TRACTABILITY

Classical computer science textbooks tell us that some problems are "hard". Yet many areas, from machine learning and computer vision, to theorem proving and software verification, have defined their own set of tools for effectively solving complex problems. *Tractability* provides an overview of these different techniques, and of the fundamental concepts and properties used to tame intractability.

This book will help you understand what to do when facing a hard computational problem. Can the problem be modelled by convex, or submodular functions? Will the instances arising in practice be of low treewidth, or exhibit another specific graph structure that makes them easy? Is it acceptable to use scalable, but approximate algorithms? A wide range of approaches is presented through self-contained chapters written by authoritative researchers on each topic. As a reference on a core problem in computer science, this book will appeal to theoreticians and practitioners alike.

LUCAS BORDEAUX is a Senior Research Software Development Engineer at Microsoft Research, where he works on the design and applications of algorithms to solve hard inference problems.

YOUSSEF HAMADI is a Senior Researcher at Microsoft Research. His work involves the practical resolution of large-scale real life problems set at the intersection of Optimization and Artificial Intelligence. His current research considers the design of complex systems based on multiple formalisms fed by different information channels which plan ahead and perform smart decisions. His current focus is on Autonomous Search, Parallel Search, and Propositional Satisfiability, with applications to Environmental Intelligence, Business Intelligence, and Software Verification.

PUSHMEET KOHLI is a Senior Research Scientist in the Machine Learning and Perception group at Microsoft Research. His research interests span the fields of Computer Vision, Machine Learning, Discrete Optimization, Game Theory, and Human-Computer Interaction with the overall aim of "teaching" computers to understand the behaviour and intent of human users, and to correctly interpret (or "see") objects and scenes depicted in colour/depth images or videos. In the context of tractability and optimization, Pushmeet has worked on developing adaptive combinatorial and message passing-based optimization algorithms that exploit the structure of large-scale optimization problems encountered in computer vision and machine learning to achieve improved performance.

TRACTABILITY

Edited by

LUCAS BORDEAUX
Microsoft Research

YOUSSEF HAMADI
Microsoft Research

PUSHMEET KOHLI
Microsoft Research

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107025196

First published 2014

Printed in the United Kingdom by Bell and Bain Ltd

A catalogue record for this publication is available from the British Library

ISBN 978-1-107-02519-6 Hardback

Contents

Contributors

Lucas Bordeaux *Microsoft Research, 21 Station Rd, Cambridge CB1 2FB, UK.*
lucasb@microsoft.com

Nikolaj Bjørner *Microsoft Research, One Microsoft Way Redmond, WA 98052, USA.*
nbjorner@microsoft.com

Adnan Darwiche *Computer Science Department, University of California Los Angeles, Los Angeles CA 90095-1596, USA.*
darwiche@cs.ucla.edu

Lara Dolecek *Department of Electrical Engineering, University of California Los Angeles, Los Angeles CA 90095-1594, USA.*
dolecek@ee.ucla.edu

Fedor V. Fomin *Department of Informatics, Universitetet i Bergen, Norway.*
fomin@ii.uib.no

Daniel Golovin *Google, Inc., Mountain View CA 94043, USA.*
golovin@gmail.com

Georg Gottlob *Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK.*
georg.gottlob@cs.ox.ac.uk

Gianluigi Greco *Department of Mathematics and Computer Science, Università degli Studi della Calabria, I-87036, Rende(CS), Italy.*
ggreco@mat.unical.it

Youssef Hamadi *Microsoft Research, 21 Station Rd, Cambridge CB1 2FB, UK.*
youssef@microsoft.com

Peter G. Jeavons *Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK.*
Peter.Jeavons@cs.ox.ac.uk

Tony Jebara *Department of Computer Science, Columbia University, New York, NY 10027, USA.*
jebara@cs.columbia.edu

Pushmeet Kohli *Microsoft Research, 21 Station Rd, Cambridge CB1 2FB, UK.*
pkohli@microsoft.com

Vladimir Kolmogorov *Institute of Science and Technology, Am Campus 1 3400 Klosterneuburg Austria.*
vladimir.kolmogorov@ist.ac.at

