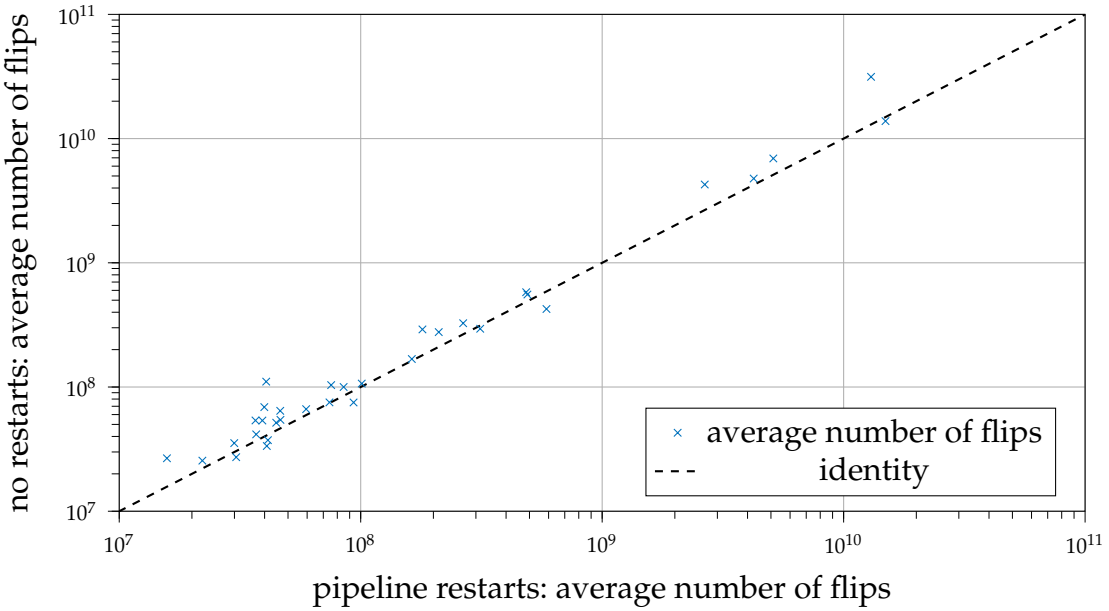


Figure 6.6: The 33 instances with the longest runs for which restarts are predicted. The average speedup on these instances is 1.216. The dots are the logarithmically scaled average runtimes. Each marker indicates the average runtimes on a single instance. The average runtimes of the pipeline restarts strategy are shown on the x-axis; those of the no restarts strategy are on the y-axis. The pipeline restarts strategy is more efficient if the marker lies above the dashed line; otherwise, its competitor is more efficient. This figure is based on [113].

Since the results are so close, a statistical analysis is warranted to clarify whether the observed speedup might be explained by coincidence. We use two different statistical tests, the t-test [79] and a modified version of the Wilcoxon signed-rank test [86]. The t-test evaluates the average speedup, while the modified version of the Wilcoxon signed-rank test assesses the median speedup.

When considering all 99 instances, an average speedup of 1.063 is achieved by the pipeline restarts strategy as compared to the no restarts strategy. This speedup is statistically significant (t-test $p = 0.0044$, Wilcoxon $p = 0.0174$) at a significance level of $\alpha = 0.05$. As mentioned above, the two strategies only differ if restarts are performed. If we restrict ourselves to the 69 instances for which restarts are performed, then the advantage of the pipeline restarts strategy increases. In this case, an average speedup of 1.108 is observed. This result is statistically significant as well (t-test $p = 5.227 \cdot 10^{-4}$, Wilcoxon $p = 9.870 \cdot 10^{-4}$).

These results suggest that the pipeline restart strategy provides the benefits of PROBSAT on easy and moderate instances. At the same time, it can reliably detect harder instances, and thus, supports PROBSAT with a restart strategy.

As a side note: As an additional comparison, we also tested PROBSAT with Luby's strategy (see Definition 5.1) on the instances in $\mathcal{K}$. However, it is not competitive with the other two strategies: The pipeline restart strategy has an average speedup of 2.725 as compared to the PROBSAT version with Luby's strategy. This result is also statistically significant (all p-values are less than 10^{-12}).

6.7 PROBSAT ON INSTANCES WITH HIDDEN SOLUTIONS

So far, we considered the performance of PROBSAT on uniform random instances. Next, we perform a similar analysis for PROBSAT on instances with a hidden solution. The SAT Competition 2018 shall serve as a motivation. At the time of writing, the 2018 SAT Competition marks the last occasion on which a random track was included. In fact, a version of PROBSAT participated in the competition. However, it finished towards the bottom of the field: With ten participating solvers, PROBSAT only ranked eighth [85]. It is worth taking a closer look at how these results came to be and especially at the underlying reasons for the poor performance of PROBSAT.

To this end, we run PROBSAT on our own system on all instances of the random track of the SAT Competition 2018 [84] subject to the conditions of the SAT competition. This means, in particular, that there is a time limit of 5000 seconds for each instance. For this purpose, we use the parameter settings as described in [15]. The parameters are $c_b = 2.06$ and $\varepsilon = 0.9$, this parameter combination is tuned specifically for SAT competitions. Besides PROBSAT, we also run SPARROW2RISS [13] and GLUHACK [184], which ranked first and second respectively at the SAT Competition 2018. All solvers are executed on a computer with 32 Intel Xeon E5-2698 v3 CPUs having a 2.30 GHz CPU clock rate. Each solver only has one attempt per instance in order to solve it within the time limit. The data for PROBSAT and SPARROW2RISS has previously been used for the findings in [116].

Overall, the instances of the SAT Competition 2018 consist of 165 hidden solution and 90 uniform instances. Table 6.5 shows the results of the three solvers analyzed by instance type.

Clearly, both SPARROW2RISS and GLUHACK perform substantially better in the aggregate than PROBSAT. It is interesting to note, however, that PROBSAT is the best of the three solvers on uniform instances. On the other hand, the question arises whether restart strategies help achieve a more competitive performance of PROBSAT

	hidden instances	uniform instances	overall
PROBSAT	78	55	133
GLUHACK	165	0	165
SPARROW2RISS	165	24	189

Table 6.5: The number of instances solved by PROBSAT, GLUHACK and SPARROW2RISS. The two middle columns indicate the result on instances with a hidden solution and uniform instances, respectively. The rightmost column represents the overall result on all instances.

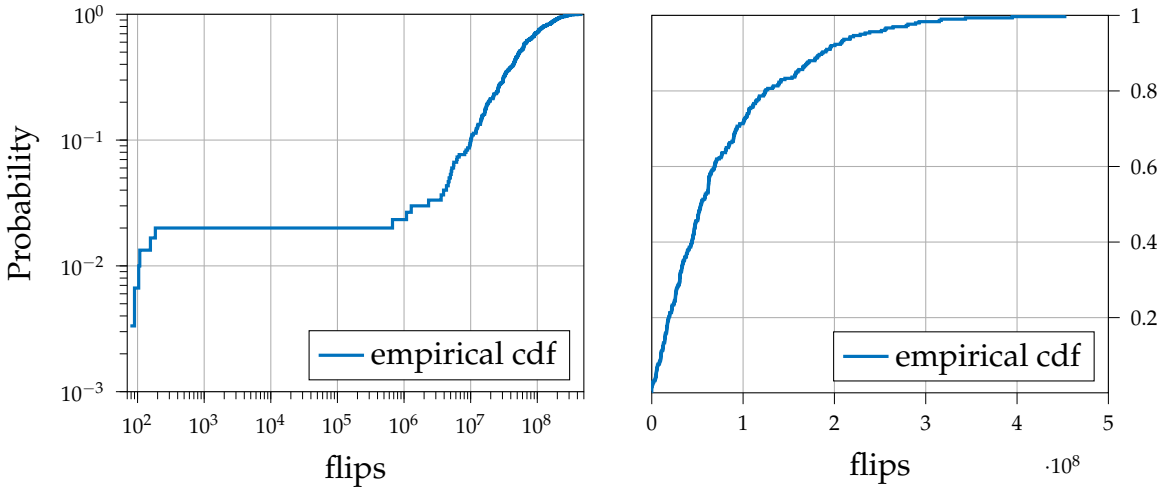
on instances with a hidden solution. This question is the subject of this section. For this purpose, we proceed similarly as before in the case of uniform instances.

6.7.1 *Runtime Distributions*

The 165 hidden solution instances from the SAT competition 2018 can be subdivided into three categories: `barthel`, `komb` and `qhid` instances (see Section 6.5). In this section, we investigate the behavior of PROBSAT on these three instance types by means of empirical runtime distributions. As seen in Table 6.5, PROBSAT has serious issues when running on instances with a hidden solution. On closer inspection, it becomes apparent that these problems primarily exist for `komb` and `qhid` instances. Most instances of these types are not solvable for PROBSAT in the size used in SAT Competition (200 - 400 variables). Therefore, we first investigate much smaller instances to study the behavior of PROBSAT. More precisely, 100 `komb` instances with 80 to 120 variables, 100 `qhid` instances with 30 to 70 variables[2], and 100 `barthel` instances with 180 to 220 variables are generated. For generating these instances, CONCEALSATGEN [115] has been used.

Similar to before, each of these instances is solved 300 times by PROBSAT, and the runtime is measured in flips. Especially on `komb` and `qhid` instances, we observed an interesting behavior, which is exemplarily illustrated in Figure 6.7. In the instance described in Figure 6.7, PROBSAT has a small, but not negligible, probability of finding a satisfying assignment within a few hundred flips. Otherwise, PROBSAT needs several million flips to find a solution. The implications for restart strategies are evident: In such a case, a restart should be performed after a few hundred flips.

2 One of the `qhid` instances having 70 variables has been replaced because only a few runs of PROBSAT have found a solution within 24 hours of runtime. In other words, this instance is too difficult for PROBSAT.



(a) log-scale **(b)** linear scale

Figure 6.7: The empirical distribution function of a typical `komb` instance with 80 variables. The left plot is logarithmically scaled, while the right plot is linearly scaled.

Then, the expected runtime is likely to improve by multiple orders of magnitude. This behavior is by no means an outlier; instead, it is frequently observed in varying degrees. One way to describe this characteristic is by using an exponential mixture distribution. This type of distribution has also been proposed in the literature as a fitting model to describe stochastic local search algorithms [87].

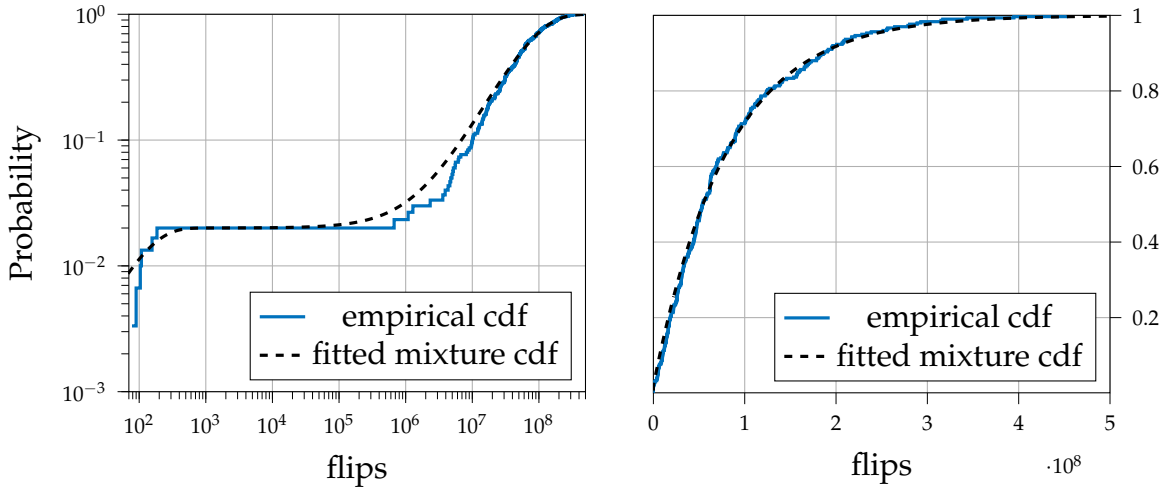
Definition 6.9 ([95])**.** A continuous random variable X with cdf

$$F(x) = q \cdot \left(1 - e^{-\lambda_1 \cdot x}\right) + (1 - q) \cdot \left(1 - e^{-\lambda_2 \cdot x}\right), \quad x \geq 0$$

has a two-component **exponential mixture distribution** with parameters $\lambda_1 \in \mathbb{R}_+$, $\lambda_2 \in \mathbb{R}_+$, and $q \in (0, 1)$. The pdf f of X is given by:

$$f(x) = q \cdot \lambda_1 \cdot e^{-\lambda_1 \cdot x} + (1 - q) \cdot \lambda_2 \cdot e^{-\lambda_2 \cdot x}, \quad x \geq 0.$$

If the special cases $q = 0$ or $q = 1$ are permitted, then this results in an exponential distribution (see Definition 2.40). For the fitting of exponential mixture distributions, we employ the Python package EXP_MIXTURE_MODEL [137]. Figure 6.8 depicts a fit of the exponential mixture distribution. The underlying instance is the same as in Figure 6.7.



(a) log-scale **(b)** linear scale

Figure 6.8: The empirical distribution function and the exponential mixture fit of a typical `komb` instance with 80 variables. This is the same instance as in Figure 6.7. The left plot is logarithmically scaled, while the right plot is linearly scaled. The fitted exponential mixture distribution has parameters $\lambda_1 = 120.167$, $\lambda_2 = 80593885.190$, and $q = 0.020$. The KS test statistic is $D_{300} = 0.0432$ and has an associated p-value of $p = 0.6132$.

Although it is by no means a perfect fit, the exponential mixture fit adequately describes the general behavior of PROBSAT on this instance. However, a potential problem is indicated at the left part of the logarithmically scaled plot in Figure 6.8. There, the fit seems to approach 0 much slower than the empirical cdf. If one would use the fit to calculate a restart time, there is a risk of using a too short restart time and thus deteriorating the performance of PROBSAT. This potential problem is also backed by theory since, according to Wolter [180], restarts are always useful for exponential mixture distributions. To hedge against the risk of a too-short restart time, we once again introduce an additional location parameter b. Unlike in the section about uniform instances, this time, b is not determined from the data; instead, we fix $b = 1.5n$, where n is the number of variables of the corresponding instance. Later on, we empirically find that this location parameter is well suited.

In the next step, we investigate how often the exponential mixture distribution appropriately describes the ecdf. As before, we compare the fitted cdf with the empirical cdf using the KS test. If the KS test is passed at a significance level of $\alpha = 0.05$, then we consider the fit to be adequate. Table 6.6 shows how often the KS test has been passed. We distinguish the data based on the corresponding instance type. Al-

	passed	total number of instances
barthel	28	100
komb	96	100
qhid	54	100

Table 6.6: The number of instances where the fitted exponential mixture distribution passed the KS test at a significance level of 0.05. The left column contains the instance type, the middle column contains the number of fits that passed the KS test and the right column indicates the total number of considered instances.

though exponential mixture distributions describe the behavior of PROBSAT on komb instances well, on the other two instance types, however, the results are mediocre; nevertheless, PROBSAT does not exhibit issues on all of these instances. Hence, considering the goal of improving the behavior of PROBSAT on the particularly challenging instances, we can confine ourselves to the instances for which PROBSAT required a lot of effort on average. Therefore, Table 6.7 only examines those instances that required at least 100 000 flips on average over 300 runs.

	passed	total number of instances
barthel	6	6
komb	85	86
qhid	47	47

Table 6.7: The number of instances where the fitted exponential mixture distribution passed the KS test at a significance level of 0.05. The left column contains the instance type, the middle column contains the number of fits that passed the KS test and the right column indicates the total number of considered instances. Here, only instances with an average runtime of more than 100 000 flips are considered.

Altogether, exponential mixture distributions are well suited to describe the behavior of PROBSAT on hard instances. Furthermore, a sufficiently long run of PROBSAT can easily isolate the easy instances. All instances that have not been solved by PROBSAT can then be assumed to be hard instances.

6.7.2 *Estimating the Potential of Restarts*

In the next step, we analyze the extent to which exponential mixture distributions are suitable to determine useful restart times. There is no known quantile function

in closed form for exponential mixture distributions to the best of our knowledge. Therefore, previously applied methods require a slight adjustment.

