Algorithmic and Computational Complexity Issues of MONET

Dissertation
zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat der
Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena

von Diplom-Informatiker Matthias Hagen
geboren am 29. Oktober 1979 in Ilmenau

Gutachter

1. Prof. Dr. Martin Mundhenk (Friedrich-Schiller-Universität Jena)
2. Prof. Dr. Michael R. Fellows (The University of Newcastle, Australien)

Tag der letzten Prüfung des Rigorosums: 26. November 2008
Tag der öffentlichen Verteidigung: 27. November 2008

Zusammenfassung

In dieser Dissertation beschäftigen wir uns mit dem Problem Monet – ein englisches Akronym für Mo(notone) n(ormal form) e(quivalence) t(est). Die Problemstellung bei Monet ist, die Äquivalenz einer monotonen disjunktiven Normalform φ und einer monotonen konjunktiven Normalform ψ zu entscheiden. Dies ist eigentlich ein Abdeckungsproblem und kann ebenso als das Aufzählen aller (in einem gewissen Sinne) minimalen Lösungen irgendeines Systems interpretiert werden. Deswegen gibt es auch eine Vielzahl sehr ähnlicher Fragestellungen aus unterschiedlichsten Anwendungsgebieten.

Unsere Resultate können grob in zwei Gruppen eingeteilt werden. Zum einen gibt es Resultate, die den Entwurf und die Analyse von Algorithmen betreffen, zum anderen sind Resultate enthalten, die eher Komplexitätsaspekte des Problems berühren. Im algorithmischen Teil geben wir untere Schranken für einige Algorithmen an und berichten über Resultate von praktischen Untersuchungen des theoretisch schnellsten Algorithmus in Experimenten. Im eher komplexitätstheoretischeren Teil dieser Dissertation zeigen wir für verschiedene eingeschränkte Klassen des Problems, dass sie sich mit logarithmischem Platzbedarf lösen lassen. Dadurch verbessern wir den Ressourcenbedarf gegenüber den vorher bekannten Polynomialzeitschranken. Darüberhinaus ordnen wir Monet für verschiedene Parameter in die Klasse der festparameter-handhabbaren Probleme ein.

Im Einzelnen beweisen wir die folgenden Hauptergebnisse unter Zuhilfenahme einer breiten Palette von algorithmischen und komplexitätstheoretischen Techniken.

- Verschiedene eingeschränkte Klassen des Problems Monet lassen sich mit logarithmischem Speicherplatzbedarf lösen. Dazu gehören die Klassen bei denen die DNF
  - nur eine konstante Anzahl Monome enthält (Abschnitt 4.1.1), nur Monome konstanter Größe enthält (Abschnitt 4.1.2), nur Monome entält, die jeweils nur höchstens konstant viele Variablen nicht enthalten (Abschnitt 4.1.3),
  - regulär (Abschnitt 4.2.1), ausgerichtet (Abschnitt 4.2.2), oder 2-monoton (Abschnitt 4.2.3) ist.
- Weder der DL-Algorithmus (Abschnitt 5.1.2), noch der BMR-Algorithmus (Abschnitt 5.1.3), noch der KS-Algorithmus (Abschnitt 5.1.4), noch der

HBC-Algorithmus (Abschnitt 5.2) für das Problem MONET laufen in Polynomialzeit bezüglich der Ein- und Ausgabegröße. Die Laufzeit der Algorithmen ist jeweils mindestens $n^{\Omega(\log\log n)}$, wobei n die Größe der Ein- und Ausgabe ist.

- Der FK-Algorithmus B für das Problem MONET erweist sich in praktischen Experimenten auf vielen Eingaben als konkurrenzfähig zum FK-Algorithmus A (Kapitel 6).

- MONET ist festparameter-handhabbar für die Parameter

  - Anzahl v der Variablen in φ und ψ (Abschnitt 7.1),
  - Anzahl m der Monome in φ (Abschnitt 7.2),
  - einen Parameter q, der die Variablenhäufigkeiten in φ beschreibt (Abschnitt 7.3),
  - und einen Parameter, der die Größe der Vereinigungen von Transversalen bzw. Kanten des Hypergraphen der DNF φ angibt (Abschnitt 7.4.3).

Diese Dissertation enthält Material, das in den Zeitschriften *Discrete Applied Mathematics*, *Information and Computation* und *Information Processing Letters* erschienen ist bzw. erscheinen wird, sowie Material, das auf der Konferenz „Mathematical Foundations of Computer Science“ (MFCS 2005), und den Workshops „Graph-Theoretic Concepts in Computer Science“ (WG 2007), „Parameterized and Exact Computation“ (IWPEC 2008) und „Workshop on Algorithm Engineering & Experiments“ (ALENEX 2009) vorgestellt und in den entsprechenden Tagungsbänden veröffentlicht wurde bzw. wird.

Abstract

In this thesis, we study the problem Monet—the Mo(notone) n(ormal form) e(quivalence) t(est)—that asks to decide equivalence of a monotone disjunctive normal form φ and a monotone conjunctive normal form ψ. This problem is a covering problem that can be interpreted as the task of enumerating all (in some sense) minimal solutions of some system. Hence, there is a huge number of similar questions in many problems from diverse applications.

Our results can roughly be divided into results on the design and evaluation of algorithms for Monet and results that rather touch complexity questions related to the problem. As for the algorithmic part, we will give lower bounds for several known algorithms and report results obtained by practically examining the theoretically fastest algorithm in computational experiments. As for the complexity part of this thesis, we show several restricted classes of the problem to be solvable in logarithmic space, which improves previously known polynomial time bounds. We also show Monet to be in the complexity class of fixed-parameter tractable problems with respect to several parameters.

More precisely, we prove the following main results using various algorithmic and computational complexity techniques.

- Several restricted classes of Monet are solvable in logarithmic space. In particular, these are the classes where the DNF
  - contains only a constant number of monomials (Section 4.1.1), contains only monomials of constant size (Section 4.1.2), contains only monomials that each do not contain only a constant number of variables (Section 4.1.3),
  - is regular (Section 4.2.1), aligned (Section 4.2.2), or 2-monotonic (Section 4.2.3).
- The DL-algorithm (Section 5.1.2), the BMR-algorithm (Section 5.1.3), the KS-algorithm (Section 5.1.4), and the HBC-algorithm (Section 5.2) for the problem Monet are not output-polynomial. Their running times are at least $n^{\Omega(\log \log n)}$, where n denotes the size of the input and output.
- FK-algorithm B for the problem Monet is experimentally competitive to FK-algorithm A on many classes (Chapter 6).