Andreas Krause *Department of Computer Science, Eidgenössische Technische Hochschule Zürich, Universitätstrasse 6, 8092 Zurich, Switzerland.*
krausea@ethz.ch

Ines Lynce *INESC-ID/IST, Technical University of Lisbon, Rua Alves Redol 9, 1000-029 Lisbon, Portugal.*
ines@sat.inesc-id.pt

Joao Marques-Silva *CASL/CSI, University College Dublin, Belfield, Dublin 4, Ireland.*
jpms@ucd.ie

Leonardo de Moura *Microsoft Research, One Microsoft Way Redmond, WA 98052, USA.*
Leonardo@microsoft.com

Saket Saurabh *Institute of Mathematical Sciences, CIT Campus, Taramani, 600 113 Chennai, India.*
saket@imsc.res.in

Francesco Scarcello *Department of Computer Science, Modelling, Electronics and Systems Engineering (DIMES), University of Calabria, I-87036, Rende(CS), Italy.*
scarcello@unical.it

Mohit Singh *Microsoft Research, One Microsoft Way Redmond, WA 98052, USA.*
mohits@microsoft.com

Suvrit Sra *Max Planck Institute for Intelligent Systems, 72076 Tübingen, Germany.*
suvrit@gmail.com

Kunal Talwar *Microsoft Research Silicon Valley, 1065 La Avenida Mountain View, CA, 94043, USA.*
kunal@microsoft.com

Stanislav Živný *Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK*
standa@cs.ox.ac.uk

Introduction

Lucas Bordeaux, Youssef Hamadi, Pushmeet Kohli

In mathematics and computer science, optimization is the process of finding the best solution from a set of alternatives that satisfy some constraints. Many applications in allied fields of computer science like machine learning, computer vision, bioinformatics, involve the solution of an optimization problem. For instance, optimization is used to schedule trains and airplanes, allocate the advertisements we see on television or in connection with internet search results, find the optimal placement of sensors to detect and neutralize security threats, or even to make decisions on what is the best way to perform medical surgery on a patient.

Optimization problems are generally hard to solve – their solution may involve exhaustively searching over a set of solutions whose size could increase exponentially with the number of variables whose values we may want to infer. That said, in practice, many of these problems can often be solved with remarkable efficiency. This is usually done by dedicated techniques, developed in each and every application domain, that exploit the "properties" of the problems encountered in practice.

Over the last few decades, researchers working in a number of different disciplines have tried to solve optimization problems that are encountered in their respective fields by exploiting some structure or properties inherent in the problems. In some cases, they have been able to isolate classes of optimization problems that can be solved optimally in time polynomial in the number of variables, while in other cases, they have been able to develop efficient algorithms that can produce solutions that, although not optimal, are good enough. For instance, researchers working on computer vision problems have developed a range of techniques to solve very large scale optimization problems that arise while labelling impage pixels. Similar sets of techniques, tools and fundamental results have been developed by

experts in machine learning, operations research, hardware/software verification, and other fields.

The goal of the proposed book is to bring together contributors from these various fields, to reflect on the state-of-the-art techniques for solving combinatorial optimization problems in these fields, and offer a multi-disciplinary understanding of the following question: which properties make hard computational problems solvable ("tractable") in practice?

The coverage of the contributions range from theory to practice, from software verification to computer vision. The researchers and practitioners in these areas work on apparently very different problems, but they are brought together by the common question:

How do we develop efficient algorithms for solving optimization problems?

The contributors to this book answer this question by identifying properties that make certain problems tractable or that allow the use of certain heuristics that can find solutions that are close to the optimal solution. In this introduction we aim to put the theme of tractability in perspective, and to provide a road map of the chapters.

Background

We conceived the book as a follow-up to a successful workshop organized at Microsoft Research, Cambridge, in July 2010[1]. The participants of this workshop were researchers working on solving optimization problems in diverse fields who had made important contributions to the analysis of tractability in their areas. The feedback we received from the participants revealed one main message: everyone was able to learn something new. Despite their extensive expertise, seeing the quest for tractability cast in a different light in a related field opened many opportunities for cross-fertilization and for a more unified view of the fundamentals.

It was this very positive feedback that prompted us to envision this volume, as an up-to-date and in-depth resource, built on multiple points of view coming from several communities.