For this purpose, let X be an exponential mixture distributed random variable with cdf F and pdf f. We are interested in the optimal restart quantile for the random variable $Y = X + 1.5 \cdot n$. Therefore, consider the condition from Corollary 4.18 for determining an optimal restart quantile p with location parameter $b = 1.5 \cdot n$, i. e.,

$$(p-1) \cdot Q(p) + p \cdot (1-p) \cdot Q'(p) - \int_0^p Q(u)\,\mathrm{d}u = 1.5 \cdot n, \tag{6.6}$$

where Q is the quantile function of X of an exponential mixture distribution without location parameters and Q' is the derivative of Q. First, examine the integral $\int_0^p Q(u)\,\mathrm{d}u$. The following equality holds true:

$$\int_0^p Q(u)\,\mathrm{d}u = \int_0^{Q(p)} x \cdot f(x)\,\mathrm{d}x. \tag{6.7}$$

Since Q is the inverse function of the cdf F, we have $Q'(u) = \frac{1}{f(Q(u))}$. Thus, Equation (6.7) is obtained by the substitution $x = Q(u)$. If Equation (6.7) is inserted into Equation (6.6) and additionally the substitution $p = F(t)$ is applied, then the following condition is obtained:

$$\big(F(t)-1\big) \cdot t + F(t) \cdot \big(1 - F(t)\big) \cdot \frac{1}{f(t)} - \int_0^t x \cdot f(x)\,\mathrm{d}x = 1.5 \cdot n. \tag{6.8}$$

Moreover, the following expression

$$-\frac{q \cdot (\lambda_1 \cdot x + 1) \cdot e^{-\lambda_1 \cdot x}}{\lambda_1} - \frac{(1-q) \cdot (\lambda_2 \cdot x + 1)\, e^{-\lambda_2 \cdot x}}{\lambda_2}$$

is an antiderivative of $x \cdot f(x)$. Thus, all terms of Equation (6.8) are known and can be solved by numerical approaches. Solving this equation yields the optimal restart quantile $p = F(t)$ for Y. Let F_Y be the cdf of Y. As per Definition 2.39,

$$F_Y(t + 1.5 \cdot n) = F(t) = p$$

holds. Put differently, $t + 1.5 \cdot n$ is the optimal restart time for Y. Moreover, t is already known by solving Equation (6.8). This method is subsequently applied whenever the optimal restart time for an exponential mixture distribution is used.

Next, we follow the methodology from Section 6.6.4, i. e., we identify the empirically best restart times from the data and calculate the average speedup. This results in an average potential speedup of 18.20. As before, this value primarily serves as a reference value. Nevertheless, it also demonstrates that PROBSAT has great potential for improvement through the use of restarts.

Next, this reference value is compared to the average speedup resulting from the calculation of the optimal restart times on the fitted distributions. A technical remark is also noteworthy. We defined exponential mixture distributions such that the parameter q is in the open interval $(0, 1)$. However, the package we use for fitting may also assign the values 0 or 1 to q. This can then be interpreted to mean that the fitted distribution is not an exponential mixture distribution but rather an exponential distribution. We refer to instances whose fitted runtime distribution corresponds to a true exponential mixture as **two-component instances**. In other words, these are instances having a fitted runtime distribution with $q \in (0, 1)$. For this kind of instance, the optimal restart time can be calculated as described above.

All other instances have an exponential distribution as fitted runtime distribution. For exponential distributions, it is known that restarts have neither a positive nor a negative effect on the expected value. This is why we do not perform restarts in such a case. Accordingly, each of these instances yields a speedup of 1.0. By distinguishing these two cases, the average speedup over all instances can be calculated. The resulting average speedup is 13.78. It should also be mentioned that it is possible that an optimal restart value is obtained that is less than the smallest observed value. In such a scenario, the speedup is undefined. However, this has only happened five times for the 300 instances. These five values are not included in the speedup calculation. For the following, it is also convenient to calculate the average speedups on different subcategories as reference values. These are shown in Table 6.8.

It is especially remarkable that the restart strategies on `komb` and `qhid` instances exhibit a high speedup value. Since these are the two domains where PROBSAT encountered the most difficulties, this raises expectations that this insight may be utilized to enhance the performance. Overall, exponential mixture distributions are a valuable model for determining a restart strategy.

	empirical	fitted
all instances	18.20	13.78
`komb` and `qhid` instances	74.05	52.31
two-component instances	228.97	157.36

Table 6.8: This table shows the average speedups by restarting on different domains. In the middle column, the optimal restart times are estimated based on the collected data. In the right column, the restart times are determined by the fitted exponential mixture distribution. The second row contains all instances, while the third row is limited to instances of the types `komb` and `qhid`. In the bottom row, only two-component instances are considered.

6.7.3 *Predicting the Runtime Distribution*

The goal of this section is once again to predict the runtime distribution of previously unseen instances and to derive a restart strategy. The eventual objective is to obtain an improved performance on the instances of the SAT Competition 2018 using this approach. In particular, the goal is to predict the runtime distribution for hard instances. We, therefore, limit the training set to two-component instances. This leaves a total of 158 instances. The quality is later evaluated using a separately generated test set.

Initially, the same procedure for generating features is used as described in Section 6.6.6. In contrast to Section 6.6.5, it is not necessary to employ an entire pipeline. The reason for this is that in Section 6.6.5, it was required to distinguish whether a lognormal or a Weibull distribution is the more appropriate model. In contrast, in this section, we have restricted ourselves to exponential mixture distributions; thus, the distribution type does not have to be determined.

The only remaining problem is estimating the parameters of the exponential mixture distribution. To predict these parameters, we use random forests for regression problems[3]. The parameters of the fitted exponential mixture distributions serve as labels. First, however, these parameters are rescaled analogously to Equation (6.4). This means that for each parameter, the minimum value is 0 and the maximum value is 1. As a result, the three parameters have essentially the same scale. We did not rescale the features here as this did not positively affect the quality of the random forest.

3 Note the difference with Section 6.6.7, where a random forest has been used for a classification problem, whereas here, the random forest is used for a regression problem.

Similar to the analysis of uniform instances (cf. Section 6.6.6), a short period of hyperparameter tuning revealed that only features with a minimum degree of variance should be utilized. Here, we restricted ourselves to features with a minimum sample variance of 0.08. This leaves a total of 16 features that are listed in Appendix C.2. In addition, the random forest consists of five decision trees, and only 10% of the features are available for each split.

Five iterations of five-fold cross-validation assess the quality. Again, the average speedup is used as a measure. Based on the predictions of Random Forest, an average speedup of 106.58 is achieved. As described above, it is possible that the predicted restart time is so short that it is less than the smallest observation. Again, such cases are not included in the calculated speedup. This case occurred on average 1.8 times per 31 or 32 instances in the cross-validation phase.

6.7.4 *Evaluation on the Test Set*

Up to now, the quality of the random forest has been assessed on the basis of the previously collected data. In order to judge the quality more accurately, a test set with a total of 200 new instances is used. The test set consists of 100 `komb` and 100 `qhid` instances, i. e., the type of instances on which PROBSAT struggles the most. The `komb` instances have 100 variables, and the `qhid` instances have 50 variables.

In the following, "random forest restarts" refers to the restart strategy based on the random forest predictions. The original version of PROBSAT is called "no restarts strategy". Each of the two strategies is executed 300 times per instance, and the average runtime (in flips) is compared. Figure 6.9 illustrates the comparison of the two strategies on the test set.

The runtimes on easy instances are clearly not or only marginally benefiting from restarts. However, the vast majority of runtimes on hard instances significantly profit from the use of the restart strategies determined by the random forest. An average speedup of 50.95 also reflects the quality of this strategy. This is, evidently, a substantial improvement over the original version of PROBSAT. As can be observed in Figure 6.9, the performance improves by several orders of magnitude on many instances.

It should be noted that this speedup value is not directly comparable to the speedup from the cross-validation. Only two-component instances are used for the training, and therefore the speedup computed during the cross-validation only refers

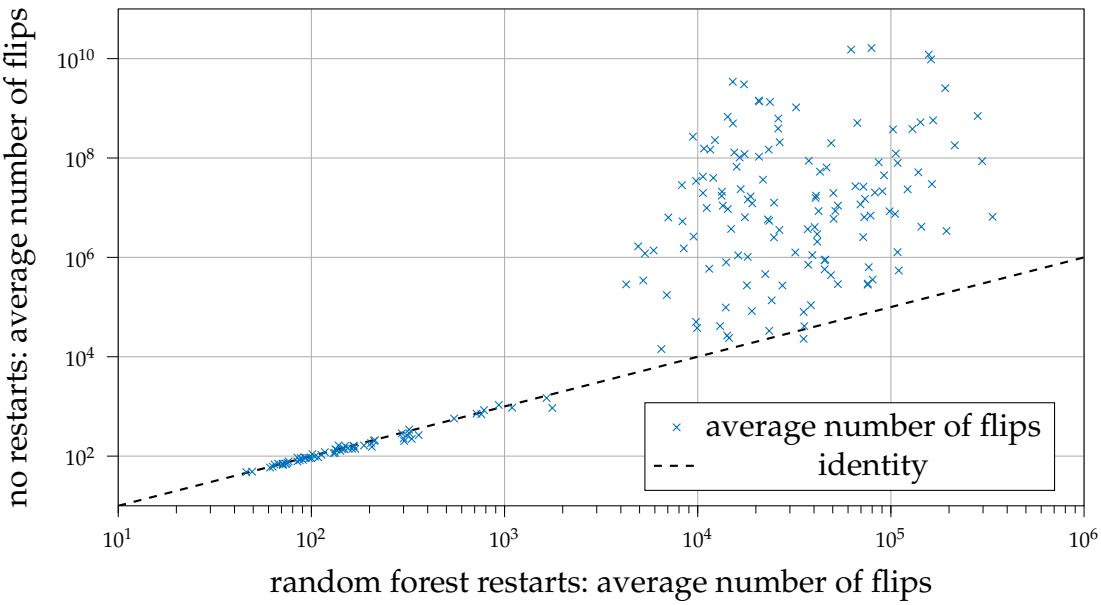


Figure 6.9: The random forest restarts strategy is compared with the no restarts strategy. The dots are the logarithmically scaled average runtimes. Each marker indicates the average runtimes on a single instance. The average runtimes of the random forest restarts strategy are shown on the x-axis; those of the no restarts strategy are on the y-axis. The random forest restarts strategy is more efficient if the marker lies above the dashed line; otherwise, its competitor is more efficient.

to such two-component instances. No such preselection has been carried out for the test set used here.

6.7.5 *Evaluation on the Instances of the SAT Competition 2018*

The restart strategies derived from the random forest are highly effective for instances similar in size to the training set. However, the original motivation is to check whether this approach can be transferred to the instances of the SAT Competition 2018. For this purpose, the following approach is employed. For certain instances, the random forest trained in Section 6.7.3 is used to predict the parameters of the exponential mixture distribution. The detailed approach is illustrated in Figure 6.10.

The rationale behind this procedure is to first identify the instances having a hidden solution. This is accomplished in two steps. First instances are filtered by the number of variables, then by the clause-to-variable ratio.

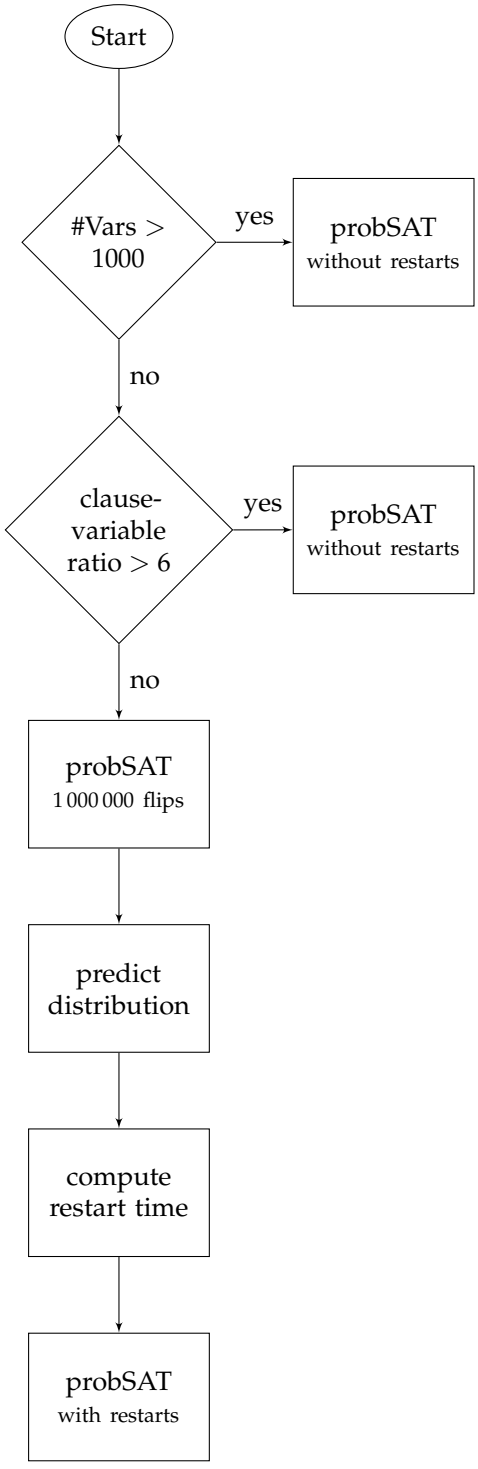


Figure 6.10: Flowchart description of MLPPROB.

The comparison to uniformly generated 3-SAT instances is especially meaningful. In the previous years of the SAT competition, uniformly generated 3-SAT instances always had several thousand variables, whereas instances with a hidden solution only had a few hundred variables. We, therefore, distinguish the two cases at 1000 variables: If the instance has more than 1000 variables, PROBSAT is executed without restarts.

The remaining instances are uniformly generated K-SAT instances with $k > 3$ and instances with a hidden solution. Since the uniform instances in the competition have a clause-to-variable ratio close to the satisfiability threshold, the uniform instances can be separated from the instances with a hidden solution based on this ratio. For K-SAT instances with $k \geq 4$, the clause-to-variable ratio is well above 6, so we choose this as a boundary: If the instance has a clause-to-variable ratio above 6, PROBSAT is performed without restarts.

We assume that all remaining instances have a hidden solution. As can be observed on the test set in Figure 6.9, the restart strategies do not positively affect particularly simple instances. Since the overhead for extracting the features must also be factored in, the overall impact on easy instances is likely to be negative. Therefore, we first attempt to filter out simple instances by running PROBSAT for one million flips. On our system, this corresponds to a runtime of about one second. For all remaining instances, the parameters of an exponential mixture distribution are estimated using the random forest. Based on these parameters, a restart time is determined, which is then utilized by PROBSAT. In the following, we refer to this strategy as MLPPROB.

MLPPROB is executed, as described at the beginning of this section, under the conditions and on the instances of the SAT Competition 2018. To review: There is only one attempt per instance and the time limit is 5000 seconds. As before, the experiments are conducted on a computer with 32 Intel Xeon E5-2698 v3 CPUs having a 2.30 GHz CPU clock rate. It should be emphasized that, in contrast to the rest of this chapter, the runtime is measured in seconds. This way, the time for the feature extraction and the restart time calculation is also factored in.

For the evaluation of MLPPROB and the comparison with other solvers, the par2 score is primarily used. The par2 value with respect to an instance is the time in seconds until a satisfying assignment is found or twice the timeout (i. e., 10000) if no solution is found. The par2 score is the sum of all par2 values over all instances. In other words, a lower par2 score corresponds to better performance.

As can be inferred from Table 6.9, the introduction of a restart strategy based on exponential mixture distributions yields a significant improvement over the original

	solved instances	par2 score
PROBSAT	133	1 234 986.01
GLUHACK	165	901 078.94
SPARROW2RISS	189	672 335.89
MLPPROB	188	707 444.40

Table 6.9: The results of PROBSAT, GLUHACK, SPARROW2RISS and MLPPROB on the instances of the SAT Competition 2018. The middle column contains the number of solved instances, while the right column reflects the par2 score.

version of PROBSAT. The enhancement is so big that MLPPROB came in second with a clear lead over GLUHACK. Additionally, although the improvement is not sufficient to rank first, the results also indicate that MLPPROB is at least competitive with SPARROW2RISS.

We also performed statistical analysis of the par2 score results using the t-test [79] and the Wilcoxon signed rank test [178]. We use a significance level of $\alpha = 0.05$ in each case. First, the improvements of MLPPROB over the original version of PROBSAT are statistically significant according to both the t-test ($p < 10^{-13}$) and the Wilcoxon test ($p \approx 0.001$). In comparison with GLUHACK, the improvements are statistically significant according to the t-test ($p \approx 0.04$), while the Wilcoxon test does not detect statistical significance ($p \approx 0.821$). Finally, SPARROW2RISS is not statistically significantly better than MLPPROB, according to both the t-test ($p \approx 0.664$) and the Wilcoxon test ($p \approx 0.172$). This is another indicator of the competitiveness of MLPPROB with SPARROW2RISS.

To gain a more precise understanding of how the restart strategy has affected the performance, it is also worth examining the three different instance types `barthel`, `komb` and `qhid`. This comparison is illustrated in Table 6.10. There the number of solved instances is decomposed by instance type.

	`barthel`	`komb`	`qhid`
PROBSAT	55	11	12
GLUHACK	55	55	55
SPARROW2RISS	55	55	55
MLPPROB	55	55	20

Table 6.10: The number of instances solved by PROBSAT, GLUHACK, SPARROW2RISS, and MLPPROB. The columns indicate the results on the three instance domains with a hidden solution.

It becomes apparent that the majority of the improvement of MLPPROB compared to the original version of PROBSAT is achieved on `komb` instances. With the restart strategy determined by the random forest, all instances of this domain are solved. The behavior on `qhid` instances has improved as well, but there is still much remaining potential as can be observed.

Further improvements could be achieved by applying a preprocessor to simplify the instances before solving them. This is, for example, a technique used by SPARROW2RISS. Secondly, determining a restart strategy using a random forest is also compatible with the approach presented in [116]. There, additional clauses are added to the instance. In [116], it is shown that this significantly improves the performance of PROBSAT. Finally, the combination of restart strategies with different initialization strategies, in particular, may synergize. However, this is not further investigated in this thesis.

