

Preface

The Analysis of Algorithms is a core Computer Science area which provides information on the expected, i.e. the average-case, performance of algorithms. Such information is useful in a variety of applications, including power estimation and resource budgeting in a real-time context. The Analysis of Algorithms also provides fundamental insights in the design of efficient software. Hence, both from an applied and a theoretical perspective, the investigation of improved methods and tools for static average-case analysis is a worthwhile goal.

Average-Case Analysis involves a variety of techniques which, typically, do not allow for automation. Currently algorithms must be analyzed on a case-by-case basis and it is not feasible in general to statically derive the average number of basic steps carried out by an algorithm during its execution. Various bottle-neck problems have been high-lighted in the literature and some well-known algorithms escape analysis.

In view of the status of the field, the ultimate aim to provide a unified foundation for average-case analysis motivated the work of many authors including [Knu73, FS95, Ram96, Vui80]. As pointed out in [Vui80]:

A progress in our understanding of these questions should drastically affect the way in which we discover and explain the fundamental algorithms, as catalogued by Knuth [Knu73] and Aho et al [AHU87].

The aim of this work is to present a new approach to the Average-Case Analysis of Algorithms, based on the novel notion of random bags and their preservation. The view presented here is that the notion of a random bag may serve as a unifying model for abstract data structures and their data distribution, while random bag preservation enables the constructive tracking of the distribution during computations. The approach inspired novel algorithms and considerably simplified their average-case analysis.

The work presents a modular calculus for static average-case analysis which drastically simplifies the analysis and opens up the way for novel explorations on static timing tools. Random bags also contribute a visual way to represent data and their distributions, which, in addition to facilitating average-case analysis, provides a useful teaching aid.

A parallel between the role of Static Analysis in Software Engineering and the role of Calculus in “real” Engineering may be helpful to illustrate the motivation behind the research. Engineering offers the capacity to analyse the strength of a construction, such as a bridge, by analyzing its blue prints, rather than subjecting it to heavy loads to test its limits. This approach should ideally find a natural parallel in Software Engineering via Static Program Analysis. Rather than executing a program on a large selection of inputs to experimentally derive information on its average-case behaviour, the goal is to derive this information statically via an analysis of the program’s source code. Calculus supports the analysis of blue-prints in Engineering. Similarly, the aim of this work is to provide a foundation for a Calculus supporting Static Average-Case Analysis of a program’s source code. This is a major challenge and our aim is not to provide an all-encompassing answer. Instead, we focus on the introduction of new advances in this area as a basis for a simplified and unified theory of average-case analysis and as a potential platform on which to build future improved modular static analysis tools.

A central aspect of the novel approach, which distinguishes it from prior approaches to Average-Case Analysis, is the use of randomness preservation to ensure the compositionality property, well-known from the Semantics and the Real-Time Language areas, in the context of the Analysis of Algorithms. Compositionality can rightfully be referred to as the “golden key” to static analysis, witnessed by its central role in static worst-case time analysis. A main theme of this work is that compositionality, combined with the capacity for tracking data distributions, unlocks a novel technique for modular average-case analysis. This approach provides the inspiration for the \mathcal{MOQA} ¹ “language”. The language essentially consists of a suite of random bag preserving data-structuring operations together with conditionals, for-loops and recursion and hence can be incorporated in *any* traditional programming language, importing all of its benefits in a familiar context².

A key feature of \mathcal{MOQA} is that its operations have been purpose designed to ensure the capacity for a compositional static average-case analysis of \mathcal{MOQA} code. The guaranteed compositionality property of \mathcal{MOQA} programs brings a strong advantage for the programmer. The capacity to combine parts of code, where the average-time is simply the sum of the times of the parts, is a helpful advantage in static analysis. Moreover, re-use is a key factor in our approach: once the average time is determined for a piece of code, then this time will hold in any context. Hence it can be re-used and the timing impact is always the same. Compositionality also improves precision of static average-case analysis, supporting the determination of accurate estimates on the average number of basic operations of programs.