Optimization Problems

Constraint satisfaction and optimization problems can be formulated in a number of different ways. We consider the following general description of optimization problems that can be described in terms of:

[1] http://research.microsoft.com/tractability2010/

- A set of a variables, $X = \{x_1, \ldots, x_n\}$;
- An objective function that is broken down into a number of terms $F = \{f_1, \ldots, f_m\}$, each of which is applied to a subset of variables $x_i \subseteq X$.

Specific problems make this high-level definition concrete by fully specifying:

- The range of the variables;
- The form allowed for the functions f_i;
- The way to aggregate the values of all of these functions.

For instance, the propositional satisfiability (SAT) problem is a special case where the variables range over $\{0, 1\}$, and the functions are Boolean constraints (usually clauses, as described in Chapter 12) that return 1 when satisfied and 0 otherwise. These constraints are aggregated as a sum, which we aim to maximize, which is one of several possible ways to indicate that we want to satisfy all constraints.

In other problems, the variables may range over arbitrary finite sets, or over the real numbers, or they may be logical variables in a language richer than the aforementioned (propositional) SAT framework, for instance the full first-order logic language. The terms in F can take diverse forms and they are usually aggregated into a global objective by summation or product[2].

While the framework described above is extremely abstract and general, it is important to understand its precise boundaries. In these formulations, a user, or domain expert, is required to translate the requirements and costs of the problem at hand into numerical functions, that can in turn be optimized using computational tools. In contrast, there is an important class of "black-box" optimization problems, where an analytical form of the objective function is not available, or cannot be written down mathematically. Instead, the problem is described by means of a black box, to which we can pass candidate solutions one by one, and that returns their quality. Optimizing functions described in such a black-box fashion require a different set of tools that exploit properties of the functions.

There is one important tool surveyed in this book that applies to black-box settings: submodularity. If we know that the function is submodular, then we can optimize it even if it is only available in a black-box form. When we cannot make this assumption, black-box problems fall out of the scope of this collection, because their intractability is due to *information-theoretic* reasons: what makes black-box settings intractable is that in many cases we simply do not know the function until we have queried the value of each

[2] Some communities focus by convention on products, which is justified by the observation that summations and products can be interchanged using the identity $\log(a \cdot b) = \log a + \log b$. In this case the graphical models are often referred to as *factor graphs*.

and every of its possible inputs. In contrast, the intractability of the model-based optimization problems we focus on is purely *complexity-theoretic*: the information about the function is all there, yet finding its optimal input may be computationally hard.

Graphical versus Language Properties

An important consequence of the framework previously described, is that there are two types of properties that can make problems tractable:

- *Graphical Properties* are properties of the (hyper-)graph (X, F). Here the functions $f \in F$ are seen purely as hyper-edges, i.e. in terms of what variables they connect, what patterns are found in the layout of these connections. The internal definition of these functions is here ignored. In some problems the graph has for instance a shape that is "tree-like" in a certain, formally-defined sense; in some others it belongs to a specific class such as *perfect* graphs. It is often possible to design algorithms that exploit these properties to solve the optimization problem more efficiently.
- What we call *Language Restrictions* are internal properties of the functions $f \in F$. These functions constitute building blocks, a "language" in which problems are described. If arbitrary functions are allowed, we have a language in which large classes of problems can be expressed, but that is in general intractable. Restricting carefully the types of functions available in this language can in a sense provide a tractability safeguard, preventing the user or modeller from writing down arbitrarily hard problems. The goal here is to find sweet spots: the interesting languages are those that are simple enough to avoid the intractability pitfall, yet expressive enough to capture interesting problems.

Types of Tractability

Graphical properties and language restrictions give rise, in fact, to several types of tractability. The goal is typically to use an algorithm whose runtime, as measured as a function of the input length, can be formally bounded[3]. This bound can be of different types:

Polynomial time guarantees: If the problem exhibits a certain graphical property, or is expressed within certain language restrictions, we can

[3] In this book we typically aim to bound the runtime *in the worst case*. One can sometimes refine this bound by analysing, further, the runtime *in expectation*, for certain distributions of problems.

sometimes determine that the time needed to solve it to optimality is bounded by a polynomial of the input length. This represents, in a sense, the ideal case: we have identified a tractable problem.