6.8 CONCLUSION AND OUTLOOK

In this chapter, the runtime behavior of a SAT solver, PROBSAT, is extensively investigated on different domains. The objective is to answer whether restarts are generally useful for PROBSAT, whether typical probability distributions are useful for determining restart times and whether such probability distributions can be predicted using machine learning techniques. As the last sections have shown, all three questions can be answered in the affirmative.

On uniformly generated instances, a speedup of up to 1.393 could be achieved by restarts. This is noteworthy because the used parameters have been tuned on uniform instances. Furthermore, the Weibull and the lognormal distribution are especially well suited to model the runtime behavior. In combination, a very good restart time can often be inferred from the fitted distributions. Finally, we proposed a machine learning pipeline for predicting runtime distributions that can be used to evaluate restart properties. The evaluation found that this strategy results in an average speedup factor of 1.06. The runtimes of particularly hard instances even have a speedup factor of 1.21.

The effect of restarts is even greater on instances having a hidden solution. On two instance types in particular, `komb` and `qhid` instances, PROBSAT struggles to find a solution without restarting. Examining PROBSAT's behavior on small instances of these two domains suggests that a potential speedup of 74.05 would be achievable

through restarts[4]. Furthermore, exponential mixture distributions can adequately model the runtime behavior on difficult instances. These distributions enable the calculation of restart times, which significantly improve the performance of PROBSAT. Furthermore, it is also possible to predict the parameters of these distributions by machine learning techniques. We opt for a random forest for regression problems. The resulting random forest is evaluated on a test set. The associated restart strategies lead to an average speedup of 50.95.

Finally, this procedure is tested on the instances of the SAT Competition 2018. Without restarts, PROBSAT ranked eighth out of ten participants. The modified version with restarts puts PROBSAT in second place. Consequently, there is considerable potential for improvement based on restarts. This approach can possibly be further optimized by considering different initialization strategies.

In this chapter, the methodology of Chapter 4 is used for each decision concerning whether restarts are useful and each calculation of a restart time. This shows that these methods, together with machine learning techniques for determining runtime distributions, can be relevant in practice. Naturally, the general approach of predicting the runtime distribution by means of a machine learning pipeline can also be applied to other problems and algorithms.

In addition, the approach presented here can, of course, be combined with other ideas. For example, there is a synergy effect between restarts and diverse initialization strategies. For example, the clauses in the `qhid` and `komb` domains are biased, i. e., the clauses mostly contain two literals not satisfied by the hidden solution. This knowledge could be exploited for a specifically crafted initialization strategy. The frequent restarts caused by the machine learning approach would then ensure that PROBSAT eventually starts with an assignment close to the hidden solution. It would certainly be interesting to investigate this approach in more detail; in any case, it is conceivable that this approach could lead to PROBSAT achieving a better performance than SPARROW2RISS.

6.9 CODE AND DATA AVAILABILITY

The source code realizing the methodologies described here, as well as the corresponding results, are freely available. The code and data for the uniform instances

4 Of course, it is conceivable that the speedup is different on larger instances. However, since PROBSAT can hardly solve larger instances of these types, it is difficult to estimate the restarts' effect.

(see Section 6.6) are available in [114]. The code and data for the instances with a hidden solution (see Section 6.7) are available in [111]. The evaluation of PROBSAT and SPARROW2RISS on the SAT Competition 2018 instances is also included in [111]. This data has been previously reported in [117].

7

COMPLETION PROBABILITIES AND PARALLEL RESTART STRATEGIES UNDER AN IMPOSED DEADLINE

Parts of this chapter appeared in the following publication:

Jan-Hendrik Lorenz.
Completion Probabilities and Parallel Restart Strategies under an Imposed Deadline.
In: *PLOS ONE* 11.10 (2016), pp. 1–15.

Up to this point, we primarily focused on the objective of optimizing an expected value by restarts. It is implicitly assumed that the associated algorithm is sequential. In this chapter, we deviate from this goal. In particular, we consider processes that are subject to a deadline, i. e., a time frame in which the process has to be completed. In formal terms, deadlines are defined as follows.

Definition 7.1 ([69])**.** Let $\mathcal{T}$ be some task, and let $D \in \mathbb{R}_+$ be any positive real number. If $\mathcal{T}$ has to be completed within D time units, then we call D a **deadline** for $\mathcal{T}$. If $\mathcal{T}$ is completed within the deadline D, then we say $\mathcal{T}$ met the deadline. Otherwise, $\mathcal{T}$ failed.

We also consider a particular form of parallelization. This is achieved by executing multiple independent copies of a randomized algorithm.

Definition 7.2 ([122])**.** Assume $\mathcal{A}$ is a (randomized) algorithm. A **parallelized algorithm** $\mathcal{A}^{(n)}$ is derived by running $\mathcal{A}$ independently on $n \in \mathbb{N}$ processors. The parallel algorithm $\mathcal{A}^{(n)}$ stops as soon as the first of the n copies of $\mathcal{A}$ stops.

When referring to parallel algorithms, we implicitly mean this kind of parallel algorithm. Since the copies have to be independent of each other, this also implies that they cannot communicate with each other or exchange information in any other way. Procedures employing such a form of parallelization necessitate a separate analysis. Naturally, restart strategies can also be applied to the type of parallel algorithms considered here.

Definition 7.3 ([122]). Let $\mathcal{A}^{(n)}$ be a parallel algorithm as specified in Definition 7.2. A parallel algorithm $\mathcal{A}_t^{(n)}$ employing a **parallel fixed-cutoff strategy** with restart time $t \in \mathbb{R}_+$ is obtained if each of the n copies of $\mathcal{A}$ uses the fixed-cutoff strategy $(t, t, \dots)$.

In this chapter, we focus on the extent to which restart strategies can be applied within the context of parallel algorithms to increase the probability of meeting a deadline. An important result is implicit in a work by Luby and Ertel [122]. As we frequently apply the following lemma, we also provide a proof.

Lemma 7.4 ([122]). *Let X be a random variable describing the runtime behavior of an algorithm $\mathcal{A}$ and let F be the cdf of X. Also, assume $X_t^{(n)}$ to be a random variable describing the parallelized algorithm's behavior $\mathcal{A}_t^{(n)}$ running on $n \in \mathbb{N}$ processors and using the parallel fixed-cutoff strategy $(t, t, \dots)$. In addition, there is a deadline $D \in \mathbb{R}_+$ for $\mathcal{A}_t^{(n)}$. The probability of $\mathcal{A}_t^{(n)}$ meeting the deadline is given by:*

$$\Pr\left(X_t^{(n)} \leq D\right) = 1 - \left(1 - F(t)\right)^{k \cdot n} \cdot \left(1 - F(\ell)\right)^n.$$

Here, $k = \lfloor \frac{D}{t} \rfloor$ is the number of restarts within the deadline, and $\ell = D - k \cdot t$ is the leftover time for the last run before the deadline.

Proof. Let X_t be a random variable describing the runtime behavior of $\mathcal{A}_t$; this is a version of $\mathcal{A}$ without parallelization employing the restart strategy $(t, t, \dots)$. The probability of $\mathcal{A}_t$ being unable to meet the deadline is:

$$\Pr\left(X_t > D\right) = \left(1 - F(\ell)\right) \cdot \left(1 - F(t)\right)^k.$$

For the reasoning refer, to Equation (5.1). As $\mathcal{A}_t^{(n)}$ consists of n independent copies of $\mathcal{A}_t$ and stops as soon as a copy stops, this immediately implies

$$\Pr\left(X_t^{(n)} > D\right) = \Pr\left(X_t > D\right)^n = \left(1 - F(\ell)\right)^n \cdot \left(1 - F(t)\right)^{n \cdot k}.$$

The desired property immediately follows from this equation. □

This lemma also includes the case $n = 1$, i. e., the situation in which the algorithm only runs on one processor, as a special case.

It is noteworthy that when introducing a deadline or parallelism, some surprising consequences regarding the restart characteristics occur. For example, it is known [122] that, in general, the fixed-cutoff strategy is no longer an optimal

restart strategy with respect to the expected value when parallelization is used. However, the expected value of the optimal strategy and the best fixed-cutoff strategy's expected value differ by a constant factor at most, according to [122]. Even if one confines oneself to fixed-cutoff strategies, there are still some interesting findings. For instance, the sequential case's best fixed-cutoff strategy is not necessarily the best fixed-cutoff strategy in the parallel case [159].

Introducing a deadline may also change the behavior regarding restarts. Wu [181] explored the effect of restart strategies on the probability of meeting the deadline (without parallelization). He discovered that the fixed-cutoff strategy is no longer optimal in this context.

i	1	2	3	100
$\Pr(X = i)$	$\frac{1}{10}$	0	$\frac{1}{3}$	$\frac{17}{30}$
$\Pr(X \leq i)$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{13}{30}$	1

Table 7.1: A description of a discrete random variable X based on the pmf and the cdf. All values i not explicitly specified in the table have an associated value of $\Pr(X = i) = 0$.

A simple example can demonstrate this fact. For this purpose, consider a discrete random variable X as indicated in Table 7.1. Suppose the deadline is 8, then there are only three reasonable fixed-cutoff strategies: Either no restarts are performed, or the restart times are either 1 or 3. The respective probabilities of meeting the deadline are calculated by means of Lemma 7.4 with $n = 1$:

$$\Pr(X \leq 8) = F(8) = \frac{13}{30} \approx 0.43,$$

$$\Pr(X_1 \leq 8) = 1 - \left(\frac{9}{10}\right)^8 \approx 0.57,$$

$$\Pr(X_3 \leq 8) = 1 - \left(1 - \frac{13}{30}\right)^2 \cdot \left(\frac{9}{10}\right) = 0.711.$$

Let $L = (3, 3, 1, 1)$ be a restart strategy, and let X_L be the associated random variable. The probability of meeting the deadline is then given as follows:

$$\Pr(X_L \leq 8) = 1 - \left(1 - \frac{13}{30}\right)^2 \cdot \left(\frac{9}{10}\right)^2 = 0.7399.$$

It is evident that L dominates all fixed-cutoff strategies regarding the probability of meeting the deadline. Thus, fixed-cutoff strategies are generally not optimal with respect to maximizing this probability.

We conclude that the introduction of a deadline, as well as parallelization, changes many properties. In this chapter, we, therefore, investigate the influence of parallel restart strategies on the probability of meeting the deadline. Although, as mentioned above, the fixed-cutoff strategy is not optimal in this setting, we continue to use this strategy. One of the main reasons is that neither for deadlines nor parallelization, a generally optimal strategy is known. For this reason, we also focus on fixed-cutoff strategies in this chapter.

The rest of this chapter is structured as follows. Section 7.1 examines how the combination of parallelization and restart strategies affects the probability of meeting a deadline. Section 7.2 explores which fixed-cutoff strategies are optimal. Finally, Section 7.3 investigates the scaling behavior of the investigated restart strategies in terms of the number of processors.

7.1 EFFECT OF PARALLEL RESTARTS ON THE COMPLETION PROBABILITY

Here we investigate how the probability of meeting the deadline changes with respect to the number of processors. Two different scenarios are considered. The first model always uses the same restart time (i. e., it is independent of the number of processors). The second version uses monotonically increasing restart times in the number of processors. Proposition 7.9 describes a scenario in which such increasing restart times occur naturally.

We start with the scenario where the restart times are identical. If D/t is a natural number where $t \in \mathbb{R}_+$ is the restart time and $D \in \mathbb{R}_+$ is the deadline, then a first result immediately follows from Lemma 7.4.

Corollary 7.5. *Suppose X is a random variable having cdf F such that X describes the runtime behavior of an algorithm $\mathcal{A}$. Let X_t be a random variable describing the runtime behavior of the algorithm $\mathcal{A}_t$ using the fixed-cutoff strategy $(t, t, \dots)$. Also, assume $X_t^{(n)}$ to be a random variable describing the behavior of the parallelized algorithm $\mathcal{A}_t^{(n)}$ running on $n \in \mathbb{N}$ processors and using the parallel fixed-cutoff strategy $(t, t, \dots)$. In addition, there is a deadline $D \in \mathbb{R}_+$ for $\mathcal{A}_t^{(n)}$. Finally, let $k = \lfloor \frac{D}{t} \rfloor$ be the number of restarts within the*

deadline and let $\ell = D - k \cdot t$ be the leftover time for the last run before the deadline. If $\ell = 0$ and $F(0) = 0$, then the following equation holds:

$$\Pr\left(X_t^{(n)} \leq D\right) = \Pr(X_t \leq n \cdot D). \tag{7.1}$$

Proof. If $\ell = 0$, this implies that $k = \frac{D}{t}$ is a natural number. In this case, $n \cdot k = \frac{n \cdot D}{t}$ is also a natural number. Thus, the two probabilities from Equation (7.1) can be calculated by using Lemma 7.4:

$$\Pr\left(X_t \leq n \cdot D\right) = 1 - \left(1 - F(t)\right)^{k \cdot n} \cdot \left(1 - F(0)\right) = 1 - \left(1 - F(t)\right)^{k \cdot n},$$
$$\Pr\left(X_t^{(n)} \leq D\right) = 1 - \left(1 - F(t)\right)^{k \cdot n} \cdot \left(1 - F(0)\right)^{n} = 1 - \left(1 - F(t)\right)^{k \cdot n}.$$

This holds because of the assumption $F(0) = 0$. As the two probabilities are equal, the statement is proven. □

In other words, a single process needs deadlines that are n times longer to achieve the same probability in comparison to a process running in parallel on n processors. Next, we examine how the probabilities behave if the deadlines are identical in the parallel and the sequential cases.

Theorem 7.6. *Let X be a random variable describing the runtime behavior of an algorithm $\mathcal{A}$ and let F be the cdf of X. The random variable X_t describes the algorithm $\mathcal{A}_t$ using the restart strategy $(t, t, \dots)$. Also, assume $X_t^{(n)}$ to be a random variable describing the behavior of the parallelized algorithm $\mathcal{A}_t^{(n)}$ running on $n = 2^i$, $i \in \mathbb{N}$, processors and using the parallel fixed-cutoff strategy $(t, t, ...)$. In addition, there is a deadline $D \in \mathbb{R}_+$ for $\mathcal{A}_t^{(n)}$. The probability of $\mathcal{A}_t^{(n)}$ meeting the deadline is given by:*

$$\Pr\left(X_t^{(n)} \leq D\right) = \Pr(X_t \leq D) \cdot \prod_{0 \leq j < i} \left(1 + \left(1 - F(t)\right)^{2^j \cdot k} \cdot \left(1 - F(\ell)\right)^{2^j}\right).$$

Here, $k = \lfloor \frac{D}{t} \rfloor$ is the number of restarts within the deadline, and $\ell = D - k \cdot t$ is the leftover time for the last run before the deadline.

Proof. The desired property is proven by an inductive argument. First, we consider the case $n = 2$ and analyze the quotient $\frac{\Pr(X_t \leq D)}{\Pr\left(X_t^{(2)} \leq D\right)}$ by applying Lemma 7.4:

$$\begin{aligned}\frac{\Pr(X_t \leq D)}{\Pr\left(X_t^{(2)} \leq D\right)} &= \frac{1 - \left(1 - F(\ell)\right) \cdot \left(1 - F(t)\right)^k}{1 - \left(1 - F(\ell)\right)^2 \cdot \left(1 - F(t)\right)^{2 \cdot k}} \\ &= \frac{1}{1 + \left(1 - F(\ell)\right) \cdot \left(1 - F(t)\right)^k}.\end{aligned}$$

We proceed by considering the case $n = 2^i$ for $i > 1$. For this purpose, Lemma 7.4 is applied once again:

$$\begin{aligned}\frac{\Pr(X_t^{\left(\frac{n}{2}\right)} \leq D)}{\Pr(X_t^{(n)} \leq D)} &= \frac{1 - \left(1 - F(t)\right)^{k \cdot 2^{i-1}} \cdot \left(1 - F(\ell)\right)^{2^{i-1}}}{1 - \left(1 - F(t)\right)^{k \cdot 2^i} \cdot \left(1 - F(\ell)\right)^{2^i}} \\ &= \frac{1 - \left(1 - F(t)\right)^{k \cdot 2^{i-1}} \cdot \left(1 - F(\ell)\right)^{2^{i-1}}}{\left[1 - \left(1 - F(t)\right)^{k \cdot 2^{i-1}} \cdot \left(1 - F(\ell)\right)^{2^{i-1}}\right] \cdot \left[1 + \left(1 - F(t)\right)^{k \cdot 2^{i-1}} \cdot \left(1 - F(\ell)\right)^{2^{i-1}}\right]} \\ &= \frac{1}{1 + \left(1 - F(t)\right)^{k \cdot 2^{i-1}} \cdot \left(1 - F(\ell)\right)^{2^{i-1}}}.\end{aligned}$$

The statement follows by simple arithmetical transformations and recursive application of the preceding equation. □

This theorem is used later when the scaling behavior is examined more closely. We now proceed to the second scenario in which the restart times increase monotonically with respect to the number of processors. Later, we describe a situation under which this scenario occurs naturally.