It is a main theme of the current work to introduce the new foundation for average-case analysis and to illustrate its applicability, as well as to motivate and specify the \mathcal{MOQA} language and discuss its associated static average-case timing tool *Dstri-Track*.

¹ \mathcal{MOQA} Modular Quantitative Analysis.

² \mathcal{MOQA} is implemented at CEOL in Java 5.0 as \mathcal{MOQA} -Java.

The work is carried out at the intersection of several areas: Analysis of Algorithms and Random Structures, Semantics, Real-Time Languages, Static Program Analysis, Modular Design and the mathematical theories of Finite Partial Orders, Linear Extensions, Multi-Sets and Probability Theory. Hence the material may be useful for a variety of researchers and students, with interests in Computer Science, Electrical Engineering or Mathematics.

We provide an overview of the chapters in this work.

Chapter 1 provides an introduction to the new techniques for average-case analysis and focuses on a motivation of the central notions involved. This includes a motivation of compositionality as the “golden key” to static timing and the need for novel language design to reach compositionality, including the related concept of an Efficiency-Oriented Language.

The chapter provides a brief introduction to the *MOQA* language, for which static average-case timing can be achieved in a modular way through the tracking of distributions. Random bags are introduced as concise ways to capture data and their distribution and distribution tracking is incorporated via the concept of random bag preservation. The split operation, well-known from algorithms such as Quicksort and Quickselect, is provided as an example of a random bag preserving operation. This example also serves to illustrate the tracking of distributions in *MOQA* and the use of the notion of a separative function to establish random bag preservation.

The chapter also discusses the central Linear-Compositionality Theorem, which forms the basis for the static derivation of the average-case time of *MOQA* programs. Advantages of the *MOQA* approach are outlined and the chapter concludes with a discussion of the related area of bridging Semantics and Complexity and the area of Real-Time Languages.

Chapter 2 presents introductory notions, including partial orders, series-parallel orders, trees, heaps and bags. A brief overview of some basic sorting algorithms is provided as well as an introduction to standard timing measures, including exact time, total time, worst-case, best-case and average-case time.

Chapter 3 introduces the central notion of compositionality, including IO-compositionality. Worst-case time is shown to be semi-IO-compositional while average-case time is shown to be IO-compositional. The Average-Case Time Paradox is discussed in this context. This paradox regards the fact that even though average-case time is shown to have better compositionality properties than worst-case time, in practice the derivation of average-case time is known to be much more difficult than worst-case time. The paradox is shown to be linked to the potential lack of randomness preservation of standard algorithms, including well-known examples such as Bubblesort and Heapsort. Moreover, the chapter motivates how IO-compositionality of the average-case time measure can be used, in combination with randomness preservation, to obtain linear-compositionality. This greatly facilitates average-case time analysis and overcomes the Average-Case Time Paradox.

Chapter 4 revisits in a slightly more general context, the fundamental notions of random structures, random bags and their preservation, which have been introduced in Chapter 1. The State Theorem is presented which enables an interpretation of

states in random structures as “generalized permutations”. Chapter 4 also introduces the central notion of an isolated subset. An isolated subset forms a subset of a partial order such that the restriction of the random structure over this partial order to the isolated subset is guaranteed to yield a new random structure. A simplified definition of an isolated subset is obtained for the case of series-parallel orders. The chapter concludes with the Extension Theorem, which demonstrates that it is sufficient to define random bag preserving operations locally on an isolated subset, where the extension of the operation to the entire random structure is obtained in a natural way.