- Monet is fixed-parameter tractable with respect to the parameters
  - number v of variables in φ and ψ (Section 7.1),
  - number m of monomials in φ (Section 7.2),
  - a parameter q describing the variable frequencies in φ (Section 7.3),
  - and a parameter bounding the unions of transversals or edges of φ's associated hypergraph (Section 7.4.3).

This thesis contains material (to be) published in the journals *Discrete Applied Mathematics*, *Information and Computation* and *Information Processing Letters*, as well as material (to be) presented at, and (to be) published in the proceedings of, the conference "Mathematical Foundations of Computer Science" (MFCS 2005), and the workshops "Graph-Theoretic Concepts in Computer Science" (WG 2007), "Parameterized and Exact Computation" (IWPEC 2008) and "Workshop on Algorithm Engineering & Experiments" (ALENEX 2009).

Acknowledgments

I am very grateful to Martin Mundhenk for his constant support during my time as a graduate and PhD student in Jena. It was he who suggested working on problems related to monotone formulas as a Diploma (MSc) thesis topic, and who offered to supervise research on the problem Monet afterwards.

However, doing research would not have been possible without any financial support. Hence, I am very grateful to the Freistaat Thüringen for supporting me by a Landesgraduiertenstipendium through the beginning of my PhD time; to Rolf Niedermeier, in whose Deutsche Forschungsgemeinschaft (DFG) funded project OPAL (optimal algorithms for hard problems in computational biology), NI-369/2, I was pleased to work in the beginning of 2007; and to the team of the OBA (optimization of vehicle electrical system architectures) project in Kassel, especially Claudia Fohry's research group on programming languages / methodologies whose member I am since early 2007.

Motivation during the writing of this thesis also emanated from several meetings with other colleagues. I very much benefited from the broad experience in hypergraph transversal research of Khaled Elbassioni, who was my host for one week at the Max-Planck-Institut für Informatik in Saarbrücken and shared many insights with me. Khaled and Imran Rauf (MPI as well) were pleasant coauthors who made major contributions to the papers that cover Sections 5.2 and 7.4. Another very inspiring stay was my visit to Utz-Uwe Haus at Otto-von-Guericke-Universität Magdeburg. During Utz' explanations, I got the impression of finally starting to really understand the relation of Monet to problems in mathematical programming. As for the algorithm engineering part of my research I owe my thanks to the organizers, Matthias Müller-Hannemann and Stefan Schirra, and attendees of the GI-Dagstuhl research seminar 06362 on "Algorithm Engineering." It was this seminar that attracted my interest in experimental work with algorithms and especially the issues related to a careful choice of test instances. The algorithm engineering part also benefited from collaboration with Peter Horatschek, who implemented the FK-algorithms and documented the experiments.

Parts of this thesis have profited from discussions, comments, pointers to literature, etc. of several people such as James Bailey, Peter Damaschke, Michael Dom, Thomas Eiter, Mike Fellows, Daniel Fötsch, Celine Hébert, Steffen Klamt, Johannes Köbler, Michael Krüger, Kaz Makino, Hannes Moser, John Pfaltz, François Rioult, Ron Rymon, Ken Satoh, Thomas Schneider, Henning Schnoor, Elias Stavropoulos, Ken Takata, and Hisao Tamaki.

I would also like to thank the anonymous referees of several journals, conferences, and workshops for their feedback on parts of this thesis.

Last, but not least, I am very grateful to Saskia and our kids, Svea and Jan, for their continuous support and encouragement during the last years. Life would not be complete without you!

Contents

Chapter 1

Introduction

1.1 What means and to what end do we study MONET?[1]

Warning: What we do not(!) do. The first association when reading MONET probably is the famous French painter Claude Monet (1840–1926). His painting "Impression soleil levant" from 1872—a morning scene from Le Havre harbor—coined the term *Impressionism* that later on was used to describe a whole field of art. But as this thesis is on Theoretical Computer Science (TCS), this first impression on MONET is very misleading. We neither reveal any ancient TCS results due to Claude Monet (as we do not even know whether he was interested in efficient computation at all), nor do we study algorithmics and computational complexity with an "impressionistic" view. Hence, a big apology to all who started reading with the intention of discovering a not yet documented relation between the fine arts and TCS: Claude Monet is not(!) the topic of this work.

So, what is MONET? In this thesis we consider the problem MONET—an acronym for MO(notone) N(ormal form) E(quivalence) T(est)—that asks to decide equivalence of a monotone DNF (disjunctive normal form) and a monotone CNF (conjunctive normal form) (for formal definitions of any term we refer to Chapter 2). MONET is equivalent to TRANSHYP—given two hypergraphs, decide whether one is the transversal hypergraph of the other. TRANSHYP is a variant of the well-known problem VERTEXCOVER, in its hypergraph version also known as HITTINGSET. But instead of just asking for one vertex set of minimum cardinality that has a non-empty intersection with all edges, we are interested in the set of *all* minimal sets that "cover" resp. "hit" all edges.

Practical relevance. Consequently, MONET is a covering problem and can be interpreted as the enumeration of all (in some sense) minimal solutions of some system. Similar questions for enumeration of minimal solutions are ubiquitous in many problems from diverse applications. In fact, covering and enumeration problems that are MONET-equivalent or strongly related can be found in fields such as

[1] Title inspired by Friedrich Schiller's inaugural lecture (1789) at this thesis' author's university.

artificial intelligence, computational biology, databases, data mining, distributed systems, graph theory, logic, machine learning, mathematical programming, matroid theory, and mobile communication systems (cf. Chapter 3 for more details). The consequence is that any approach solving MONET can be easily transformed to solve a wide range of problems in very different fields. Thus, on the one hand, research on algorithms solving MONET and even any technique improving known procedures is very important from the point of view of practical applications.