Theorem 7.7. *Let $D \in \mathbb{R}_+$ be a deadline and let X be a random variable, which describes the runtime behavior of an algorithm $\mathcal{A}$, together with its cdf F. Furthermore, let $(t_0, t_1, t_2, \dots)$ be a sequence of restart times together with the associated number of restarts $k_i = \left\lfloor \frac{D}{t_i} \right\rfloor$. We require that $\forall i \in \mathbb{N}_0 : t_{i+1} \geq t_i$ and $\forall i \in \mathbb{N}_0 : k_i \geq 2$ is valid.*

For all $i \in \mathbb{N}_0$, define $X_{t_i}^{(n_i)}$ as a random variable describing the runtime of the parallel algorithm $\mathcal{A}_{t_i}^{(n_i)}$ running on $n_i = 2^i \cdot {k_0}^i$ processors using the fixed-cutoff strategy $(t_i, t_i, \dots)$. Define $m = i + i \cdot \lceil \log_2 k_0 \rceil$. Then

$$\Pr\left(X_{t_i}^{(n_i)} \leq D\right) \geq \Pr\left(X_{t_0}^{(n_0)} \leq D\right) \cdot \prod_{1 \leq j \leq i} \left[1 + \left(1 - F(t_j)\right)^{(2 \cdot k_0)^j}\right] \tag{7.2}$$

and

$$\Pr\left(X_{t_i}^{(n_i)} \leq D\right) \leq \Pr\left(X_{t_i} \leq D\right) \cdot \prod_{0 \leq j < m} \left[1 + \left(1 - F(t_i)\right)^{k_i \cdot 2^j} \cdot \left(1 - F(\ell_i)\right)^{2^j}\right] \tag{7.3}$$

holds, where $\ell_i = D - k_i \cdot t_i$ is the leftover time and X_{t_i} is a random variable describing the runtime behavior of $\mathcal{A}_{t_i}$ using the fixed-cutoff strategy $(t_i, t_i, \dots)$.

Proof. We start by showing a lower bound as in Inequality (7.2).

$$\frac{\Pr\left(X_{t_0}^{(n_0)} \leq D\right)}{\Pr\left(X_{t_1}^{(n_1)} \leq D\right)} = \frac{1 - \left(1 - F(t_0)\right)^{k_0} \cdot \left(1 - F(\ell_0)\right)}{1 - \left(1 - F(t_1)\right)^{2 \cdot k_0 \cdot k_1} \cdot \left(1 - F(\ell_1)\right)^{2 \cdot k_0}}.$$

As per the definition of the leftover time, we have $\ell_0 \leq t_0$. The monotonicity of the cdf yields $F(\ell_0) \leq F(t_0)$. By using this bound, we obtain:

$$\begin{aligned}
\frac{\Pr\left(X_{t_0}^{(n_0)} \leq D\right)}{\Pr\left(X_{t_1}^{(n_1)} \leq D\right)} &\leq \frac{1 - \left(1 - F(t_0)\right)^{k_0+1}}{1 - \left(1 - F(t_1)\right)^{2 \cdot k_0 \cdot k_1} \cdot \left(1 - F(\ell_1)\right)^{2k_0}} \leq \frac{1 - \left(1 - F(t_0)\right)^{k_0+1}}{1 - \left(1 - F(t_1)\right)^{2 \cdot k_0 \cdot k_1}} \\
&\overset{t_0 \leq t_1}{\leq} \frac{1 - \left(1 - F(t_0)\right)^{k_0+1}}{1 - \left(1 - F(t_0)\right)^{2 \cdot k_0 \cdot k_1}} \overset{k_1 \geq 2}{\leq} \frac{1 - \left(1 - F(t_0)\right)^{2 \cdot k_0}}{1 - \left(1 - F(t_0)\right)^{4 \cdot k_0}} \\
&= \frac{1}{1 + \left(1 - F(t_0)\right)^{2 \cdot k_0}}.
\end{aligned}$$

Therefore, $\Pr\left(X_{t_0}^{(n_0)} \leq D\right) \cdot \left[1 + \left(1 - F(t_1)\right)^{2k_0}\right] \leq \Pr\left(X_{t_1}^{(n_1)} \leq D\right)$ holds. In the next step, we investigate $\Pr\left(X_{t_i}^{(n_i)} \leq D\right) / \Pr\left(X_{t_{i+1}}^{(n_{i+1})} \leq D\right)$. First, the numerator is analyzed in more detail:

$$\begin{aligned}\Pr\left(X_{t_i}^{(n_i)} \leq D\right) &= 1 - \left(1 - F(t_i)\right)^{k_i \cdot (2 \cdot k_0)^i} \cdot \left(1 - F(\ell_i)\right)^{(2 \cdot k_0)^i} \\ &\leq 1 - \left(1 - F(t_i)\right)^{k_i \cdot (2 \cdot k_0)^i} \cdot \left(1 - F(t_i)\right)^{(2 \cdot k_0)^i} \\ &= 1 - \left(1 - F(t_i)\right)^{(k_i+1) \cdot (2 \cdot k_0)^i}.\end{aligned}$$

In the first step, the monotonicity of the cdf together with $\ell_i \leq t_i$ is applied once more. Furthermore, as the restart times are non-decreasing, the number of restarts is non-increasing, i. e., $k_0 \geq k_i$. Applying this fact yields:

$$\Pr\left(X_{t_i}^{(n_i)} \leq D\right) \leq 1 - \left(1 - F(t_i)\right)^{(k_0+1) \cdot (2 \cdot k_0)^i} \overset{k_0 \geq 2}{\leq} 1 - \left(1 - F(t_i)\right)^{(2 \cdot k_0)^{i+1}}. \quad (7.4)$$

Next, we consider the denominator $\Pr\left(X_{t_{i+1}}^{(n_{i+1})} \leq D\right)$:

$$\begin{aligned}\Pr\left(X_{t_{i+1}}^{(n_{i+1})} \leq D\right) &= 1 - \left(1 - F(t_{i+1})\right)^{k_{i+1} \cdot (2 \cdot k_0)^{i+1}} \cdot \left(1 - F(\ell_{i+1})\right)^{(2 \cdot k_0)^{i+1}} \\ &\geq 1 - \left(1 - F(t_{i+1})\right)^{k_{i+1} \cdot (2 \cdot k_0)^{i+1}} \\ &\overset{k_{i+1} \geq 2}{\geq} 1 - \left(1 - F(t_{i+1})\right)^{2^{i+2} \cdot k_0^{i+1}} \\ &\geq 1 - \left(1 - F(t_i)\right)^{2^{i+2} \cdot k_0^{i+1}}. \quad (7.5)\end{aligned}$$

In the last step $t_{i+1} \geq t_i$ and the monotonicity of the cdf is utilized. We are now ready to analyze $\Pr\left(X_{t_i}^{(n_i)} \leq D\right) / \Pr\left(X_{t_{i+1}}^{(n_{i+1})} \leq D\right)$ in more detail by applying Inequality (7.4) and Inequality (7.5):

$$\frac{\Pr\left(X_{t_i}^{(n_i)} \leq D\right)}{\Pr\left(X_{t_{i+1}}^{(n_{i+1})} \leq D\right)} \leq \frac{1 - \left(1 - F(t_i)\right)^{(2 \cdot k_0)^{i+1}}}{1 - \left(1 - F(t_i)\right)^{2^{i+2} \cdot k_0^{i+1}}} = \frac{1}{1 + \left(1 - F(t_i)\right)^{(2 \cdot k_0)^{i+1}}}.$$

The following conclusion can be reached through an inductive argument:

$$\Pr\left(X_{t_i}^{(n_i)} \leq D\right) \geq \Pr\left(X_{t_0} \leq D\right) \cdot \prod_{1 \leq j \leq i} \left[1 + \left(1 - F(t_j)\right)^{(2 \cdot k_0)^j}\right].$$

We now move on to Inequality (7.3). We start by defining $m_i = i + i \cdot \lceil \log_2 k_0 \rceil$ and $n_i' = 2^{m_i}$. It is obvious that $n_i = 2^i \cdot k_0^i = 2^{i+i\cdot\log_2 k_0} \leq 2^{m_i} = n_i'$ holds. Therefore, $\Pr\left(X_{t_i}^{(n_i)} \leq D\right) \leq \Pr\left(X_{t_i}^{(n_i')} \leq D\right)$ is also valid. The rest follows from Theorem 7.6:

$$\Pr\left(X_{t_i}^{(n_i')} \leq D\right) = \Pr(X_{t_i} \leq D) \cdot \prod_{0 \leq j < m_i} \left[1 + \left(1 - F(t_i)\right)^{k_i \cdot 2^j} \cdot \left(1 - F(\ell_i)\right)^{2^j}\right].$$

This is precisely the desired property. □

We now consider a case in which this theorem is of special interest. First, the concept of unimodality is formally introduced.

Definition 7.8 ([146])**.** Let $g : \mathbb{R} \to \mathbb{R}$ be a continuously differentiable function with derivative g'. The function g is **unimodal** if and only if there is a unique point $x \in \mathbb{R}$ with $g'(x) = 0$ and g has a global maximum at x.

Note that this definition implies that g is strictly increasing on $(-\infty, x)$ and strictly decreasing on (x, ∞).

Suppose the primary purpose is to find a parallel restart strategy on n processors minimizing an expected value. Shylo et al. [159] proved that if the hazard rate function (see Definition 4.9) is unimodal, then $t_n^* > t^*$ applies. Here, t_n^* is the restart time of the optimal fixed-cutoff strategy on n processors (w. r. t. an expected value), and t^* is the restart time of the optimal fixed-cutoff strategy on one processor (w. r. t. an expected value). This result can be generalized to the statement that the restart times are strictly monotonically increasing in the number of processors in this case. This generalization is implicitly stated in [159]. For the proof, one can restrict oneself to the techniques introduced in this paper. In the following, we briefly sketch the proof with reference to the used methods from [159].

Proposition 7.9 ([159])**.** *Assume $\mathcal{A}$ is a randomized algorithm whose runtime behavior is described by a random variable X having a unimodal hazard rate function r. In addition, $X_{t_n^*}^{(n)}$ and $X_{t_m^*}^{(m)}$ are random variables describing a parallelized version of $\mathcal{A}$ on $n \in \mathbb{N}$ and respectively $m \in \mathbb{N}$ processors with $n > m$. Furthermore, this parallelized version employs a parallel fixed-cutoff strategy with restart times $t_n^* \in \mathbb{R}_+$ and respectively $t_m^* \in \mathbb{R}_+$. It is*

further assumed that t_n^* and t_m^* are optimal restart times with respect to an expected value for the parallelized algorithm on n and on m processors, respectively. If

$$\frac{E\left[X_{t_m^*}^{(m)}\right]}{E\left[X_{t_n^*}^{(n)}\right]} < \frac{n}{m} \tag{7.6}$$

is valid, then $t_n^* > t_m^*$ is also valid.

Proof sketch. Let $t^* \in \mathbb{R}_+$ be the optimal restart time regarding an expected value without parallelization. First, we apply Theorem 3 from [159], whose statement can be described as follows:

$$t_n^* > t^*$$
$$t_m^* > t^*.$$

Furthermore, the following characterization of the expected value $E\left[X_{t_n^*}^{(n)}\right]$ has been proven by Shylo et al. [159]:

$$E\left[X_{t_n^*}^{(n)}\right] = \frac{1}{n \cdot r(t_n^*)} = \frac{1 - F(t_n^*)}{n \cdot f(t_n^*)},$$

where F is the cdf and f is the pdf of X. This fact can be utilized to transform Inequality (7.6) as follows:

$$\begin{aligned} &\frac{1 - F(t_m^*)}{m \cdot f(t_m^*)} \cdot \frac{n \cdot f(t_n^*)}{1 - F(t_n^*)} < \frac{n}{m} \\ \Leftrightarrow\ & r(t_n^*) = \frac{f(t_n^*)}{1 - F(t_n^*)} < \frac{f(t_m^*)}{1 - F(t_m^*)} = r(t_m^*). \end{aligned} \tag{7.7}$$

Furthermore, Theorem 1 in [159] states that the hazard rate function r is non-increasing on an interval $[x_1, x_2]$ with $t^* \in [x_1, x_2]$. Since the hazard rate function r is unimodal by assumption, r is strictly monotonically decreasing on the interval $[x_1, \infty)$. Since both $t_n^* > t^*$ and $t_m^* > t^*$, it follows that t_n^* and t_m^* are both in the interval $[x_1, \infty)$. The monotonicity of r on $[x_1, \infty)$ in conjunction with Inequality (7.7) immediately suggests that $t_n^* > t_m^*$ must hold.

Proposition 7.9 primarily serves as motivation for Theorem 7.7. For example, suppose the main goal is obtaining a low expected value. Then, according to Proposition 7.9, the restart times are often monotonically increasing in the number of

processors. Nevertheless, the behavior concerning a deadline might also be of interest. The probabilities concerning the deadline can then be analyzed by means of Theorem 7.7.

A similar situation is examined in Section 6.7.5. The primary purpose is to minimize the par2 score, which has similarities with the expected value. Simultaneously, the number of instances that can be solved within the deadline of 5000 seconds is also investigated. If, in addition, a parallelized version of PROBSAT were to be examined, this would correspond precisely to the scenario in which Theorem 7.7 is applicable.

7.2 IDEAL RESTART TIMES

We now focus on the analysis of which fixed-cutoff strategies maximize the probability of meeting the deadline. To avoid confusion with the optimal restart time in terms of the expected value, we refer to the best restart time w. r. t. the probability of meeting a deadline as the ideal restart time.

Definition 7.10. Assume $X_t^{(n)}$ to be a random variable describing the behavior of the parallelized algorithm $\mathcal{A}_t^{(n)}$ running on $n \in \mathbb{N}$ processors and using the parallel fixed-cutoff strategy $(t, t, \dots)$ with $t \in \mathbb{R}_+$. In addition, there is a deadline $D \in \mathbb{R}_+$ for $\mathcal{A}_t^{(n)}$. The value t is an **ideal restart time** with respect to the probability of completion on n processors if and only if

$$\Pr\left(X_t^{(n)} \leq D\right) \geq \Pr\left(X_z^{(n)} \leq D\right)$$

holds for all $z \in \mathbb{R}_+$.

It is now possible to specify a necessary condition for an ideal restart time. The following result has been shown by van Moorsel and Wolter [130] for the special case when the algorithm runs on only one processor. Here, we generalize this result to parallelized algorithms.

Theorem 7.11. *Let X be a continuous random variable describing the runtime behavior of an algorithm $\mathcal{A}$ and let r be the hazard rate function of X. Furthermore, assume $X_{t^*}^{(n)}$ to be a random variable describing the behavior of the parallelized algorithm $\mathcal{A}_{t^*}^{(n)}$ running on $n \in \mathbb{N}$ processors and using the parallel fixed-cutoff strategy $(t^*, t^*, \dots)$ with $t^* \in \mathbb{R}_+$.*

Finally, there is a deadline $D \in \mathbb{R}_+$ for $\mathcal{A}_{t^*}^{(n)}$. If t^* is an ideal restart time with respect to the probability of completion on n processors, then t^* satisfies the following condition:

$$r(t^*) = r\left(D - \left\lfloor \frac{D}{t^*} \right\rfloor \cdot t^*\right). \tag{7.8}$$

Proof. The probability of completion can be specified by Lemma 7.4 as a function of the restart time $t \in \mathbb{R}_+$:

$$P(t) = \Pr\left(X_t^{(n)} \leq D\right) = 1 - \big(1 - F(t)\big)^{k \cdot n} \cdot \big(1 - F(\ell)\big)^n,$$

where $k = \lfloor \frac{D}{t} \rfloor$ is the number of restarts and $\ell = D - k \cdot t$ is the leftover time for the last run before the deadline. Furthermore, F represents the cdf and f the pdf of X. In order to identify the extrema of this probability, we investigate the derivative p' of $P(t)$ with respect to t:

$$p'(t) = nk \cdot f(t)\big(1 - F(\ell)\big)^n \big(1 - F(t)\big)^{nk-1} - nk \cdot f(\ell)\big(1 - F(t)\big)^{nk} \big(1 - F(\ell)\big)^{n-1}.$$

The derivative has to be zero for maxima. For this reason, we equate $p'(t)$ with zero in the next step:

$$\begin{aligned} & nkf(t)\big(1 - F(\ell)\big)^n \big(1 - F(t)\big)^{nk-1} - nkf(\ell)\big(1 - F(t)\big)^{nk} \big(1 - F(\ell)\big)^{n-1} = 0 \\ \Leftrightarrow\ & f(t)\big(1 - F(t)\big)^{-1} - f(\ell)\big(1 - F(\ell)\big)^{-1} = 0 \\ \Leftrightarrow\ & r(t) = \frac{f(t)}{1 - F(t)} = \frac{f(\ell)}{1 - F(\ell)} = r(\ell). \end{aligned} \tag{7.9}$$

Therefore, any ideal restart time with respect to the probability of completion has to satisfy Equation (7.9). □

As a side note, Proposition 7.9 and Theorem 7.11 are the only results in this chapter that requires X to be a continuous random variable.

It is especially remarkable that the condition from Equation (7.8) does not depend on the number of processors n. This implies that the local maxima are at the same locations, regardless of whether the process is analyzed on one or on n processors. This also leads to the hypothesis that the global maxima, i. e., the ideal restart times, are the same regardless of the number of processors. The following theorem confirms this assumption.