Chapter 5 introduces the basic *MOQA* operations, including the Random Product, the Random Deletion and Percolation, the Random Projection, the Random Split and the Top and Bot operations. Each of these *MOQA* operations is shown to be random bag preserving. Deletion operations typically are not included in the context of automated average-case analysis, since the analysis of deletions with respect to average-case time is well-known to be problematic, even in the context of traditional average-case analysis. Hence the Random Deletion opens up the way for the inclusion of novel algorithms, such as Percolating Heapsort and Treapsort, which are analyzed in Chapter 9. The Extension Theorem of Chapter 4 is applied to extend these operations from local applications on isolated subsets to applications over the entire random structure. Uniformly random bag preserving operations are singled out as of particular interest, since this type of operations enables simplifications of probability computations in later chapters. The *MOQA* operations are shown to preserve series-parallel data structures which yields a characterization of the so-called *MOQA* atomic-constructible data structures as series-parallel orders. Finally, some simplifications for the series-parallel case are obtained in the context of the computation of cardinalities of random structures. Such simplifications for series-parallel orders will also be useful in the context of Chapter 6, which regards the average-case analysis of the basic *MOQA* operations.

Chapter 6, joint with D. Early, presents the detailed average-case analysis of the basic *MOQA* operations, resulting in the formulas obtained by D. Early. As shown in Chapter 7, *MOQA* programs are Linearly-Compositional with respect to the average-case time, i.e. their average-case time can be expressed as linear combinations of the average-case times of more basic components. Hence, ultimately, a successful average-case time derivation yields the average-case time of *MOQA* programs, expressed in terms of the average-case times of the basic *MOQA* operations. Formulas for the average-case times of basic *MOQA* operations are obtained in Chapter 6 and simplified formulas are derived for the case of series-parallel orders. These formulas are systematically applied in Chapter 9, which presents examples of compositional average-case time derivations of *MOQA* programs. Finally, the formulas of this chapter are illustrated via basic applications involving inductively defined data structures, such as linear orders and complete binary trees. Chapter 6 concludes with a demonstration of combinatorial identities used in the derivation of the average-case time formulas.

Chapter 7 provides the specifications for the *MOQA* language, with special attention given to conditionals and recursion, which typically form a challenge for static timing analysis. The random bag preservation of *MOQA* programs is demon-

strated and the method for the linear-compositional derivation of the average-time of \mathcal{MOQA} programs is outlined.

Chapter 8 provides examples of well-known sorting and search algorithms implemented in \mathcal{MOQA} . It also includes examples of two novel algorithms, Percolating Heapsort, the first randomness preserving version of the Heapsort algorithm, and Treapsort, a sorting algorithm over treaps; both of which are essentially based on the Random Deletion operation of Chapter 5.

Chapter 9 provides the compositional average-case time derivation of the programs discussed in Chapter 8, with a main focus on illustrating the use of random bags in this context. The chapter in particular presents the first exact average-case time analysis of a heapsort variant via an analysis of Percolating Heapsort. Compositional average-case time derivations, whenever appropriate, rely on the formulas obtained in Chapter 6. The derivations obtained in this chapter illustrate the basic techniques involved in the static timing tool *Distri-Track*.

Chapter 10, joint with D. Hickey and M. Boubekur, discusses in more detail the static timing tool *Distri-Track*, developed by D. Hickey. *Distri-Track* analyses \mathcal{MOQA} algorithms programmed in Java, using an implementation of \mathcal{MOQA} by J. Townley. *Distri-Track* enables the automated static derivation of average-case time of most of the \mathcal{MOQA} programs presented in Chapter 8. Experiments, including comparisons with time derivations relying on a Java profiler, are discussed, as well as potential implications for Real-Time Languages. Finally, Chapter 11 presents the conclusion and some potential future work.

The book is accompanied by a software tutorial “Static average-case analysis of programs: a beginner’s guide to successful tracking”. The tutorial requires Adobe Flash Player, which is freely available online at <http://www.adobe.com/>. The tutorial provides an introduction to the main concepts used in this work as well as videos illustrating the basic \mathcal{MOQA} operations and a selection of \mathcal{MOQA} programs. The reader is advised to read Chapters 1 and 3, followed by a viewing of the tutorial, before proceeding with later chapters in this work.

A Modular Calculus for the Average Cost of Data
Structuring

Schellekens, M.

2008, XXIV, 245 p., Hardcover

ISBN: 978-0-387-73383-8