And in theory: Unsettled complexity. On the other hand, MONET research is faced with some very interesting theoretical issues that we will describe in the following. The equivalence test for arbitrary monotone formulas (not necessarily in normal form) is coNP-complete [Rei03]. The same bound can be easily proven for arbitrary Boolean formulas in normal form. The coNP-completeness in this context means that these problems are "hard" since it is very unlikely that they are solvable by "fast" algorithms in the sense of deterministic polynomial running time (running time is usually measured with respect to the input size and polynomial running time is the usual notion of efficient solutions in complexity theory).

But note that in the MONET setting we require the input formulas to be monotone *and* in normal form, which together represent stronger "structural" restrictions and thus may ease the solution process. Whether these restrictions really can be exploited to develop polynomial MONET algorithms is an exciting open question for more than 25 years now [DT87, EG95, Joh91, LLK80, Lov92, Man02, Pap97]. The best currently known MONET algorithm has quasi-polynomial running time $n^{o(\log n)}$ [FK96], which still, after all, can be seen as an indication that MONET is probably not coNP-complete. Otherwise, all coNP-complete problems would be solvable in quasi-polynomial time—a result that hardly any expert expects. But as it is *quasi*-polynomial this algorithm is not "fast" as well.

Another indication that MONET probably is not coNP-complete is a recent result that shows MONET to be solvable using only $O(\log^2 n)$ guessed bits [EGM03, KS03a, KS03b]. Again, no expert expects any coNP-complete problem to be solvable with $O(\log^2 n)$ guessed bits as usually a polynomial number is assumed to be required.

MONET and the P-vs.-NP question. The currently unsolved complexity of MONET places it in the group of a handful—and thus really rare—problems that yet cannot be classified as "easy" (polynomial time solvable) or "hard" (NP- or coNP-complete). Hence, MONET, along with the prominent GRAPH ISOMORPHISM problem—given two graphs, decide if they are isomorphic—, may play some role in research on the popular P-vs.-NP question (one of the seven major problems in mathematics whose solution is worth a reward of $1,000,000$ \$ denoted by the Clay Mathematics Institute). The question is whether the class of deterministic polynomial time solvable problems (the class P) and the class of nondeterministic polynomial time solvable problems (the classes NP, resp. coNP) coincide. If

MONET is not solvable in polynomial time (formally written as MONET $\notin$ P), then this immediately implies a separation of P and NP. But as the P-vs.-NP question is open since the field of complexity theory emerged, we expect showing MONET $\notin$ P a really tough problem. On the other hand, there are no known complexity theoretic consequences that would follow from MONET $\in$ P, although this question is also open for many years now and, hence, seems to be tough as well. But note that also PRIMES—given an integer, decide whether it is prime—had an analogue unsettled complexity status for a long time until it was actually shown to be solvable in polynomial time only a few years ago [AKS04]. Thus, there is still a possibility of a breakthrough showing MONET $\in$ P. (Unfortunately, this breaktrough is not contained in this thesis.) Nevertheless, we expect new techniques to be necessary as we conjecture that none of the known MONET algorithms (for which not always upper and lower bounds on the running time are known) is polynomial.

Hence, the problems between "easy" or "hard" (resp., "fast" / no "fast" solution) constitute two very exciting challenges to algorithmicists and complexity theorists as well. One is to actually find a fast algorithm and the other is to find some kind of complexity theoretic arguments for or against a fast solution. As for PRIMES these challenges finally led to a fast solution [AKS04]. In the case of GRAPH ISOMORPHISM we have a complexity theoretic classification that would yield a solution to an unsolved question very strongly related to the P-vs.-NP question [KST93] in case that GRAPH ISOMORPHISM is not(!) polynomial. More precisely, GRAPH ISOMORPHISM cannot be NP-hard unless the so-called polynomial hierarchy collapses to its second level—an event that is supposed to be rather unlikely. As for MONET the situation is comparable to that of GRAPH ISOMORPHISM. There are several indications that MONET is more likely to be polynomial than to be hard—like the quasi-polynomial algorithms by Fredman and Khachiyan [FK96] and the solvability with bounded nondeterminism [EGM03, KS03a, KS03b]. Furthermore, the consequences of being not polynomial even seem to be a little stronger for MONET than for GRAPH ISOMORPHISM.

However, all that is known for the fast solvability yet is that MONET is polynomial if and only if its computational variant MONET$'$—given an irredundant, monotone DNF, compute the equivalent irredundant, monotone CNF—is output-polynomial [BI95a]. Here, output-polynomiality is an appropriate notion of fast solvability for computation problems [JPY88]. Note in this context, that a slight generalization of MONET$'$ is very unlikely to have an output-polynomial solution. Namely, finding an algorithm that, given a monotone formula (not necessarily in DNF), computes the irredundant, monotone CNF in output-polynomial time is the same as showing P = NP and thus as hard as solving the P-vs.-NP question [GHM05].

Easy classes as a way out?! As there is no known fast algorithm for MONET yet, one branch of research focuses on identifying restrictions of the inputs that sufficiently simplify the problem to allow for polynomial time solutions. Such restrictions then yield "easy" classes of the problem. One example might be to examine instances where the DNF only contains monomials of constantly bounded size. And in fact, MONET with constantly size bounded monomials has polynomial time algorithms [BEGK00, EG95]. Many other easy classes are known (cf. Chapter 4 for more details). Due to the successes in searching for more and more easy classes, many practically interesting input instances can be solved in polynomial time although there is no known polynomial algorithm for MONET itself. This is maybe the major reason that makes worthwhile every effort invested in research on easy classes of MONET. But there are two other, possibly not less important reasons. One is that the knowledge of easy classes reveals new information about the really hard problems apparent when trying to attack polynomial solvability of the general problem MONET itself. And the other is that the easy classes might prove useful when looking for lower bound or hardness results for MONET.

Known MONET algorithms. There are many known algorithms solving MONET or MONET′ (cf. Chapter 5 for more details). They are inspired by very diverse techniques from different fields—which is not that surprising having in mind the broad range of equivalent problems. We do not really have to distinguish between algorithms for MONET or MONET′ as they can be easily transformed to solve the respective other problem version.