Theorem 7.12. *Assume $X_t^{(n)}$ to be a random variable describing the behavior of a parallelized algorithm $\mathcal{A}_t^{(n)}$ running on $n \in \mathbb{N}$ processors and using the parallel fixed-cutoff strategy $(t, t, \dots)$ with $t \in \mathbb{R}_+$. In addition, there is a deadline $D \in \mathbb{R}_+$ for $\mathcal{A}_t^{(n)}$. The value t is an ideal restart time regarding the probability of completion on n processors if and only if t is an ideal restart time on one processor.*

Proof. Suppose $t^* \in \mathbb{R}_+$ is an ideal restart time with respect to the probability of completion on one processor. The following property is therefore valid:

$$\Pr\left(X_{t^*} \leq D\right) \geq \Pr\left(X_z \leq D\right), \ \forall z \in \mathbb{R}_+,$$

where X_t is a random variable representing the non-parallelized process (i.e. the process running on only one processor) using the fixed-cutoff restart strategy $(t, t, \dots)$. Furthermore, let $t_n^* \in \mathbb{R}_+$ be an ideal restart time regarding the probability of completion on n processors. Put differently:

$$\Pr\left(X_{t_n^*}^{(n)} \leq D\right) \geq \Pr\left(X_z^{(n)} \leq D\right), \ \forall z \in \mathbb{R}_+. \tag{7.10}$$

In addition, suppose the following also holds true:

$$\Pr\left(X_{t^*} \leq D\right) > \Pr\left(X_{t_n^*} \leq D\right).$$

In other words, t^* is strictly superior to t_n^* on a single processor. The two probabilities can be analyzed more closely. For this purpose, define $k^* = \lfloor \frac{D}{t^*} \rfloor$ and $\ell^* = D - k^* \cdot t^*$ as the number of restarts and the leftover time for the last run associated with the restart time t^*. Accordingly, let $k_n^* = \lfloor \frac{D}{t_n^*} \rfloor$ and $\ell_n^* = D - k_n^* \cdot t_n^*$ be the number of restarts and the leftover time corresponding to t_n^*. Additionally, let F be the appropriate cdf as required in Lemma 7.4. This lemma allows a more detailed specification of the probabilities:

$$\begin{aligned}
&\Pr\left(X_{t^*} \leq D\right) > \Pr\left(X_{t_n^*} \leq D\right) \\
\Leftrightarrow\ & 1 - \left(1 - F(t^*)\right)^{k^*} \cdot \left(1 - F(\ell^*)\right) > 1 - \left(1 - F(t_n^*)\right)^{k_n^*} \cdot \left(1 - F(\ell_n^*)\right) \\
\Leftrightarrow\ & \left(1 - F(t_n^*)\right)^{k_n^*} \cdot \left(1 - F(\ell_n^*)\right) > \left(1 - F(t^*)\right)^{k^*} \cdot \left(1 - F(\ell^*)\right) \\
\Leftrightarrow\ & \left(1 - F(t_n^*)\right)^{n \cdot k_n^*} \cdot \left(1 - F(\ell_n^*)\right)^n > \left(1 - F(t^*)\right)^{n \cdot k^*} \cdot \left(1 - F(\ell^*)\right)^n \\
\Leftrightarrow\ & 1 - \left(1 - F(t^*)\right)^{n \cdot k^*} \cdot \left(1 - F(\ell^*)\right)^n > 1 - \left(1 - F(t_n^*)\right)^{n \cdot k_n^*} \cdot \left(1 - F(\ell_n^*)\right)^n \\
\Leftrightarrow\ & \Pr\left(X_{t^*}^{(n)} \leq D\right) > \Pr\left(X_{t_n^*}^{(n)} \leq D\right).
\end{aligned} \tag{7.11}$$

In the last line, Lemma 7.4 is applied once more. Furthermore, Inequality (7.11) contradicts the definition of t_n^* as an ideal restart time for n processors (compare Inequality (7.10)). We can therefore conclude that t_n^* is an ideal restart time for one processor. Since only equivalence transformations are used, the reverse direction also follows analogously. □

Furthermore, Proposition 7.9 together with Theorem 7.12 imply that the optimal restart time in terms of the expected value and the ideal restart time in terms of the probability of meeting a deadline are generally not identical.

7.3 THE SCALING BEHAVIOR

In this section, we aim to quantify the effect of parallelization on the probability of meeting the deadline. Shylo et al. [159] evaluated the effect of parallelization employing the optimal restart strategy in order to assess the impact on the expected value. To be more precise, they studied the speedup (see Equation (6.3)) and proved that it is at best linear in the number of processors. In mathematical terms, their statement can be described as follows:

$$\frac{E[X_{t^*}]}{E\left[X_{t_n^*}^{(n)}\right]} \leq n. \tag{7.12}$$

Here, X_{t^*} is a random variable describing the runtime behavior when using an optimal restart time (regarding the expected value) t^* on one processor. Likewise, $X_{t_n^*}^{(n)}$ represents a random variable characterizing the runtime behavior utilizing an optimal restart time t_n^* on n processors.

Our objective is to compare the effect on the probability of meeting a deadline with Inequality (7.12). However, the probability of meeting the deadline itself is an inappropriate measure for quantifying a type of speedup. To illustrate, consider an increase of the probability from $\frac{1}{4}$ to $\frac{1}{2}$ on the one hand and from $\frac{1}{2}$ to 1.0 on the other hand. Although both increments double the respective probability, it is evident that the second increment has a much higher effect because it turns an intrinsically probabilistic procedure into a deterministic one. Thus, increases in probability have differing effects, and one, therefore, requires a different measure.

For this purpose, we define a suitable expected value. Say an algorithm $\mathcal{A}$ is subject to a certain deadline $D \in \mathbb{R}_+$ and we repeat the execution of the algorithm $\mathcal{A}$ with the same deadline D until it succeeds. Let Y be a random variable counting the

number of failed attempts of $\mathcal{A}$ subject to the deadline D. The random variable Y follows a geometric distribution.

Definition 7.13 ([49])**.** A discrete random variable X with pmf

$$p(k) = (1-p)^k \cdot p,\ k \in \mathbb{N}_0$$

has a **geometric distribution** with parameter $p \in (0,1]$. The expected value of X is given as follows:

$$E[X] = \frac{1-p}{p}.$$

In our case, the parameter p of the geometric distribution is given by the probability of meeting the deadline. We are naturally interested in ensuring that the expected value $E[Y]$ is as close as possible to zero. Most importantly, this approach allows us to evaluate the effect of parallelization. For this purpose, the expected value of the number of failed attempts is calculated once with parallelization and once without. If the ratio of these two expected values is calculated, the result is a type of speedup that can be compared with the finding by Shylo et al. [159] (see Inequality (7.12)).

In the following, we focus on the case in which the restart times remain constant for an increasing number of processors. This is because Theorem 7.12 shows that the ideal restart times are identical for both the parallel and sequential cases. Next, we show that the speedup described above is superlinear in the number of processors.

Theorem 7.14. *Assume $\mathcal{A}$, $\mathcal{A}_t$ and $\mathcal{A}_t^{(n)}$ to be three related algorithms. In this context, $\mathcal{A}_t$ is a modified version of $\mathcal{A}$ using the fixed-cutoff strategy with restart time $t \in \mathbb{R}_+$ and furthermore $\mathcal{A}_t^{(n)}$ is a parallelized version of $\mathcal{A}$ running on $n = 2^i$, $i \in \mathbb{N}$, processors employing the parallel fixed-cutoff strategy $(t, t, \dots)$. Let X be a random variable describing the runtime behavior of $\mathcal{A}$ and let F be the cdf of X. In addition, suppose Y_t is a random variable indicating the number of failed attempts of $\mathcal{A}_t$ to meet a deadline $D \in \mathbb{R}_+$. Likewise, let $Y_t^{(n)}$ be a random variable describing the number of failed attempts of $\mathcal{A}_t^{(n)}$ to meet the deadline D. If $F(t) \in (0,1)$ is valid, then*

$$\frac{E[Y_t]}{E\left[Y_t^{(n)}\right]} = \omega(n) \tag{7.13}$$

is also valid.

Proof. Consider two random variables X_t and $X_t^{(n)}$ describing the runtime behavior of $\mathcal{A}_t$ and $\mathcal{A}_t^{(n)}$ respectively. The random variables Y_t and $Y_t^{(n)}$ are both geometrically distributed, hence the ratio in Equation (7.13) is given as follows:

$$\frac{E[Y_t]}{E\left[Y_t^{(n)}\right]} = \frac{(1-p_1)\cdot p_2}{(1-p_2)\cdot p_1}.$$

Here, p_1 corresponds to the probability of $\mathcal{A}_t$ meeting the deadline and accordingly p_2 is the probability that $\mathcal{A}_t^{(n)}$ meets the deadline. The values of p_1 and p_2 can be specified as follows by means of the random variables X_t and $X_t^{(n)}$:

$$p_1 = \Pr(X_t \leq D)$$
$$p_2 = \Pr\left(X_t^{(n)} \leq D\right).$$

These two probabilities can now be analyzed more closely by employing the methods introduced in this chapter. In the following, $k = \lfloor \frac{D}{t} \rfloor$ is the number of restarts and $\ell = D - k \cdot t$ is the leftover time for the last run before the deadline:

$$\begin{aligned}\frac{E[Y_t]}{E\left[Y_t^{(n)}\right]} &= \frac{[1-\Pr(X_t \leq D)]\cdot \Pr\left(X_t^{(n)} \leq D\right)}{\left[1-\Pr\left(X_t^{(n)} \leq D\right)\right]\cdot \Pr(X_t \leq D)} \\ &= \frac{[1-\Pr(X_t \leq D)]\cdot \prod_{j=0}^{i-1}\left(1+\left(1-F(t)\right)^{2^j\cdot k}\cdot\left(1-F(\ell)\right)^{2^j}\right)}{1-\Pr\left(X_t^{(n)} \leq D\right)}.\end{aligned}$$

Here, Theorem 7.6 is applied to $\Pr\left(X_t^{(n)} \leq D\right)$ and the fraction is reduced accordingly. In the next step, Lemma 7.4 is utilized in both the denominator and the numerator:

$$\begin{aligned}
\frac{E\left[Y_t\right]}{E\left[Y_t^{(n)}\right]} &= \frac{\left[1-\Pr\left(X_t \leq D\right)\right] \cdot \prod_{j=0}^{i-1}\left(1+\left(1-F(t)\right)^{2^j \cdot k} \cdot \left(1-F(\ell)\right)^{2^j}\right)}{1-\Pr\left(X_t^{(n)} \leq D\right)} \\
&= \frac{\left(1-F(t)\right)^k \cdot \left(1-F(\ell)\right) \cdot \prod_{j=0}^{i-1}\left(1+\left(1-F(t)\right)^{2^j \cdot k} \cdot \left(1-F(\ell)\right)^{2^j}\right)}{\left(1-F(t)\right)^{k \cdot 2^i} \cdot \left(1-F(\ell)\right)^{2^i}} \\
&= \frac{\prod_{j=0}^{i-1}\left(1+\left(1-F(t)\right)^{2^j \cdot k} \cdot \left(1-F(\ell)\right)^{2^j}\right)}{\left(1-F(t)\right)^{k \cdot (2^i-1)} \cdot \left(1-F(\ell)\right)^{2^i-1}} \\
&\geq \frac{1}{\left(1-F(t)\right)^{k \cdot (2^i-1)} \cdot \left(1-F(\ell)\right)^{2^i-1}}.
\end{aligned} \tag{7.14}$$

It is obvious that the last value approaches infinity as i approaches infinity. Next, we compare the limiting behavior of the right side of Inequality (7.14) against 2^i as i tends towards infinity:

$$\lim_{i \to \infty} \frac{\frac{1}{\left(1-F(t)\right)^{k \cdot (2^i-1)} \cdot \left(1-F(\ell)\right)^{2^i-1}}}{2^i}. \tag{7.15}$$

Although i is defined as a natural number, in the following, we consider an extension into the real numbers. Since both the numerator and denominator are approaching infinity, L'Hospitals rule can be applied to the limit in Equation (7.15):

$$\begin{aligned}
&\lim_{i \to \infty} \frac{\frac{1}{\left(1-F(t)\right)^{k \cdot (2^i-1)} \cdot \left(1-F(\ell)\right)^{2^i-1}}}{2^i} \\
&= \lim_{i \to \infty} \frac{-\frac{\ln(2) \cdot \left[k \ln\left(1-F(t)\right) + \ln\left(1-F(\ell)\right)\right] \cdot 2^i}{\left(1-F(t)\right)^{k(2^i-1)} \cdot \left(1-F(\ell)\right)^{2^i-1}}}{\ln(2) \cdot 2^i} \\
&= \lim_{i \to \infty} -\frac{k \ln\left(1-F(t)\right) + \ln\left(1-F(\ell)\right)}{\left(1-F(t)\right)^{k(2^i-1)} \cdot \left(1-F(\ell)\right)^{2^i-1}} = \infty.
\end{aligned}$$

Thus, $\frac{E[Y_t]}{E\left[Y_t^{(n)}\right]} = \omega(2^i) = \omega(n)$ is valid. The proof is therefore complete. □

Here, we confined ourselves on the number of processors being a power of two in order to be able to analyze the corresponding probabilities by means of Theorem 7.6. A generalization of this result to allow for any number of processors would also be of interest.

It is noteworthy that Theorem 7.14 does not require the restart times to be ideal. Such a scenario occurs, for example, in practical applications where the precise runtime distribution is usually unknown. One can then, at best, rely on empirically-based estimates to determine the restart time. In such a scenario, the restart time is, therefore, generally not ideal. Nevertheless, according to Theorem 7.14, a superlinear effect is obtained (with respect to meeting the deadline).

7.4 CONCLUSION

In this chapter, we consider the effects of parallel restart strategies on the probability of meeting a deadline. Essentially, two different models are discussed.

When trying to minimize the expected value by restarts in a parallel setting, the optimal restart times are monotonically increasing in the number of processors under reasonable assumptions (cf. [159]). In case the characteristics regarding a deadline need to be checked in this scenario, Theorem 7.7 can be applied for the analysis.

On the other hand, we demonstrate in Theorem 7.12 that the ideal restart times (i. e., regarding the probability to meet a deadline) remain constant with an increasing number of processors. Furthermore, according to Theorem 7.14, this situation yields a superlinear speedup in the number of processors. For the calculation of the speedup, we take the number of failed attempts to meet the deadline. Since the superlinearity, in particular, does not require ideal restart times, this result is also applicable in practice if the underlying distribution is unknown.

Altogether, there is an interesting discrepancy between the scenario in which an expected value is optimized compared to that in which the probability of meeting a deadline is optimized. In the first case, the speedup is at most linear. In contrast, the speedup in terms of meeting a deadline is superlinear. From this observation, we conclude that the combination of parallelization and restarts is highly efficient for meeting a deadline, whereas the benefit of parallelizing using optimal restarts is limited when considering the expected value.

A possible direction for further research might be to consider these two issues as a multi-objective optimization problem. One would then attempt to simultaneously optimize the expected value and the probability to meet a deadline. The deadline can be seen as a soft deadline in such a setting. This would correspond to a process that should be completed within the deadline but which is, in principle, allowed to exceed it. All questions regarding restarting properties would have to be addressed again in this situation.

8

DISCUSSION

The primary purpose of restarts is to improve the runtime of algorithms. However, there are comparatively few theoretical results concerning the properties of restart strategies. In this thesis, we explore various, mostly theoretical, aspects of restart strategies. One objective of this thesis is to contribute to the theoretical comprehension of such strategies. This chapter first summarizes the main results obtained in this thesis and proposes potential directions for further research.

8.1 MAIN CONTRIBUTIONS

1. **Complexity of restarts**
   Chapter 3 investigates the complexity-theoretical characteristics of restart strategies. To the best of our knowledge, this is a novel approach to the analysis of restart strategies. For this purpose, it is assumed that the probability distribution is provided as a function in the form of a straight-line program. We showed that almost all questions related to restarts are hard with respect to the complexity. To be more precise, deciding whether restarts positively affect the expected value is NP-hard (cf. Section 3.2.1). Even the computation of the expected value for a given restart strategy is #P-hard (cf. Section 3.2.2). For the calculation of the optimal restart time, an interesting separation occurs. Considering the case in which the provided straight-line program represents a pmf, the existence of an efficient approximation algorithm implies $\mathrm{P} = \mathrm{NP}$ (cf. Theorem 3.14). One can infer that the existence of such an algorithm is unlikely. On the other hand, we developed an efficient $(4+\varepsilon)$-approximation algorithm for the case in which the cdf is given (cf. Corollary 3.19). Thus, in this situation, it is possible to obtain good restart times.

   This chapter mostly assumed that the given functions are indeed a valid description of a probability distribution. Section 3.3 examines how difficult it is to check this precondition. We found that checking whether a function describes a pmf is $\mathrm{P}^{\#\mathrm{P}}$-hard; whereas checking the cdf property is coNP-hard.