Controlled combinatorial explosion: In some cases there is no clear polynomial time guarantee, but we can identify a parameter p that is key in estimating problem hardness. This parameter will typically have an intractably high value in the worst case, but lower values in instances met in reality. For instance, problems with n variables might in fact have a "true dimensionality" that is significantly lower than n in the instances we are considering. The complexity is "exponential only" in the reduced dimension, which can be acceptable when this parameter is small enough. We can sometimes, as another example, apply a pre-processing technique that translates the problem into a form for which we know polynomial time algorithms. Such a translation typically blows-up in the worst case and the key parameter here is the size of the output of the pre-processing.

There remain, unfortunately, problems for which we cannot determine any reasonable bound on the runtime. These problems might still be solved in some cases, using the following approaches:

Approximations: Rather than looking for a global optimum to a model that is, after all, only an approximation of reality, it is sometimes acceptable to compute solutions that are only "close enough" to this optimum, or to solve a slightly different model that is tractable, provided that it does not deviate too much from the original model.

Heuristics: While complete algorithms have an exponential runtime in the worst case, the instances for which this worst case provably occurs are often "pathological" or "unnatural" problems: for instance artifically constructed, or completely random, problems that are unlikely to be found in practice. This is therefore a grey area in the resolution of intractable problems: we have an algorithm that has no provable runtime guarantee, and a problem that is not proved, or believed, to be hard. A common practice is to apply the algorithm to the problem and see whether it converges in an acceptable time.

Outline

We have divided the book into five parts:

- Part 1 focusses on fundamental concepts and techniques for graphical structure;

- Part 2 considers fundamental concepts and techniques for language restrictions;
- Part 3 looks at techniques and efficient algorithms for exact or approximate solutions;
- Part 4 focusses on how tractability is approached in different fields in computational sciences;
- Part 5 deals with heuristics.

We start with two chapters on fundamental concepts and techniques related to the two big classes of tractability properties we have mentioned: graphical structure and language restrictions.

Part 1: Graphical Structure

Part 1 starts with a chapter written by Georg Gottlob, Gianluigi Greco, and Francesco Scarcello who show how a number of optimization problems defined over tree (or *hyper-tree*) structures are tractable. Hyper-tree Width, which measures how "tree-like" the graphical model is, is identified as a important parameter for understanding graphical structure.

This is followed by a chapter by Tony Jebara that shows that a variety of hard problems such as graph colouring, maximum clique and maximum stable set are all solvable in polynomial time when the input graph is restricted to be a *perfect graph*.

Part 2: Language Restrictions

Part 2 starts with the chapter by Andreas Krause and Daniel Golovin on submodular function maximization. Submodular functions are a prime example of tractable language: restricting oneself to such functions when modelling a problem offers many tractability guarantees, for instance maximization problems can be approximated with provable bounds using simple and scalable heuristics.

This is followed by a chapter written by Peter G. Jeavons and Stanislav Živný that discusses the family of optimization problems that can be cast as valued constraint satisfaction problems (VCSPs) and provides examples of tractable cases.

Part 2 concludes with a chapter written by Adnan Darwiche which reviews a set of tractable knowledge representation formalisms. A key concept here is that of *Decomposable Negation Normal Forms (DNNF)* for Boolean formulae, a fairly broad class that supersedes, in particular, the widely used

Ordered Binary Decision Diagrams (OBDDs). Formulae that respect this syntactical restriction offer a number of tractability guarantees. This chapter illustrates the notion of tractability as "controlled combinatorial explosion" discussed above: logical formulae can be "compiled" into specific forms such as DNNF: this compilation is in the worst case itself intractable, but turns out to be feasible in interesting cases.

Part 3: Algorithms and their Analysis

Part 3 presents a number of algorithms and techniques that have been proposed to solve optimization problems in different disciplines.

It starts with a chapter from Vladimir Kolmogorov which describes the *tree-reweighted message passing* algorithm that is popularly used for inference of the Maximum a Posteriori (MAP) solutions in probabilistic graphical models.