One of the earliest approaches was the Berge-multiplication algorithm (cf., e. g., [Ber89]). So far, it is the only algorithm having a known lower bound (that states what resources (e. g., running time) are minimally needed using the algorithm to solve arbitrary MONET instances). Takata's lower bound shows that Berge-multiplication is not fast [Tak07]. But as Berge-multiplication can be easily implemented, there have been several improvements of it [BMR03, DL05, KS05, Uno02, US03]. Analyses of the running times of the improved versions have been pending.

Other algorithms do not follow a term based approach as Berge-multiplication does but use a variable based decomposition technique [BMR03, EGM03, Eit91, MR94, Rym92, Rym94a]. Maybe the most famous variable based algorithms are the FK-algorithms A and B by Fredman and Khachiyan [FK96]. Algorithm B is the MONET algorithm with the currently best upper bound on the running time of $n^{o(\log n)}$. This upper bound is a guarantee on the amount of running time that will never be exceeded on any instance by FK-algorithm B.

Inspired by model-based diagnosis techniques are Reiter's algorithm [Rei87] (and its improvements [GSW89, Wot01]) and several other recently published algorithms, e. g., [LJ03, TT02]. There are MONET algorithms based on techniques that were useful in data mining settings [GKM$^+$03, HBC07], as well as genetic algorithms [LJ02, Vin99a, VØ00b] or parallel approaches [Elb08, KBEG07a].

Although there are many very diverse algorithms, all approaches have in common that there are only very few proven non-trivial upper or lower bounds on their running times. Hence, theoretical knowledge of the algorithms' behavior is not really well developed. Having in mind the wide applicability of MONET results, this situation is not satisfying at all.

Algorithm Engineering. One possibility—besides thorough theoretical analyses—to get some impression of (practical) performance of an algorithm is to implement it and run it on a lot of well-chosen inputs. This is one branch of the emerging field of Algorithm Engineering (cf. Chapter 6 for more details). Though, not that many experimental studies on MONET algorithms have been published and all have some lack of coverage [BMR03, DL05, KBEG06, KS05, LJ03, TT02, US03]. Some of the studies only show the potential of a single approach and do not really compare it to others. In addition, none of them includes the theoretically best algorithm, FK-algorithm B [FK96]. Hence, it is not clear at all, which of the currently known algorithms is the best choice on which kind of instances.

This thesis' contributions. In this thesis, we try to shed some light on the open questions discussed in the previous paragraphs.

As for the easy classes, there are two main questions. First, how easy are the easy classes? Can we do better than polynomial running time? The second one is to find new easy classes. In this thesis we work on both questions. We analyze the known easy classes with the intention of tightening the known resource bounds. Thereby, we show some of the easy classes to be solvable with logarithmic space, which improves the known polynomial time bounds. And we discover some new easy classes.

As for the known MONET algorithms, our conjecture is that none of the currently known approaches is fast in the sense of polynomial (resp., output-polynomial) running time. We start working on proving our conjecture, by giving non-trivial lower bounds on the running times of several algorithms. From the point of view of practical applications there is also another interesting open question, namely that for the performance of the theoretically fastest MONET algorithm, FK-algorithm B, in algorithmic experiments. Using the Algorithm Engineering methodology we compare FK-algorithm B to FK-algorithm A and show it to be competitive on our testbed. This result somehow adjusts the folklore assumption that FK-algorithm B should be practically much slower than its theoretically worse relative, FK-algorithm A.

As for computational complexity issues, we examine the fixed-parameter tractability of MONET, a subject that has not been addressed in the literature so far. Informally, a problem is said to be fixed-parameter tractable if there is an algorithm whose running time is arbitrary (exponential or even worse) in a given parameter but only includes polynomial terms for the input size (cf. Chapter 7 for more details). The idea then is that for small values of the parameter the running time of the algorithm behaves like being polynomial. We show MONET to be in the class FPT of fixed-parameter tractable problems with respect to several parameters.

1.2 Legend to this thesis

This thesis is organized as follows. In Chapter 2, we introduce basic notations and concepts as well as the problem MONET. Chapter 3 then gives some further motivation for studying MONET in the sense that we collect problems equivalent or strongly related to MONET from very different fields. Afterwards, in Chapter 4, we introduce easy classes of MONET that are restrictions of the input formulas that allow for polynomial time solutions. In contrast, Chapter 5 contains algorithms solving the general problem MONET without any restrictions on the input. Some experimental results for important MONET algorithms follow in Chapter 6. In Chapter 7, we then turn to the emerging field of parameterized complexity and analyze MONET in that setting. Finally, some concluding remarks follow in Chapter 8.

Parts of this thesis have already appeared or are accepted for publication in refereed journals, in refereed conference and workshop proceedings, as technical reports, or as manuscripts. In particular, Chapter 4 contains results from [GHM05, GHM08], and [Hag06]; Chapter 5 contains results from [Hag07a] and [EHR08a]; Chapter 6 contains results from [HHM09], whereas the choice of test instances is based on observations contained in [BHHM07]; and Chapter 7 contains results from [Hag07b] and [EHR08b].

Chapter 2

Preliminaries

In this chapter we introduce basic notations and concepts used throughout the whole thesis. We start with some of the basic notions used in analyzing algorithms and evaluating problems based on their computational complexity in Section 2.1.

In Section 2.2 we formally define the problem MONET. Therefore, we first have to give some facts and notation on formulas and the like.

As sometimes things can be expressed more easily using hypergraph notations, we introduce this concept in Section 2.3. We also give the problem TRANSHYP that then, in Section 2.4, turns out to be the hypergraph analogue of MONET.

Finally, in Section 2.5 we show that basic computations needed throughout the thesis can be managed by very efficient procedures in the sense of their computational complexity. We especially point out the preprocessing steps from Section 2.5.1 as we later on assume that all the corresponding conditions are implicitly checked when necessary without explicitly mentioning it every time.

2.1 Algorithms and computational complexity

In computational complexity one is interested in the amount of resources necessary to solve problems. Problems considered in this thesis are usually decision or computation problems. The term *decision problem* means that the answer of an algorithm just is a positive or negative answer to a question concerning the input whereas for a *computation problem* we expect some "real" output.

Example 2.1.1. An example of a decision problem on natural numbers is the problem PRIMES—given a natural number, decide if it is prime. An appropriate algorithm would just produce "Yes" as output on the inputs 2, 3, 5, 7, 11, ..., and "No" for all inputs that are not prime.