2. **Continuous restart strategies**
   As discussed above, finding an optimal restart time and quantifying the speedup is, in general, a challenging problem, even if the probability distribution is known. Chapter 4, however, demonstrates that the problem can be addressed efficiently by confining oneself to certain types of distribution. For this purpose, we investigate continuous distributions in terms of their restart quantiles. In particular, conditions are established to determine whether there are useful restart quantiles (cf. Section 4.1) and if so, how to identify optimal restart quantiles (cf. Section 4.2). For a large class of distributions, namely long-tailed distributions, restarts can be shown to be useful by utilizing these conditions (cf. Theorem 4.12). Furthermore, it is proven that scale parameters do not affect the restart properties (cf. Section 4.3). Therefore, such a scale parameter can be set to a convenient value in the analysis, simplifying the evaluation considerably. By means of these findings, three widespread distributions for describing the runtime behavior of algorithms are investigated. It is shown that there are always useful restart quantiles for any lognormal distribution (cf. Theorem 4.20). Furthermore, the Weibull and the generalized Pareto distribution exhibit an intriguing property. If restarts are useful for these distributions, the optimal restart time approaches zero (cf. Theorem 4.23 and 4.27). Therefore, restarts should be performed as early as possible.

   The main part of this thesis focuses on fixed-cutoff restarts, which have been proven to be the optimal strategy when using the optimal restart time [123]. However, since this optimal restart time is usually unknown in practice, Luby's strategy is frequently employed in algorithms. This is a universal strategy, which is at most a logarithmic factor worse than the optimal fixed-cutoff strategy in a discrete setting [123]. Chapter 5 is devoted to such universal strategies in a continuous setting. It is shown that, in general, there are no universal strategies providing a performance guarantee comparable to Luby's strategy in the discrete setting (cf. Theorem 5.4). On the other hand, we define a large subclass of continuous distributions on which Luby's strategy provides a performance guarantee (cf. Theorem 5.7). This also indicates that, in a continuous setting, one should not merely resort to Luby's strategy without first checking whether a performance guarantee is applicable.

3. **Predicting runtime distributions**
   There are two main arguments against employing a fixed cutoff strategy. First,

as shown in Chapter 3, the decision of whether to restart is NP-hard, even if the runtime distribution is known. Secondly, the exact runtime distribution is usually unknown, and thus the optimal restart time cannot be calculated. In Chapter 6, this problem is addressed exemplarily for the SAT solver PROBSAT [14]. For this purpose, machine learning techniques are used to predict the runtime distribution of previously unsolved instances. Based on the predicted distributions and the techniques described in Chapter 4, restart times can be determined. The quality of the predicted distributions is so good that the restart strategies derived from these distributions yield an average speedup factor of more than 50 on instances with hidden solutions. Furthermore, these techniques improve the performance of PROBSAT on the instances of the SAT Competition 2018 to such an extent that the improved version would have achieved the second place in the competition, competing closely with the winning solver. Above all, these results demonstrate that the methods presented in Chapter 4 are not only of theoretical interest but can also be applied in practice in conjunction with machine learning techniques.

4. **Impact of parallel restart strategies on the probability of meeting deadlines** Finally, a slightly different scenario is considered in Chapter 7. The objective is to maximize the probability that a randomized, parallelized algorithm meets a certain deadline. In particular, it is proven that the ideal strategy in a parallel setting is identical to the ideal strategy in a sequential context (cf. Section 7.2). This is worth mentioning since this property does not apply when optimizing, for example, with respect to the expected value [159]. Another noteworthy result is that the expected number of failures to meet the deadline is superlinearly (in the number of used processors) approaching zero (cf. Section 7.3). This contrasts with parallel optimal restart strategies regarding the expected value: The expected value of the optimal restart strategy scales sublinearly in the number of processors [159].

8.2 OUTLOOK

There are some opportunities to extend the research presented in this thesis. Some of these possibilities are briefly illustrated in this section. The most obvious starting point might be the part concerning complexity theory. Overall, the complexity-theoretical consideration of restarts is relatively young, and accordingly, many

aspects are still unexplored. For example, one could analyze similar questions for continuous distributions. However, this would also require the choice of a suitable computational model, such as the model developed by Ko [102]. Furthermore, only fixed-cutoff strategies and only the optimization concerning the expected value are considered. Therefore, it might be worthwhile to investigate other related issues, as well. For example, in Chapter 7, we addressed the problem of meeting a deadline by means of restarts. It might also be interesting to study the complexity-theoretical characteristics of this problem. It is particularly intriguing whether it is possible to devise an efficient algorithm capable of finding an ideal restart strategy.

Another not entirely resolved question is *why* restarts might be worthwhile. This work attempts to address this question through runtime distributions. For example, the literature often employs so-called power-laws to describe algorithms' behavior (e. g. [76]). This is a kind of distribution that resembles the generalized Pareto distribution for extreme values. However, this way, the problem only shifts to justifying why an algorithm follows a certain distribution. In particular, there are only few theoretical explanations concerning this issue. There has been some progress regarding the reasoning for the occurrence of power-laws in the runtime behavior of backtracking algorithms (e. g. [36, 39]). But as seen, for example, in Chapter 6, there are other types of algorithms better described by other kinds of distributions, such as the lognormal distribution. Overall, there is a lack of theoretical justifications why this might be the case and, by extension, why restarts are useful.

Finally, it should be mentioned that the type of restarts discussed in this thesis assumes the independence of the runs; this is the case, for example, in stochastic local search algorithms. Other types of algorithms, however, sometimes implement restarts exploiting inter-run dependencies. For instance, in so-called CDCL SAT solvers, such restarts are used [28]. The question on the effect of this type of restarts and under which circumstances they are useful is also not entirely resolved, although there has been some progress in this direction [109, 144].

BIBLIOGRAPHY

[1] Charu C. Aggarwal. *Neural networks and deep learning*. Springer, 2018.

[2] Helmut Alt, Leonidas Guibas, Kurt Mehlhorn, Richard M. Karp, and Avi Wigderson. *A Method for Obtaining Randomized Algorithms with Small Tail Probabilities*. Tech. rep. ICSI Technical Report tr-91-057. Berkeley, CA, USA: ICSI, 1991, p. 8.

[3] Rajeev Alur and David L. Dill. "A theory of timed automata." In: *Theoretical computer science* 126.2 (1994), pp. 183–235.

[4] Alejandro Arbelaez, Charlotte Truchet, and Philippe Codognet. "Using sequential runtime distributions for the parallel speedup prediction of SAT local search." In: *Theory and Practice of Logic Programming* 13.4-5 (2013), pp. 625–639.

[5] Alejandro Arbelaez, Charlotte Truchet, and Philippe Codognet. "Using sequential runtime distributions for the parallel speedup prediction of SAT local search." In: *Theory and Practice of Logic Programming* 13.4-5 (2013), pp. 625–639.

[6] Alejandro Arbelaez, Charlotte Truchet, and Barry O'Sullivan. "Learning Sequential and Parallel Runtime Distributions for Randomized Algorithms." In: *ICTAI 2016: 28th International Conference on Tools with Artificial Intelligence, San Jose, California, USA*. IEEE, 2016, pp. 655–662.

[7] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[8] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. "Clause-learning algorithms with many restarts and bounded-width resolution." In: *Journal of Artificial Intelligence Research* 40 (2011), pp. 353–373.

[9] Gilles Audemard and Laurent Simon. "Refining restarts strategies for SAT and UNSAT." In: *Principles and Practice of Constraint Programming*. 2012, pp. 118–126.

[10] László Babai. *Monte-Carlo algorithms in graph isomorphism testing*. Tech. rep. DMS 79-10. Département de mathématiques et de statistique, Université de Montréal, 1979.

[11] Adrian Balint. *probSAT (version SC13_v2)*. Source code. Retrieved from https://github.com/adrianopolus/probSAT. 2013.

[12] Adrian Balint and Andreas Fröhlich. "Improving stochastic local search for SAT with a new probability distribution." In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2010, pp. 10–15.

[13] Adrian Balint and Norbert Manthey. "SparrowToRiss 2018." In: *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Vol. B-2018-1. Department of Computer Science Series of Publications B, University of Helsinki, 2018, pp. 38–39.

[14] Adrian Balint and Uwe Schöning. "Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break." In: *Theory and Applications of Satisfiability Testing – SAT 2012*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2012, pp. 16–29.

[15] Adrian Balint and Uwe Schöning. "probSAT." In: *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*. Vol. B-2013-1. Department of Computer Science Series of Publications B, University of Helsinki, 2013, p. 70.

[16] Adrian Balint and Uwe Schöning. "probSAT and pprobSAT." In: *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions*. Vol. B-2014-2. Department of Computer Science Series of Publications B, University of Helsinki, 2014, p. 63.

[17] Adrian Balint and Uwe Schöning. "Engineering a lightweight and efficient local search SAT solver." In: *Algorithm Engineering*. Springer, 2016, pp. 1–18.

[18] August A. Balkema and Laurens De Haan. "Residual Life Time at Great Age." In: *The Annals of probability* (1974), pp. 792–804.

[19] Tomáš Balyo and Lukáš Chrpa. "Using Algorithm Configuration Tools to Generate Hard SAT Benchmarks." In: *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS '18)*. AAAI Press, 2018, pp. 133–137.

[20] David F. Barrero, Pablo Muñoz, David Camacho, and María D. R-Moreno. "On the statistical distribution of the expected run-time in population-based search algorithms." In: *Soft Computing* 19.10 (2015), pp. 2717–2734.

[21] Wolfgang Barthel, Alexander K Hartmann, Michele Leone, Federico Ricci-Tersenghi, Martin Weigt, and Riccardo Zecchina. "Hiding solutions in random satisfiability problems: A statistical mechanics approach." In: *Physical review letters* 88.18 (2002), p. 188701.

[22] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive Search and Intelligent Optimization*. Vol. 45. Operations Research/Computer Science Interfaces Series. Boston, MA: Springer US, 2009.

[23] Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. "Testing that distributions are close." In: *Foundations of Computer Science, 2000. Proceedings*. IEEE, 2000, pp. 259–269.

[24] Roberto J. Bayardo Jr. and Robert Schrag. "Using CSP Look-Back Techniques to Solve Real-World SAT Instances." In: *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*. AAAI Press, 1997, pp. 203–208.

[25] Sergey Belan. "Restart could optimize the probability of success in a Bernoulli trial." In: *Physical review letters* 120.8 (2018), p. 080601.

[26] Olivier Bénichou, Claude Loverdo, Michel Moreau, and Raphael Voituriez. "Intermittent search strategies." In: *Reviews of Modern Physics* 83.1 (2011), p. 81.

[27] Armin Biere. "Cadical, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2017." In: *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*. Vol. B-2017-1. Department of Computer Science Series of Publications B, University of Helsinki, 2017, pp. 14–15.

[28] Armin Biere, Marijn J. H. Heule, and Hans van Maaren. *Handbook of satisfiability*. Vol. 185. IOS press, 2009.

[29] Peter Borwein and Joe Hobart. "The extraordinary power of division in straight line programs." In: *The American Mathematical Monthly* 119.7 (2012), pp. 584–592.

[30] Olivier Bournez and Manuel L. Campagnolo. "A survey on continuous time computations." In: *New Computational Paradigms*. Springer, 2008, pp. 383–423.

[31] Leo Breiman. "Random forests." In: *Machine learning* 45.1 (2001), pp. 5–32.

[32] Shaowei Cai, Chuan Luo, and Kaile Su. "CCASat: solver description." In: *Proceedings of SAT challenge* 2012 (2012), p. 13.

[33] Clément L. Canonne. "A Survey on Distribution Testing: Your Data is Big. But is it Blue?" In: *Electronic Colloquium on Computational Complexity (ECCC)* 22.63 (2015).

[34] Clément L. Canonne, Ilias Diakonikolas, Themis Gouleakis, and Ronitt Rubinfeld. "Testing shape restrictions of discrete distributions." In: *Theory of Computing Systems* 62.1 (2018), pp. 4–62.

[35] Marek Capinski and Peter E. Kopp. *Measure, integral and probability*. Springer Science & Business Media, 2013.

[36] Alda Carvalho and Carlos Santos. "A Generator of Heavy-Tailed Search Trees." In: *Recent Developments in Modeling and Applications in Statistics*. Ed. by Paulo Eduardo Oliveira, Maria da Graça Temido, Carla Henriques, and Maurizio Vichi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 107–113.

[37] George Casella and Roger L. Berger. *Statistical inference*. 2nd ed. Duxbury Pacific Grove, CA, 2002.

[38] Champagne, Carl, and Hill. "Search theory, agent-based simulation, and u-boats in the Bay of Biscay." In: *Proceedings of the 2003 Winter Simulation Conference*. Vol. 1. IEEE, 2003, pp. 991–998.

[39] Hubie Chen, Carla Gomes, and Bart Selman. "Formal models of heavy-tailed behavior in combinatorial search." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2001, pp. 408–421.

[40] François Chollet et al. *Keras*. https://github.com/keras-team/keras. 2015.

[41] Andre A. Cire, Serdar Kadioglu, and Meinolf Sellmann. "Parallel Restarted Search." In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI'14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 842–848.

[42] Philippe Codognet and Daniel Diaz. "Yet another local search method for constraint solving." In: *International Symposium on Stochastic Algorithms*. Springer, 2001, pp. 73–90.

[43] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures." In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*. 1971, pp. 151–158.

[44] Yves Crama and Peter L. Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

[45] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of boolean constraint satisfaction problems*. Vol. 7. SIAM, 2001.

[46] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros. "Heavy-tailed probability distributions in the World Wide Web." In: *A practical guide to heavy tails* 1 (1998), pp. 3–26.

[47] Edwin L. Crow and Kunio Shimizu. *Lognormal Distributions: Theory and Applications*. Statistics: textbooks and monographs 88. Marcel Dekker, Inc., 1988.

[48] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. "Theta*: Any-angle path planning on grids." In: *Journal of Artificial Intelligence Research* 39 (2010), pp. 533–579.

[49] Yadolah Dodge. *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008.

[50] Andrew M. Edwards, Richard A. Phillips, Nicholas W. Watkins, Mervyn P. Freeman, Eugene J. Murphy, Vsevolod Afanasyev, Sergey V. Buldyrev, Marcos G. E. da Luz, Ernesto P. Raposo, H. Eugene Stanley, et al. "Revisiting Lévy flight search patterns of wandering albatrosses, bumblebees and deer." In: *Nature* 449.7165 (2007), pp. 1044–1048.

[51] Katharina Eggensperger, Marius Lindauer, and Frank Hutter. "Predicting Runtime Distributions using Deep Neural Networks." In: *arXiv:1709.07615 [cs]* (2017). arXiv: 1709.07615.

[52] Iddo Eliazar, Tal Koren, and Joseph Klafter. "Searching circular DNA strands." In: *Journal of Physics: Condensed Matter* 19.6 (2007), p. 065140.

[53] Martin R. Evans and Satya N. Majumdar. "Diffusion with Stochastic Resetting." In: *Physical Review Letters* 106.16 (2011), p. 160601.

[54] Shimon Even, Alan L. Selman, and Yacov Yacobi. "The complexity of promise problems with applications to public-key cryptography." In: *Information and Control* 61.2 (1984), pp. 159–173.

[55] Willliam Feller. *An Introduction to Probability Theory and its Applications*. 3rd ed. Vol. 1. John Wiley & Sons, 1970.

[56] Willliam Feller. *An Introduction to Probability Theory and its Applications*. Vol. 2. John Wiley & Sons, 1970.

[57] Matteo Fischetti and Michele Monaci. "Exploiting erraticism in search." In: *Operations Research* 62.1 (2014), pp. 114–122.

[58] Philip J. Fleming and John J. Wallace. "How not to lie with statistics: the correct way to summarize benchmark results." In: *Communications of the ACM* 29.3 (1986), pp. 218–221.

[59] J. C. Flores. "Dispersal time for ancient human migrations: Americas and Europe colonization." In: *EPL (Europhysics Letters)* 79.1 (2007), p. 18004.

[60] Ionut Florescu and Ciprian A. Tudor. *Handbook of probability*. John Wiley & Sons, 2013.

[61] Sergey Foss, Dmitry Korshunov, and Stan Zachary. *An introduction to heavy-tailed and subexponential distributions*. Vol. 6. Springer, 2011.

[62] Tobias Friedrich, Timo Kötzing, Francesco Quinzan, and Andrew M. Sutton. "Improving the Run Time of the (1 + 1) Evolutionary Algorithm with Luby Sequences." In: GECCO '18. New York, NY, USA: ACM, 2018, pp. 301–308.

[63] Tobias Friedrich, Timo Kötzing, and Markus Wagner. "A generic bet-and-run strategy for speeding up stochastic local search." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[64] Daniel Frost, Irina Rish, and Lluís Vila. "Summarizing CSP Hardness with Continuous Probability Distributions." In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*. AAAI'97/IAAI'97. Providence, Rhode Island: AAAI Press, 1997, pp. 327–333.

[65] John R. Frost and Lawrence D. Stone. *Review of search theory: advances and applications to search and rescue decision support*. Tech. rep. United States Coast Guard, 2001.

[66] Oliver Gableske. *kcnfgen (version 1.0)*. Source code. Retrieved from `https://www.gableske.net/downloads/kcnfgen_v1.0.tar.gz`. 2015.

[67] Oliver Gableske. *Dimetheus (version 2.113)*. Source code. Retrieved from `https://www.gableske.net/downloads/dimetheus_v2.113.tar.gz`. 2016.

[68] Oliver Gableske. "SAT Solving with Message Passing: A Dissertation." PhD thesis. Universität Ulm, 2016.

[69] Stanisław Gawiejnowicz. *Time-dependent scheduling*. Springer Science & Business Media, 2008.

[70] Ian P. Gent and Toby Walsh. "Unsatisfied variables in local search." In: *Hybrid problems, hybrid solutions* 27 (1995), p. 73.