This is followed by a chapter from Suvrit Sra that provides an overview of the methods used for solving *convex optimization* problems encountered in machine learning. This chapter considers optimization problems that are defined over continuous variables where the property of convexity plays a central role.

We then have a chapter from Mohit Singh and Kunal Talwar that describes techniques that have been developed in the design of *approximation algorithms*. The study of these algorithms is also a broad area in its own right, one that borrows concepts from all other tractability approaches. The chapter gives an overview of the rich possibilities, and also well-understood limitations, of these algorithms.

Part 3 ends with a chapter, written by Fedor V. Fomin and Saket Saurabh, that addresses the topic of kernelization methods for Fixed-Parametrized Complexity. Fixed Parameter Complexity offers a general perspective on the idea, presented above, of identifying key parameters that provide bounds on problem hardness. The chapter focusses specifically on kernelization methods, which provide a systematic study of pre-processing algorithms.

Part 4: Tractability in Some Specific Areas

This part deals with application of optimization algorithms in different application areas and mentions the latest results in the field.

We have a chapter by Pushmeet Kohli that describes the role played by submodular function minimization solvers (and in particular graph cuts), in

the field of computer vision, for solving image labelling problems such as image segmentation.

In the final chapter of this part, Lara Dolecek writes about some applications of tractability concepts to coding theory. She shows how some techniques originating from graphical models play a major role in the success of Low Density Parity Codes. Decoding is in this context a seemingly complex task that is in practice handled with surprising efficiency. The author shares recent insights and explanations of this good behaviour.

Part 5: Heuristics

We conclude the book with contributions that explore an open frontier in tractability. Solvers for SAT (Boolean constraints) and for the more general framework of "Satisfiability Modulo Theories" are highly successful in applications such as software verification. These solvers belong to the heuristic category we described before, in that they can in theory consume unreasonable amounts of computational resources in the worst case, but that they turn out to be very efficient in certain classes of applications, for reasons that sometimes defy our current understanding. Like the other contributors, the authors we have invited for these chapters nonetheless succeed in fullfilling two requirements: that of overviewing the state of the art in their repective areas, and that of sharing formalized or personal insights on the causes of tractability.

Part 5 starts off with a chapter from J. Marques-Silva and I. Lynce that gives a high-level overview of the techniques used in popular *SAT solvers*.

This is followed by a chapter from Nikolaj Bjørner and Leonardo de Moura that deals with Modern Satisfiability Modulo Theories (SMT) solvers.

Conclusions

Understanding how to solve an optimization problem is often hard. There is not a well-defined property that singlehandedly determines whether the problem can be efficiently solved. Our current understanding is that there is a toolbox of methods to approach the problems, rather than a comprehensive, unified theory of problem tractability.

For practitioners this is a complicated state of affairs as they face a set of options when dealing with a specific problem, and should try to understand which techniques apply. Often several techniques will work, depending on whether they look at the problem from the viewpoint of graphical structure,

of language restriction, or use, say, heuristic approaches. This collection should be a useful resource for such practitioners.

Ultimately, the broad range of viewpoints collected here does more than simply providing practical solutions and answers; it also gives us an overview of the deep questions that are currently open. In many applications such as those presented in the Propositional Satisfiability and Satisfiability Modulo Theory solvers, we have, indeed, a far from full understanding of, or agreement on, the reasons that make problems tractable.

What is it that makes these algorithms work dramatically better on instances found in practice, than on "worst-case" instances or "average-case" ones drawn from certain distributions? Have we explored all the facets of tractability, or can some new approaches emerge within or outside the known broad categories of graphical structure and knowledge restriction? Is there ultimately a unified way of making sense of tractability, or are we dealing with a phenomenon that intrinsically needs multi-faceted answers and solutions? These are some of the questions that are currently open. By overviewing the rich body of results developed across a number of research areas, and by also charting the limits of our current knowledge, we hope that this collection will also be useful to researchers who aim to advance the field.

Acknowledgements

We would like to thank Robert Mateescu, who was a co-organiser of the 2010 workshop and who contributed to early discussions about this book. We would also like to thank the staff at Cambridge University Press who worked hard on this volume: Clare Dennison, Róisín Munnelly and David Tranah.