An example of a computational problem is to compute, on input a natural number n, the value 2^n. Here no longer a "Yes" or "No" suffices as output.

The usual machine model on which algorithms run in computational complexity settings is that of Turing machines with input, output and working tapes. Admittedly, we do not "program" Turing machines at a low level giving complete transition functions but provide our algorithms in more readable pseudocode also often abstracting away from the different tapes of the machines. The reader who is interested in the basics of Turing machines may find appropriate definitions and justification that low level Turing machine programming can be replaced by giving pseudocode algorithms, e. g., in [Pap94].

2.1.1 Tools for algorithm analysis

It would be very cumbersome if one would always have to give *exact* values for the resource requirements of algorithms. Thus, we only estimate *asymptotic* running times or space requirements. One of the main tools in asymptotic estimations is that of *O-notation* defined as follows.

Definition 2.1.2 (O-Notation)**.** *For a given function $f(n)$ on the natural numbers we have the following.*

$$
\begin{aligned}
\Theta(f(n)) &= \{g(n) : \textit{there exist positive constants } c_1, c_2, \textit{ and } n_0 \textit{ such that} \\
&\qquad c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n) \textit{ for all } n \geq n_0\}. \\
O(f(n)) &= \{g(n) : \textit{there exist positive constants } c \textit{ and } n_0 \textit{ such that} \\
&\qquad g(n) \leq c \cdot f(n) \textit{ for all } n \geq n_0\}. \\
\Omega(f(n)) &= \{g(n) : \textit{there exist positive constants } c \textit{ and } n_0 \textit{ such that} \\
&\qquad c \cdot f(n) \leq g(n) \textit{ for all } n \geq n_0\}. \\
o(f(n)) &= \{g(n) : \textit{for any constant } c > 0 \textit{ there exists a constant } n_0 > 0 \\
&\qquad \textit{such that } g(n) \leq c \cdot f(n) \textit{ for all } n \geq n_0\}.
\end{aligned}
$$

Thus, if $g(n) \in \Theta(f(n))$, then for large enough n the function $g(n)$ is equal to $f(n)$ up to a constant factor. We also say that $f(n)$ is an *asymptotically tight* bound for $g(n)$. The notion $O(f(n))$ is used to give an asymptotic *upper bound* on $f(n)$, $\Omega(f(n))$ gives an asymptotic *lower bound* on $f(n)$, and $o(f(n))$ is used to denote an upper bound that is not asymptotically tight.

Example 2.1.3. Consider the functions $2n$, n^2, and $2n^2$. We have $2n \in O(n^2)$ and $2n^2 \in O(n^2)$. Among these, only the second bound is asymptotically tight, whereas the first is not. This yields $2n \in o(n^2)$, $2n^2 \notin o(n^2)$ but $2n^2 \in \Theta(n^2)$.

2.1.2 Complexity classes

We classify problems into *complexity classes* according to the resources needed to solve them. The resources are usually measured according to the size of the input. Examples of complexity classes are L (consisting of problems solvable using logarithmic space), NL (problems solvable using nondeterminism and logarithmic space), P (problems solvable using polynomial time), NP (problems solvable using nondeterminism and polynomial time), coNP (complements of problems in NP), PSPACE (problems solvable using polynomial space). The inclusion structure is

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \begin{array}{c}\mathsf{NP}\\ \mathsf{coNP}\end{array} \subseteq \mathsf{PSPACE}.$$

See [Pap94] for formal definitions of these classes. Note that only problems in P are usually said to be solvable "fast", whereas problems above P are usually considered to be "difficult" in the sense that too many resources are needed to solve them with justifiable effort.

It is not known which of the above inclusions are proper and especially one of the central open questions in computational complexity is that of whether P = NP, also known as the P-vs.-NP question. This corresponds to the question of whether using a polynomial number of nondeterministic (guessed) bits adds any power to polynomial time computations. It is straightforward to simulate any such NP computation in $2^{O(\mathrm{poly}(n))}$ deterministic time, where $\mathrm{poly}(n)$ denotes functions polynomial in n. But it is not known whether $2^{O(\mathrm{poly}(n))}$ also is a tight lower bound.

Note that the above classes only make sense for decision problems.

Example 2.1.4. Consider the example of computing 2^n given a natural number n. Using a bit encoding, the "size" of the input is $\log n$ and the size of the output is n. Hence, the sheer size of the output causes the problem not to fit in P as already writing the output on the output tape causes any machine to use exponential time in the input size. But the procedure of just writing a single 1 followed by n 0's is quite efficient and, hence, for computation problems there is another notion of fast solvability.

Definition 2.1.5 (Output-Polynomial, [JPY88])**.** *An algorithm is said to be* output-polynomial *iff its running time is bounded polynomially in the sum of the sizes of the input and output.*

Note that output-polynomiality is a meaningful notion of fast solvability in case of computation problems.

2.1.3 Reductions and hardness

One powerful tool to compare different decision problems is complexity theoretic reduction.

Definition 2.1.6 (Reduction)**.** *A problem A is* polynomial-time reducible *to the problem B, $A \leq_m^p B$ for short, iff there is a polynomial time computable function f that maps inputs for A to inputs for B such that for all inputs x for A we have $x \in A$ iff $f(x) \in B$.*

There is also a notion of equivalence of problems relative to polynomial time computations.

Definition 2.1.7 (Polynomial-Time Equivalence)**.** *Problems A and B are* polynomial-time equivalent*, $A \equiv_m^p B$ for short, iff $A \leq_m^p B$ and $B \leq_m^p A$.*

Using the reduction concept we define hardness and completeness of problems for complexity classes.

Definition 2.1.8 (Hardness, Completeness)**.** *Let $\mathcal{C}$ be a complexity class and A a problem. A is said to be $\mathcal{C}$-hard iff for every $C \in \mathcal{C}$ we have $C \leq_m^p A$. If A is $\mathcal{C}$-hard and contained in $\mathcal{C}$ we also say that A is $\mathcal{C}$-complete.*

Note that $\leq_m^p$ is only interesting for problems "above" the class P as the resources allowed for $\leq_m^p$ are as powerful as P itself is and hence the reduction does not allow any meaningful separation of problems in P (except trivial cases, all problems in P are P-hard relative to $\leq_m^p$). Hence, the above defined notions of hardness and completeness relative to $\leq_m^p$ are only meaningful in context of classes containing P. For the purpose of comparing problems in P there are more restrictive notions of reduction. But as we will mainly discuss problems not known to be in P, these more fine-grained reductions are beyond the scope of this thesis.