[71] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees." In: *Machine learning* 63.1 (2006), pp. 3–42.

[72] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric Statistical Inference: Revised and Expanded*. 4th ed. Taylor & Francis, 2014.

[73] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[74] Carla P. Gomes and Bart Selman. "Algorithm portfolio design: theory vs. practice." In: *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*. 1997, pp. 190–197.

[75] Carla P. Gomes, Bart Selman, and Nuno Crato. "Heavy-tailed distributions in combinatorial search." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 1997, pp. 121–135.

[76] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. "Heavy-tailed phenomena in satisfiability and constraint satisfaction problems." In: *Journal of automated reasoning* 24.1-2 (2000), pp. 67–100.

[77] Carla P. Gomes, Bart Selman, and Henry Kautz. "Boosting combinatorial search through randomization." In: AAAI Press, 1998, pp. 431–437.

[78] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[79] William Sealy Gosset. "The Probable Error of a Mean." In: *Biometrika* 6.1 (1908). Originally published under the pseudonym "Student"., pp. 1–25.

[80] Shai Haim and Toby Walsh. "Restart Strategy Selection using Machine Learning Techniques." In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2009, pp. 312–325.

[81] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009.

[82] Marijn J. H. Heule. "Generating the Uniform Random Benchmarks." In: *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. Vol. B-2016-1. Department of Computer Science Series of Publications B, University of Helsinki, 2016, p. 59.

[83] Marijn J. H. Heule, Matti Jarvisalo, and Tomáš Balyo. *Results of the SAT Competition 2016*. website. Retrieved from https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=results. 2016.

[84] Marijn J. H. Heule, Matti Jarvisalo, and Martin Suda. *Instances of the random track of the SAT Competition 2018*. data collection. Retrieved from http://sat2018.forsyte.tuwien.ac.at/benchmarks/Random.zip. 2018.

[85] Marijn J. H. Heule, Matti Jarvisalo, and Martin Suda. *Results of the SAT Competition 2018*. website. Retrieved from http://sat2018.forsyte.tuwien.ac.at/indexbff8.html?cat=results. 2018.

[86] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. *Nonparametric statistical methods*. John Wiley & Sons, 2013.

[87] Holger H. Hoos. "A Mixture-Model for the Behaviour of SLS Algorithms for SAT." In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press / The MIT Press, 2002, pp. 661–667.

[88] Holger H. Hoos and Thomas Stützle. "Evaluating Las Vegas Algorithms: Pitfalls and Remedies." In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 238–245.

[89] Holger H. Hoos and Thomas Stützle. "Towards a characterisation of the behaviour of stochastic local search algorithms for SAT." In: *Artificial Intelligence* 112.1-2 (1999), pp. 213–232.

[90] Holger H. Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[91] Jinbo Huang. "The Effect of Restarts on the Efficiency of Clause Learning." In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. Vol. 7. San Francisco, 2007, pp. 2318–2323.

[92] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. 7th printing. Springer, 2013.

[93] Haixia Jia and Cristopher Moore. "How much backtracking does it take to color random graphs? Rigorous results on heavy tails." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 2004, pp. 742–746.

[94] Norman L. Johnson, Adrienne W. Kemp, and Samuel Kotz. *Univariate Discrete Distributions*. 3rd ed. Wiley Series in Probability and Statistics. John Wiley & Sons, 2005.

[95] Norman L. Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous Univariate Distributions, Volume 1*. 2nd ed. Wiley Series in Probability and Statistics. John Wiley & Sons, 1994.

[96] Serdar Kadioglu, Meinolf Sellmann, and Markus Wagner. "Learning a Reactive Restart Strategy to Improve Stochastic Search." In: *Learning and Intelligent Optimization*. Cham: Springer International Publishing, 2017, pp. 109–123.

[97] Richard M. Karp. "Reducibility among combinatorial problems." In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[98] Henry Kautz, Eric Horvitz, Yongshao Ruan, Carla P. Gomes, and Bart Selman. "Dynamic Restart Policies." In: *Eighteenth national conference on Artificial intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 674–681.

[99] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *arXiv:1412.6980 [cs]* (2014). arXiv: 1412.6980.

[100] Christian Kleiber and Samuel Kotz. *Statistical size distributions in economics and actuarial sciences*. Vol. 470. John Wiley & Sons, 2003.

[101] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, 2014.

[102] Ker-I Ko. *Complexity Theory of Real Functions*. Birkhäuser Boston Inc., 1991.

[103] Wolfgang König. *Stochastische Algorithmen*. Universität zu Köln, lecture notes (in German). available at http://www.wias-berlin.de/people/koenig/www/AlgStoch.pdf. 2003.

[104] Samuel Kotz and Saralees Nadarajah. *Extreme value distributions: theory and applications*. World Scientific, 2000.

[105] Donald L. Kramer and Robert L. McLaughlin. "The behavioral ecology of intermittent locomotion." In: *American Zoologist* 41.2 (2001), pp. 137–153.

[106] Mark W. Krentel. "The complexity of optimization problems." In: *Journal of Computer and System Sciences* 36.3 (1988), pp. 490–509.

[107] Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, Matthias Steinbrecher, Frank Klawonn, and Christian Moewes. *Computational intelligence*. Springer, 2011.

[108] Chin Diew Lai and Min Xie. *Stochastic ageing and dependence for reliability*. Springer Science & Business Media, 2006.

[109] Chunxiao Li, Noah Fleming, Marc Vinyals, Toniann Pitassi, and Vijay Ganesh. "Towards a Complexity-Theoretic Understanding of Restarts in SAT Solvers." In: *Theory and Applications of Satisfiability Testing*. Cham: Springer International Publishing, 2020, pp. 233–249.

[110] Jan-Hendrik Lorenz. "Runtime Distributions and Criteria for Restarts." In: *SOFSEM 2018: Theory and Practice of Computer Science*. Springer International Publishing, 2018, pp. 493–507.

[111] Jan-Hendrik Lorenz. *The Potential of Restarts for ProbSAT on Instances with a Hidden Solution*. Supplementary data and source code. Version v1.0. The data and code is available at https://doi.org/10.5281/zenodo.4533804. Zenodo, 2021.

[112] Jan-Hendrik Lorenz and Julian Nickerl. "The Potential of Restarts for ProbSAT." In: *International Conference on Computer Aided Systems Theory*. Springer, 2019, pp. 352–360.

[113] Jan-Hendrik Lorenz and Julian Nickerl. "The Potential of Restarts for ProbSAT." In: *arXiv preprint arXiv:1904.11757* (2019).

[114] Jan-Hendrik Lorenz and Julian Nickerl. *The Potential of Restarts for ProbSAT on Uniform Instances*. Supplementary data and source code. Version v1.0. The data and code is available at https://doi.org/10.5281/zenodo.4533842. Zenodo, 2021.

[115] Jan-Hendrik Lorenz and Florian Wörz. *concealSATgen*. Source code. The code is available at https://github.com/FlorianWoerz/concealSATgen. 2020.

[116] Jan-Hendrik Lorenz and Florian Wörz. "On the Effect of Learned Clauses on Stochastic Local Search." In: *Theory and Applications of Satisfiability Testing*. Springer International Publishing, 2020, pp. 89–106.

[117] Jan-Hendrik Lorenz and Florian Wörz. *On the Effect of Learned Clauses on Stochastic Local Search*. Supplementary data. Version 1.0. The data is available at https://doi.org/10.5281/zenodo.3776052. Zenodo, 2020.

[118] Max O. Lorenz. "Methods of measuring the concentration of wealth." In: *Publications of the American statistical association* 9.70 (1905), pp. 209–219.

[119] Ilya Loshchilov and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." In: *International Conference on Learning Representations (ICLR) 2017 Conference Track*. 2017.

[120] Edward R. Love. "Some logarithm inequalities." In: *The Mathematical Gazette* 64.427 (1980), pp. 55–57.

[121] Miodrag Lovric. *International Encyclopedia of Statistical Science*. Springer, 2011.

[122] Michael Luby and Wolfgang Ertel. "Optimal parallelization of Las Vegas algorithms." In: *Annual Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science 775. Springer, 1994, pp. 461–474.

[123] Michael Luby, Alistair Sinclair, and David Zuckerman. "Optimal Speedup of Las Vegas Algorithms." In: *Information Processing Letters* 47.4 (1993), pp. 173–180.

[124] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[125] Ole J. Mengshoel, David C. Wilkins, and Dan Roth. "Initialization and Restart in Stochastic Local Search: Computing a Most Probable Explanation in Bayesian Networks." In: *IEEE Transactions on Knowledge and Data Engineering* 23.2 (2011), pp. 235–247.

[126] Stephan Mertens, Marc Mézard, and Riccardo Zecchina. "Threshold values of random *K*-SAT from the cavity method." In: *Random Structures & Algorithms* 28.3 (2006), pp. 340–373.

[127] Leonid Mirny. "Cell commuters avoid delays." In: *Nature Physics* 4.2 (2008), pp. 93–95.

[128] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. 2nd. USA: Cambridge University Press, 2017.

[129] Aad P. A. van Moorsel and Katinka Wolter. "Analysis and algorithms for restart." In: *Proceedings of the First International Conference on the Quantitative Evaluation of Systems*. Proceedings of the First International Conference on the Quantitative Evaluation of Systems. 2004, pp. 195–204.

[130] Aad P. A. van Moorsel and Katinka Wolter. "Meeting Deadlines through Restart." In: *MMB & PGTS 2004, 12th GI/ITG Conference on Measuring and Evaluation of Computer and Communication Systems (MMB) together with 3rd Polish-German Teletraffic Symposium (PGTS)*. VDE Verlag, 2004, pp. 155–160.

[131] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. "Chaff: Engineering an Efficient SAT Solver." In: *Proceedings of the 38th Design Automation Conference (DAC '01)*. 2001, pp. 530–535.

[132] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge university press, 1995.

[133] Pablo Muñoz, David F. Barrero, and María D. R-Moreno. "Run-time analysis of classical path-planning algorithms." In: *Research and Development in Intelligent Systems XXIX*. Springer, 2012, pp. 137–148.

[134] Jayakrishnan Nair, Adam Wierman, and Bert Zwart. "The Fundamentals of Heavy Tails: Properties, Emergence, and Estimation." Preprint, California Institute of Technology. 2020.

[135] W. John O'Brien, Howard I. Browman, and Barbara I. Evans. "Search Strategies of Foraging Animals." In: *American Scientist* 78.2 (1990), pp. 152–160.

[136] Chanseok Oh. "Between SAT and UNSAT: the fundamental difference in CDCL SAT." In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2015, pp. 307–323.

[137] Makoto Okada, Kenji Yamanishi, and Naoki Masuda. "Long-tailed distributions of inter-event times as mixtures of exponential distributions." In: *Royal Society open science* 7.2 (2020), p. 191643.

[138] Arnab Pal and Shlomi Reuveni. "First Passage under Restart." In: *Physical Review Letters* 118.3 (2017), p. 030603.

[139] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Pub. Co., 1994.

[140] F. G. Parsons and P. H. Wirsching. "A Kolmogorov-Smirnov goodness-of-fit test for the two-parameter weibull distribution when the parameters are estimated from the data." In: *Microelectronics Reliability* 22.2 (1982), pp. 163–167.

[141] Vern Paxson, Mark Allman, Jerry Chu, and Matt Sargent. *Computing TCP's retransmission timer*. Tech. rep. 2011.

[142] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[143] James Pickands. "Statistical Inference Using Extreme Order Statistics." In: *The Annals of Statistics* 3.1 (1975), pp. 119–131.

[144] Knot Pipatsrisawat and Adnan Darwiche. "On the power of clause-learning SAT solvers as resolution engines." In: *Artificial Intelligence* 175.2 (2011), pp. 512–525.

[145] Jim Pitman. *Probability*. Springer, New York, 1993.

[146] Elijah Polak. *Optimization: algorithms and consistent approximations*. Vol. 124. Springer Science & Business Media, 2012.

[147] Marvin Rausand, Anne Barros, and Arnljot Hoyland. *System reliability theory: models, statistical methods, and applications*. 2nd ed. John Wiley & Sons, 2003.

[148] Sidney I. Resnick. *A probability path*. Springer, 2003.

[149] Horst Rinne. *The Weibull Distribution: A Handbook*. CRC press, 2008.

[150] Édgar Roldán, Ana Lisica, Daniel Sánchez-Taltavull, and Stephan W. Grill. "Stochastic resetting in backtrack recovery by RNA polymerases." In: *Physical Review E* 93 (6 2016), p. 062411.

[151] Sheldon M. Ross. *Introduction to Probability Models*. 12th. Los Angeles, CA, USA: Academic Press, 2019.

[152] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[153] Walter Rudin. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York, 1964.

[154] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. 3rd ed. Pearson, 2009.

[155] Uwe Schöning. "A probabilistic algorithm for k-SAT and constraint satisfaction problems." In: *40th Annual Symposium on Foundations of Computer Science*. 1999, pp. 410–414.

[156] Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*. Vol. 3. Lehmanns media, 2013.

[157] Bart Selman, David G. Mitchell, and Hector J. Levesque. "Generating hard satisfiability problems." In: *Artificial Intelligence* 81.1 (1996), pp. 17–29.

[158] Claude E. Shannon. "Mathematical theory of the differential analyzer." In: *Journal of Mathematics and Physics* 20.1-4 (1941), pp. 337–354.

[159] Oleg V. Shylo, Timothy Middelkoop, and Panos M. Pardalos. "Restart strategies in optimization: parallel and serial cases." In: *Parallel Computing* 37.1 (Jan. 2011). Elsevier, 2011, pp. 60–68.

[160] Oleg V. Shylo, Oleg A. Prokopyev, and Jayant Rajgopal. "On algorithm portfolios and restart strategies." In: *Operations Research Letters* 39.1 (Jan. 2011). Elsevier, 2011, pp. 49–52.

[161] João P. Marques Silva and Karem A. Sakallah. "GRASP—a New Search Algorithm for Satisfiability." In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*. ICCAD '96. San Jose, California, USA: IEEE Computer Society, 1996, pp. 220–227.

[162] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

[163] Jerome Spanier and Keith B. Oldham. *An atlas of functions*. Hemisphere publishing corporation, 1987.

[164] Terence Tao. *An introduction to measure theory*. Vol. 126. American Mathematical Society Providence, RI, 2011.

[165] Seinosuke Toda. "The complexity of finding medians." In: *31st Annual Symposium on Foundations of Computer Science*. IEEE, 1990, pp. 778–787.

[166] Seinosuke Toda. "PP is as hard as the polynomial-time hierarchy." In: *SIAM Journal on Computing* 20.5 (1991), pp. 865–877.

[167] Charlotte Truchet, Alejandro Arbelaez, Florian Richoux, and Philippe Codognet. "Estimating parallel runtimes for randomized algorithms in constraint solving." In: *Journal of Heuristics* 22.4 (2016), pp. 613–648.

[168] Leslie G. Valiant. "The complexity of computing the permanent." In: *Theoretical Computer Science* 8.2 (1979), pp. 189–201.

[169] Leslie G. Valiant and Vijay V. Vazirani. "NP is as easy as detecting unique solutions." In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1985, pp. 458–463.

[170] Aad P. A. Van Moorsel and Katinka Wolter. "Analysis of restart mechanisms in software systems." In: *IEEE Transactions on Software Engineering* 32.8 (2006), pp. 547–558.

[171] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2003.

[172] Giuseppe Vitali. *Sul problema della misura dei Gruppi di punti di una retta: Nota*. Tip. Gamberini e Parmeggiani, 1905.

[173] Gunnar Völkel, Ludwig Lausser, Florian Schmid, Johann M. Kraus, and Hans A. Kestler. "Sputnik: ad hoc distributed computation." In: *Bioinformatics* 31.8 (2014), pp. 1298–1301.

[174] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Berlin Heidelberg: Springer-Verlag, 1999.

[175] Abraham Wald. *Sequential analysis*. Courier Corporation, 2004.

[176] Toby Walsh. "Search in a Small World." In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. IJCAI '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 1172–1177.

[177] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[178] Frank Wilcoxon. "Individual Comparisons by Ranking Methods." In: *Biometrics* 1.6 (1945), pp. 80–83.

[179] Robert L. Winkler, Gary M. Roodman, and Robert R. Britney. "The determination of partial moments." In: *Management Science* 19.3 (1972), pp. 290–296.

[180] Katinka Wolter. *Stochastic Models for Fault Tolerance: Restart, Rejuvenation and Checkpointing*. Springer Science & Business Media, 2010.

[181] Huayue Wu. "Randomization and restart strategies." Master's thesis. University of Waterloo, 2006.

[182] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Features for SAT." In: *University of British Columbia, Tech. Rep* (2012).

[183] Lin Xu, Frank Hutter, Jonathan Shen, Holger H. Hoos, and Kevin Leyton-Brown. "SATzilla2012: Improved Algorithm Selection Based on Cost-sensitive Classification Models." In: *Proceedings of SAT Challenge* (2012), pp. 57–58.

[184] Aolong Zha. "GluHack." In: *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Vol. B-2018-1. Department of Computer Science Series of Publications B, University of Helsinki, 2018, p. 26.