In Lemma 2.4.1 we give a very basic example of polynomial time problem equivalence upon defining the corresponding formula and hypergraph problems.

2.2 Boolean formulas, monotonicity, and the problem Monet

In this section, we introduce the basic concepts necessary for defining and understanding the problem Monet.

2.2.1 Syntax and semantics of Boolean formulas

As Monet is a formula problem, we give some basics on Boolean formulas before defining the problem itself. We start with the *syntax* of Boolean formulas.

Definition 2.2.1 (Syntax of Boolean Formulas)**.** *The constants* 0 *and* 1 *and every variable x_i, with* index $i \in \mathbb{N}$, *are* Boolean formulas. *Furthermore, let α and β be Boolean formulas, then so are $\neg(\alpha)$, $(\alpha \land \beta)$, and $(\alpha \lor \beta)$.*

The connectives in Definition 2.2.1 are called NOT or *negation* ($\neg$), AND or *conjunction* ($\land$), and OR or *disjunction* ($\lor$). Variables and their negations are called *literals*. We say that a literal is *negative* iff it is a negated variable. Otherwise, it is a *positive* literal.

Note that Boolean formulas might be arbitrarily nested and that we may often leave out some parentheses.

Example 2.2.2. The following is an example of a Boolean formula.

$$(((x_1 \land \neg x_2) \lor \neg x_3) \land \neg(((x_4 \lor \neg x_5) \land \neg x_1) \lor x_6)) \lor (x_4 \land x_5 \land \neg x_7).$$

Now that we have defined the syntax of Boolean formulas, we turn to the corresponding semantics. We first need the notion of truth values and assignments.

Definition 2.2.3 (Assignments). *A Boolean variable x_i can be set to the* truth value *either* **`true`** *(also denoted by 1) or* **`false`** *(denoted by 0).*

An assignment *$\mathcal{A}$ for a Boolean formula α is a subset of α's variables. The notion is that variable x_i is set to* **`true`** *by $\mathcal{A}$ iff $x_i \in \mathcal{A}$. When the underlying variable set V is unambiguous, we denote by $\overline{\mathcal{A}} = V \setminus \mathcal{A}$ the* complement *of the assignment $\mathcal{A}$.*

Based on assignments we now describe the *evaluation* of Boolean formulas—their *semantics*.

Definition 2.2.4 (Semantics of Boolean Formulas, Satisfiability). *For any assignment $\mathcal{A}$ we have $\mathcal{A}(0) = 0$ and $\mathcal{A}(1) = 1$. Now let α and β be Boolean formulas and $\mathcal{A}$ an assignment. We have the following properties.*

$$\begin{aligned} \mathcal{A}(\neg\alpha) = 1 \quad &\text{iff} \quad \mathcal{A}(\alpha) = 0, \\ \mathcal{A}(\alpha \wedge \beta) = 1 \quad &\text{iff} \quad \mathcal{A}(\alpha) = 1 \text{ and } \mathcal{A}(\beta) = 1, \\ \mathcal{A}(\alpha \vee \beta) = 1 \quad &\text{iff} \quad \mathcal{A}(\alpha) = 1 \text{ or } \mathcal{A}(\beta) = 1. \end{aligned}$$

In case of $\mathcal{A}(\alpha) = 1$ we also say that $\mathcal{A}$ satisfies *α.*

Example 2.2.5. Let α be the formula from Example 2.2.2, i. e.,

$$\alpha = (((x_1 \wedge \neg x_2) \vee \neg x_3) \wedge \neg(((x_4 \vee \neg x_5) \wedge \neg x_1) \vee x_6)) \vee (x_4 \wedge x_5 \wedge \neg x_7).$$

Consider the assignment $\mathcal{A} = \{x_4, x_5\}$. The complement on the variable set of α then is $\overline{\mathcal{A}} = \{x_1, x_2, x_3, x_6, x_7\}$. It turns out that $\mathcal{A}(\alpha) = 1$ as $\mathcal{A}$ satisfies $(x_4 \wedge x_5 \wedge \neg x_7)$ and this is a disjunctive part of α. Now consider the assignment $\mathcal{A}' = \{x_1, \ldots, x_7\}$ containing all variables and note that $\mathcal{A}'(\alpha) = 0$. Hence, in this case "increasing" the assignment $\mathcal{A}$ to include all variables "decreases" the truth value.

2.2.2 Monotone formulas and equivalence

There is a special class of Boolean formulas where such an increase-decrease contrast as observed in Example 2.2.5 is impossible—the so-called *monotone* or *positive* formulas.

Definition 2.2.6 (Monotone Formulas). *A Boolean formula is* monotone *iff it has only $\wedge$ and $\vee$ as connectives. No negation signs are allowed!*

For monotone formulas every superset of a satisfying assignment is guaranteed to also be satisfying. This behavior of "monotone increasing" truth values is the reason for the name "monotone" (the alternative naming "positive" stems from the fact that there are only positive literals).

Example 2.2.7. The formula from Example 2.2.5 is not monotone since it contains negations. As an example for a monotone formula consider

$$\alpha = (((x_1 \wedge x_2) \vee x_3) \wedge ((x_4 \vee x_5) \wedge x_1)) \vee (x_4 \wedge x_5).$$

Note that the assignment $\mathcal{A} = \{x_4, x_5\}$ satisfies α as $\mathcal{A}$ satisfies $(x_4 \wedge x_5)$, a disjunctive part of α. In this case, every superset of $\mathcal{A}$ and especially the assignment $\mathcal{A}' = \{x_1, \ldots, x_5\}$ containing all variables also satisfies α.

Based on the semantics of formulas we have the following equivalence relation.

Definition 2.2.8 (Equivalence)**.** *Two Boolean formulas α and β are* equivalent *iff for all assignments $\mathcal{A}$ it holds that $\mathcal{A}(\alpha) = \mathcal{A}(\beta)$.*

Example 2.2.9. The formulas $\neg(x_1 \vee \neg x_2) \vee (\neg x_1 \wedge x_3)$ and $\neg x_1 \wedge (x_2 \vee x_3)$ are equivalent.

An easy observation is that there is no equivalent monotone formula for such a simple Boolean formula like $\neg x_1$. Hence, monotone formulas are a proper subclass of Boolean formulas and thus a real restriction. Nevertheless, they appear in many practically relevant settings and often problems restricted to monotone formulas become very much easier than for arbitrary ones. Consider for instance the problem of deciding satisfiability of formulas. This is an NP-complete task for arbitrary formulas [Coo71]. Informally, this means that it is very unlikely to check satisfiability of arbitrary formulas within an acceptable time bound.

For monotone formulas it becomes trivial. Just evaluate the formula on a single assignment, namely the one containing all variables. If this assignment does not satisfy a monotone formula then no other can and the formula is equivalent to the unsatisfiable constant 0. Note that this might only happen if the constant 0 itself is "nested" somewhere in the monotone formula under consideration.

2.2.3 Normal forms of formulas

Note that so far formulas, and even monotone ones, may be arbitrarily nested. In many situations it is more practicable to require formulas to be of a certain form, the so-called *normal forms*.

Definition 2.2.10 (Normal Forms)**.** *A Boolean formula is in* DNF (disjunctive normal form) *if it is a disjunction of conjunctions of literals. Analogously, a Boolean formula is in* CNF (conjunctive normal form) *if it is a conjunction of disjunctions of literals.*

In the case of monotone normal forms we can replace "literals" by "variables" in Definition 2.2.10.

Example 2.2.11. The formula $(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_5 \wedge x_6)$ is a DNF, whereas $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_5 \vee x_6)$ is a CNF. Both are monotone.

Note, that formulas in normal form have a very regular structure—they consist of well-defined subparts.

Definition 2.2.12 (Monomial, Clause)**.** *A conjunction of literals is called a* monomial*. A disjunction of literals is called a* clause*. Monomials and clauses are also called* terms*.*

Example 2.2.13. A monomial: $(x_1 \wedge x_2 \wedge x_3)$. A clause: $(x_1 \vee x_2 \vee x_3)$. Again, both are monotone.

Definition 2.2.14 (Containment)**.** *A monomial m* contains *another monomial m' iff each assignment satisfying m also satisfies m'. A clause c* contains *another clause c' iff each assignment satisfying c' also satisfies c.*

Example 2.2.15. The monomial $(x_1 \wedge x_2 \wedge x_3)$ contains the monomial $(x_1 \wedge x_3)$. The clause $(x_1 \vee x_2 \vee x_3)$ contains the clause $(x_1 \vee x_2)$.

For each formula there are important special terms, the min- and the maxterms.

Definition 2.2.16 (Minterm, Maxterm)**.** *A monomial m is a* minterm *or* prime implicant *of a Boolean formula α iff (1) each assignment that satisfies m also satisfies α and (2) m does not contain another monomial also having property (1).*

A clause c is a maxterm *or* prime implicate *of a Boolean formula α iff (1) each assignment that satisfies α also satisfies c and (2) c is not contained in another clause also having property (1).*

Example 2.2.17. The monomial $(x_1 \wedge x_2 \wedge x_3)$ is a minterm of $\alpha = (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_5 \wedge x_6)$, whereas $(x_3 \vee x_4 \vee x_5)$ is a maxterm of α.

As for monotone terms, if there is no danger of ambiguity, we often abstract from their representation as formulas and just view them as variable subsets. Thereby, a monotone term t is represented by the subset of the variables that occur in t.

Example 2.2.18. The monomial $(x_1 \wedge x_2 \wedge x_3)$ and the clause $(x_1 \vee x_2 \vee x_3)$ both correspond to the set $\{x_1, x_2, x_3\}$.

Note that in this way, monotone terms can also be seen as assignments. Hence, the naming of min- and maxterms becomes clearer. Minterms of a formula α are the minimal terms that, seen as assignments, satisfy α. Maxterms of α are the maximal terms whose complements are non-satisfying assignments of α.

Using the set interpretation of terms we also view monotone normal forms as subsets of their variables' power set.

Example 2.2.19. The set representation of both the DNF and CNF from Example 2.2.11 is $\{\{x_1, x_2, x_3\}, \{x_1, x_4\}, \{x_2, x_5, x_6\}\}$.

One often is interested in *shortest* normal forms—normal forms with fewest variable occurrences equivalent to a given formula. From the definition of min- and maxterms it is clear that shortest DNFs have to consist of minterms and shortest CNFs have to consist of maxterms. To further describe shortest monotone normal forms, we first define an appropriate cover relation between monotone terms.

Definition 2.2.20 (Cover). *A monotone term t* covers *a monotone term t' if $t \subseteq t'$.*

Note that coverage is strongly related to the notion of containedness from Definition 2.2.14. In the normal form setting this means that whenever terms $t \subseteq t'$ both appear in the same normal form, the term t' would be logically "absorbed."

Example 2.2.21. The monomial $(x_1 \wedge x_2 \wedge x_3)$ covers the monomial $(x_1 \wedge x_2 \wedge x_3 \wedge x_4)$. The DNF $(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$ of both then is equivalent to $(x_1 \wedge x_2 \wedge x_3)$.

Now we are ready to define the notion of shortest monotone normal forms in another way.

Definition 2.2.22 (Irredundancy). *A monotone normal form is* irredundant *iff there are no two terms in it such that one covers the other.*

Example 2.2.23. The DNF and the CNF from Example 2.2.11 both are irredundant, whereas $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$ is not.

It is a well-known fact that the irredundant DNF and CNF of a monotone formula are unique [Qui53]. (It is straightforward that the DNF consists of exactly all the minterms whereas the CNF consists of all the maxterms.) Hence, the irredundant normal forms of monotone formulas are the shortest equivalent normal forms. Note that in case of arbitrary formulas it is not that easy as, e. g., not all minterms have to be included in an equivalent DNF.

2.2.4 The problem Monet

In our problem setting, where we are interested in equivalence of monotone normal forms, we concentrate on irredundant normal forms as inputs. This is reasonable as terms in monotone normal forms that are covered by other terms do not change the truth table of the formula. Such terms can be identified by a trivial logspace preprocessing step (cf. Section 2.5.1). Thus, the main subject of this thesis—the problem Monet of deciding equivalence of monotone normal forms—can be defined as follows.