INDEX

APPENDIX

A

CONSTRUCTION OF DECISION TREES

The following description of how to construct a decision tree is mainly taken from [81]. In general, the construction of a decision tree is performed using a greedy approach. For this purpose, for an inner node, a partition

$$R_1(j,s) = \{X \mid X_j \leq s\} \text{ and } R_2(j,s) = \{X \mid X_j > s\}$$

is chosen such that the loss function is minimized. For example, for regression problems, the squared error or a related size is typically chosen as the loss function. This results in the following minimization condition for an internal node

$$\min_{j,s} \left(\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_1)^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_2)^2 \right), \tag{A.1}$$

where $\hat{y}_1$ is the mean of the target variables y in $R_1(j,s)$. Likewise, $\hat{y}_2$ is the mean of the target variables y in $R_2(j,s)$.

On the other hand, when considering a classification problem, the squared error function is not suitable as the loss function. Instead, a function that accurately reflects the classification error has to be chosen. Suppose there are K classes to distinguish. Also, suppose that a decision tree with M leaves is considered. These leaves induce a partition $R_1, \ldots, R_M$ of the observations. The proportion of the k-th class in the m-th node is denoted by $\hat{p}_{m,k}$, where $\hat{p}_{m,k}$ can be calculated as follows

$$\hat{p}_{m,k} = \frac{1}{|R_m|} \sum_{y_i \in R_m} I(y_i = k),$$

where I is the indicator function. From this, a so-called impurity measure can be derived. We use the entropy Q_m for this purpose:

$$Q_m = -\sum_{k=1}^{K} \hat{p}_{m,k} \cdot \log \hat{p}_{m,k}.$$

The entropy provides an analogous minimization objective, similar to Equation (A.1). Thus, decision trees for classification problems can be constructed algorithmically. In general, decision trees have additional pruning mechanisms to counteract overfitting. However, since we are dealing with random forests, these mechanisms are omitted here.

B

ADAM AND THE BACKPROPAGATION ALGORITHM

In Section 6.3.2, we described how an existing neural network can be evaluated. The objective, however, is to find a suitable network for the task at hand. For this purpose, training algorithms like gradient descent are used. In this appendix, we describe a method for training a neural network. Some parts are repetitions from Chapter 6, which only serves the purpose of keeping this appendix reasonably self-contained.

The general procedure is as follows. A network structure is fixed, i. e., the number of nodes in each layer is determined, and a fully connected network is used. A training algorithm does not change this structure; the algorithm can adjust only the edge weights and the biases. For our purposes, we have chosen the training algorithm ADAM [99]. The fundamental concept behind this algorithm is to first calculate the gradient for the training data at hand. This calculation yields an estimate of the mean and the second moment of the gradient. The network's weights are then adjusted using an adequate ratio of the mean and the second moment. The underlying reasoning is that this way, the learning process avoids stagnation in plateaus of the loss function while not making too large changes in steep areas. The algorithm is presented in detail in Algorithm B.1 as pseudocode.

In this algorithm, $NN(\cdot;\theta)$ denotes the neural network's output if the current parameters (the weights and the biases) are θ. As can be seen in lines 8 and 9, the estimates for the mean and second moments are calculated by an exponential moving average. Since the two corresponding variables are initialized to 0, this calculation is biased, especially in the first iterations. The terms for bias correction (lines 10 and 11) are derived and described in [99].

In line 7, the gradient of the loss function is calculated with respect to the parameters. In our case, the edge weights of the network and the biases are the parameters of interest. This calculation is a non-trivial but ultimately solved problem. As an example, the backpropagation algorithm, which is presented in the following, can be used for this calculation. The presentation of the backpropagation algorithm is based on [78]. Backpropagation is a technique that is used to calculate the gradient. In the following, the loss function L is assumed to be differentiable. Furthermore,

Algorithm B.1 Pseudocode for the ADAM optimizer. The pseudocode is an adapted version from [99]. The term dg^2 in line 9 refers to an elementwise square operation.

Require: α: learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: decay for exponential moving average
Require: $\varepsilon \in \mathbb{R}_+$: term for avoiding numerical issues
Require: $\theta_0 = (W_0, b_0)$: initial parameters. In our case, the weights and the biases are the parameters.
Require: L, the loss function
1: $m \leftarrow 0$, a biased estimate of the mean is contained in m
2: $s \leftarrow 0$, a biased estimate of the second moment is contained in s
3: $t \leftarrow 0$, a step counter
4: **while** θ_t is not converged **do**
5: $(x, y) \leftarrow \left(\{x^{(1)}, \ldots, x^{(n)}\}, \{y^{(1)}, \ldots, y^{(n)}\}\right)$, sample of size n from the training set
6: $t \leftarrow t + 1$
7: $dg \leftarrow \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta L \left(NN(x^{(i)}; \theta_{t-1}), y^{(i)}\right)$, the gradient of the loss L w. r. t. the parameters θ, evaluated at θ_{t-1}
8: $m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot dg$, update biased estimate of the mean
9: $s \leftarrow \beta_2 \cdot s + (1 - \beta_2) \cdot dg^2$, update biased estimate of the second moment
10: $\hat{m} \leftarrow m / (1 - \beta_1^t)$, bias correction for the mean
11: $\hat{s} \leftarrow s / (1 - \beta_2^t)$, bias correction for the second moment
12: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m} / (\sqrt{\hat{s}} + \varepsilon)$, update the parameters
13: **return** θ_t

the activation functions $f : \mathbb{R} \to \mathbb{R}$ shall be non-linear and differentiable with derivative f'.

As an example, consider a neural network with one hidden layer and n_y output nodes. We intend to calculate the gradient of the loss function L with respect to the network's weights. First, we consider the j-th node's output in the k-th layer having activation function f. That is, if $k = 1$, then it is a hidden layer node; for $k = 2$, it is a node in the output layer.

We use the following expressions: $b_j^{(k)}$ is the bias of the node, $w_{ij}^{(k)}$ corresponds to the weight of the i-th node in the $(k-1)$-st layer to the j-th node in the k-th layer, furthermore $h_i^{(k-1)}$ denotes the output of the i-th node in the $(k-1)$-st layer. As an additional convention, $h_i^{(0)} = x_i$, where x_i is the input to the i-th input node. Let n_{k-1}

be the number of neurons in the $(k-1)$-st layer. Then, the output $h_j^{(k)}$ can then be calculated as follows:

$$a_j^{(k)} = b_j^{(k)} + \sum_{i=1}^{n_{k-1}} w_{ij}^{(k)} h_i^{(k-1)}$$
$$h_j^{(k)} = f\left(a_j^{(k)}\right).$$

By repeatedly applying the chain rule of calculus, one can now determine the derivative of the loss function L with respect to the weight $w_{ij}^{(k)}$. We distinguish the cases $k = 2$ and $k = 1$:

$$\frac{\partial L}{\partial w_{ij}^{(2)}} = \frac{\partial L}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial w_{ij}^{(2)}} = \frac{\partial L}{\partial \hat{y}_j} \cdot f'(a_j^{(2)}) \cdot h_i^{(1)} \tag{B.1}$$

$$\begin{aligned}\frac{\partial L}{\partial w_{ij}^{(1)}} &= \sum_{m=1}^{n_y} \frac{\partial L}{\partial \hat{y}_m} \cdot \frac{\partial \hat{y}_m}{\partial a_m^{(2)}} \cdot \frac{\partial a_m^{(2)}}{\partial h_j^{(1)}} \cdot \frac{\partial h_j^{(1)}}{\partial a_j^{(1)}} \cdot \frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}} \\ &= \frac{\partial h_j^{(1)}}{\partial a_j^{(1)}} \cdot \frac{\partial a_j^{(1)}}{\partial w_{ij}^{(1)}} \cdot \sum_{m=1}^{n_y} \underbrace{\frac{\partial L}{\partial \hat{y}_m} \cdot f'(a_m^{(2)})}_{\text{already calculated}} \cdot w_{jm}^{(2)}.\end{aligned} \tag{B.2}$$

In Equation (B.2), one expression is explicitly emphasized by sub-parentheses. As can be seen, this expression is also used in Equation (B.1) to calculate $\frac{\partial L}{\partial w_{ij}^{(2)}}$. In fact, this is not limited to networks having only one hidden layer. In general neural networks, the same partial calculations can also be used repeatedly. An approach using dynamic programming can exploit this fact. Furthermore, it is possible to vectorize the computational steps, both for the calculation of the outputs and for the gradient's determination. Among other things, this allows a concise representation of the code. A suitable procedure for calculating the output is shown in Algorithm B.2.

Algorithm B.2 Pseudocode for forward propagation. The code is an adapted version from [78]. In this version, regularization is not used.

Require: Neural network with depth ℓ.
Require: $W^{(i)}$, for $i \in \{1, \ldots, \ell\}$, the weights of layer i as a matrix
Require: $b^{(i)}$, for $i \in \{1, \ldots, \ell\}$, the biases of layer i as a vector
Require: x, the input (i. e., the training observations) as a vector
Require: y, the corresponding target outputs as a vector

$h^{(0)} \leftarrow x$
for $k \leftarrow 1$ to ℓ **do**
    $a^{(k)} \leftarrow b^{(k)} + W^{(k)} \cdot h^{(k-1)}$
    $h^{(k)} \leftarrow f(a^{(k)})$, here f is applied elementwise
$\hat{y} \leftarrow h^{(\ell)}$
$J \leftarrow L(\hat{y}, y)$

A method for calculating the gradient of the loss function is presented in Algorithm B.3. Here, it is assumed that Algorithm B.2 has been executed beforehand, and thus the calculated intermediate results can be used. In the algorithm, the notation $\nabla_x f$ is used, where $x = (x_1, x_2, \ldots, x_k)$ is a vector and f is a differentiable function. Then, $\nabla_x f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_k}\right)$. An analogous situation applies to $\nabla_W f$, where W is a matrix.

Algorithm B.3 Pseudocode for backward propagation. The code is taken from [78]. In this version, regularization is not used.

$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y) = \left(\frac{\partial L(\hat{y},y)}{\partial \hat{y}_1}, \frac{\partial L(\hat{y},y)}{\partial \hat{y}_2}, \frac{\partial L(\hat{y},y)}{\partial \hat{y}_3}, \ldots\right)$
for $k \leftarrow \ell$ downto 1 **do**
    $g \leftarrow g \odot f'(a^{(k)})$, where $\odot$ is the elementwise multiplication
    $\nabla_{b^{(k)}} J \leftarrow g$
    $\nabla_{W^{(k)}} J \leftarrow g h^{(k-1)\top}$, where $\top$ denotes the transpose operation.
    $g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)\top} g$

Overall, these two algorithms can effectively determine the gradient of the loss function with respect to the bias values and weights. These can thus be used to improve the network, for example, with the ADAM algorithm.

C

SPECIFICATION OF THE MACHINE LEARNING COMPONENTS

Parts of this appendix appeared in the following publication:

Jan-Hendrik Lorenz and Julian Nickerl. "The Potential of Restarts for ProbSAT." In: *arXiv preprint arXiv:1904.11757* (2019).

C.1 SPECIFICATION FOR UNIFORM INSTANCES

This section provides details regarding the machine learning pipeline for uniform instances.

C.1.1 *Features for Uniform Instances*

All components use 34 features. The neural networks use them in a normalized form with mean 0 and variance 1. A description of the features can be found in [182].

The following are the features grouped based on [182]. The number in brackets indicates the number of corresponding features.

- Problem Size Features:
  (2) Number of variables and clauses in the original formula
  (2) Number of variables and clauses after simplification

- Variable-Clause Graph Features:
  (3) Variable node degree statistics (mean, min, max)
  (3) Clause node degree statistics (mean, min, max)

- Variable Graph Features:
  (3) Node degree statistics (mean, min, max)

- Clause Graph Features:
  (2) Node degree statistics (mean, max)

- Proximity to Horn Formula:
  (2) Number of occurrences in a Horn clause for each variable (mean, min)

- DPLL Probing Features:
  (1) Search space size estimate

- Local Search Probing Features, based on 2 seconds of both SAPS and GSAT:
  (10) Number of steps to the best local minimum in a run
  (2) Average improvement to best in a run
  (3) Fraction of improvement due to first local minimum

- Timing Features:
  (1) CPU time required for feature computation

The internal names of the features are:

```
nvarsOrig, nclausesOrig, nvars, nclauses, VCG-CLAUSE-mean, VCG-CLAUSE-min,
VCG-CLAUSE-max, VCG-VAR-mean, VCG-VAR-min, VCG-VAR-max, HORNY-VAR-mean,
HORNY-VAR-min, VG-mean, VG-min, VG-max, CG-mean, CG-max, CG-featuretime,
saps_BestSolution_Mean, saps_BestSolution_CoeffVariance,
saps_FirstLocalMinStep_Mean, saps_FirstLocalMinStep_CoeffVariance,
saps_FirstLocalMinStep_Median, saps_FirstLocalMinStep_Q.10,
```

`saps_FirstLocalMinStep_Q.90, saps_BestAvgImprovement_Mean,`

`gsat_BestSolution_Mean, gsat_FirstLocalMinStep_Mean,`

`gsat_FirstLocalMinStep_CoeffVariance, gsat_FirstLocalMinStep_Median,`

`gsat_FirstLocalMinStep_Q.10, gsat_FirstLocalMinStep_Q.90,`

`gsat_BestAvgImprovement_Mean, lobjois-mean-depth-over-vars`

Thirty of the features were selected because they carry a minimum amount of variance. In addition, four more features were hand-selected. The internal names of these features are: `VCG-VAR-max`, `VG-max`, `CG-max`, `saps_BestSolution_CoeffVariance`.

C.1.2 *Specifications and Validation of the Random Forest*

The random forest consists of 50 trees and uses cross-entropy as an impurity measure; all other parameters were the default values used in SCIKIT-LEARN [142].

Validating the random forest by ten iterations of a 10-fold cross validation resulted in an average potential speedup of 1.218. Evaluating the final model on an independent set of 162 instances resulted in a speedup of 1.138. The evaluation of the forests on other metrics is displayed in Table C.1. All values concern the Weibull distribution as the label.

metric	cross validation	final model
true positive rate	0.803	0.750
true negative rate	0.648	0.571
precision	0.764	0.697
negative prediction value	0.698	0.635
fall-out	0.351	0.429
miss rate	0.197	0.25
accuracy	0.738	0.673
balanced accuracy	0.726	0.661

Table C.1: The evaluation of the random forest for different metrics. All values concern the Weibull distribution as the label.

We were not able to improve these values significantly. Especially an accuracy of below 75% is not a desirable result. However, we used the random forest since

the potential speedup was still promising. An explanation for this is that often both lognormal and Weibull distributions describe the runtime distribution well and, therefore, produce good restart times. Hence, classifying the wrong distribution does not necessarily harm the potential speedup.

C.1.3 *Specification of the Neural Networks*

All networks used ADAM as the optimizer, with an initial learning rate 0.0005 and clipnorm 0.5. The inputs were processed in batches of 16.

The parameters of the networks are displayed in Table C.2. Regarding the number of output neurons: While the relevant number of neurons was two, we had to use three in the lognormal and four in the Weibull network for technical reasons. The reasons for this have already been discussed in the discussion of Table 6.4. To reiterate, only two parameters, the shape and scale, are of interest. However, three or, respectively, four labels were used for the calculation of the loss function. This was implemented by using three or, respectively, four output neurons.

C.2 FEATURES FOR THE RANDOM FOREST FOR HIDDEN SOLUTION INSTANCES

The random forest for the hidden solution instances uses 16 features. A description of the features can be found in [182].

The following are the features grouped based on [182]. The number in brackets indicates the number of corresponding features.

- Problem Size Features:
  - (2) Number of variables and clauses in the original formula
  - (2) Number of variables and clauses after simplification

- Local Search Probing Features, based on 2 seconds of both SAPS and GSAT:
  - (8) Number of steps to the best local minimum in a run
  - (2) Average improvement to best in a run
- Timing Features:

layer	parameter	location	Weibull	lognormal
input	neurons	34	34	34
	gaussian noise	0.01	0.01	0.01
hidden1	neurons	14	14	14
	activation	*tanh*	*tanh*	*tanh*
	l2 regularization	0.01	0.01	0.01
	batch normalization	*true*	*true*	*true*
	gaussian noise	0.12	0.12	0.12
	dropout	0.01	0.01	0.01
hidden2	neurons	1	7	7
	activation	*tanh*	*tanh*	*tanh*
	l2 regularization	0.01	0.01	0.01
	batch normalization	*true*	*true*	*true*
	gaussian noise	0.12	0.12	0.12
	dropout	0.01	0.01	0.01
output	neurons	1	2(4)	2(3)
	activation	*exp*	*sigmoid*	*sigmoid*
	l2 regularization	0.01	0.01	0.01

Table C.2: The specifications of the location, Weibull, and lognormal neural network, separated by layer.

(2) CPU time required for the SAPS and GSAT features

The internal names of the features are:

```
nvarsOrig, nclausesOrig, nvars, nclauses,

saps_BestSolution_Mean, saps_FirstLocalMinStep_Mean,

saps_FirstLocalMinStep_Median, saps_FirstLocalMinStep_Q.10,

saps_FirstLocalMinStep_Q.90, ls-saps-featuretime,

gsat_BestSolution_Mean, gsat_FirstLocalMinStep_Mean,

gsat_FirstLocalMinStep_Median, gsat_FirstLocalMinStep_Q.10,

gsat_FirstLocalMinStep_Q.90, ls-gsat-featuretime
```