Monet:	instance:	irredundant, monotone DNF φ and CNF ψ
	question:	are φ and ψ equivalent?

MONET is an acronym for MO(notone) N(ormal form) E(quivalence) T(est). Note that we only concentrate on the case of two different normal forms as input, as testing equivalence of two irredundant, monotone DNFs or two irredundant, monotone CNFs is just identity checking.

The *size* of a MONET-instance (φ, ψ) is the number of variable occurrences in φ and ψ. Throughout this thesis φ will always denote a monotone DNF and ψ will always denote a monotone CNF. Also note that MONET is a decision problem. For those who prefer "real" computation, the appropriate version is the following computational variant MONET′.

MONET′:	input:	irredundant, monotone DNF φ
	output:	equivalent irredundant, monotone CNF ψ

Note that the size of the equivalent CNF may be exponential.

Example 2.2.24. The DNF $\varphi = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \ldots (x_{2n-1} \wedge x_{2n})$ has size $2n$ but the equivalent CNF contains 2^n clauses.

Hence, there cannot be an algorithm solving MONET′ in time polynomial in the input size. Thus, a suitable notion of fast solvability for MONET′ is that of output-polynomial time as described in Section 2.1.2.

2.2.5 State of the art in MONET complexity

The exact complexity of MONET is an exciting open question for more than 25 years now [DT87, EG95, Joh91, LLK80, Lov92, Man02, Pap97]. The algorithm with the currently best known upper bound is the Fredman-Khachiyan-algorithm B (cf. Section 5.3). It has quasi-polynomial running time $n^{o(\log n)}$ [FK96], which still, after all, can be seen as an indication that MONET is probably not coNP-complete. Another indication that MONET probably is not coNP-complete is a recent result that shows MONET to be solvable using only $O(\log^2 n)$ nondeterministic bits [EGM03, KS03a, KS03b]. Hardly any expert expects any coNP-complete problem to be solvable with quasi-polynomial time or only $O(\log^2 n)$ nondeterministic bits.

But MONET is not expected to be hard for the class $\mathsf{coNP}[\log^2 n]$ of problems solvable in polynomial time using $O(\log^2 n)$ nondeterministic bits. The reason is that already $O(\log^2 n / \log\log n)$ nondeterministic bits suffice to decide MONET [EGM03]. Moreover, we do not have any hardness result for MONET yet—not even for classes contained in L. We also do not have any arguments against polynomiality of MONET although showing MONET $\in$ P seems to be a tough problem. But a breakthrough result showing MONET $\in$ P is not impossible yet.

2.3 Hypergraphs, transversals, and the problem TransHyp

In this section we introduce some of the basic concepts related to the hypergraph version TransHyp of the problem Monet. The main reason is that often things related to Monet (e. g., some algorithms, cf. Chapter 5) can be expressed in a more convenient and precise way when using hypergraph notation.

2.3.1 Hypergraphs and transversals

Definition 2.3.1 (Hypergraph)**.** *A* hypergraph $\mathcal{H} = (V, E)$ *consists of a set* V *of* vertices *and a finite family* E *of subsets of* V *—the* edges *of* $\mathcal{H}$*.*

If there is no danger of ambiguity, we also identify a hypergraph by its edge set or use the edge set to refer to $\mathcal{H}$. The notion then is that the vertex set of $\mathcal{H}$ is exactly the set of vertices present in the edges. The *size* of $\mathcal{H}$ is the number of occurrences of vertices in the edges.

Example 2.3.2. The hypergraph $\mathcal{H} = \{\{v_1, v_2\}, \{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_4, v_5\}\}$ has size 10.

A very important notion in the course of establishing a hypergraph variant of Monet is that of transversals—also often called hitting sets.

Definition 2.3.3 ((Minimal) Transversal)**.** *A* transversal *of a hypergraph* $\mathcal{H}$ *is a vertex set* $t \subseteq V$ *that has a non-empty intersection with each edge of* $\mathcal{H}$*. A transversal* t *is* minimal *iff no proper subset of* t *is a transversal.*

Thus, just as hypergraphs are a generalization of graphs, transversals generalize the notion of vertex covers. Note that the set of all minimal transversals also is a hypergraph.

Definition 2.3.4 (Transversal Hypergraph)**.** *The set of all minimal transversals of* $\mathcal{H}$ *forms the* transversal hypergraph $\mathrm{Tr}(\mathcal{H})$*.*

Example 2.3.5. The hypergraph $\mathcal{H} = \{\{v_1, v_2\}, \{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_4, v_5\}\}$ has the transversal hypergraph $\mathrm{Tr}(\mathcal{H}) = \{\{v_1, v_3\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}\}$.

Definition 2.3.6 (Simple)**.** *A hypergraph* $\mathcal{H}$ *is* simple *iff it does not contain two hyperedges* e, f *with* $e \subseteq f$*.*

By $\min(\mathcal{H})$ we denote the simple hypergraph consisting of the minimal hyperedges of $\mathcal{H}$ with respect to set inclusion.

Example 2.3.7. The hypergraph $\mathcal{H} = \{\{v_1, v_2\}, \{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_4, v_5\}\}$ is not simple as $\{v_1, v_2, v_3\}$ contains the two other edges $\{v_1, v_2\}$ and $\{v_2, v_3\}$. We can derive $\min(\mathcal{H})$ by simply excluding $\{v_1, v_2, v_3\}$.

Since $\min(\mathcal{H})$ can be easily computed in polynomial time and we have $\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\min(\mathcal{H}))$ for every hypergraph $\mathcal{H}$, we concentrate on the transversal hypergraphs for simple hypergraphs.

2.3.2 The problem TransHyp

The problem TransHyp is defined as follows.

TransHyp:	instance:	simple hypergraphs $\mathcal{G}$ and $\mathcal{H}$
	question:	$\mathcal{H} = \mathrm{Tr}(\mathcal{G})$?

Note that for simple hypergraphs we have the following identity.

Lemma 2.3.8 ([Ber89])**.** *We have* $\mathrm{Tr}(\mathrm{Tr}(\mathcal{H})) = \mathcal{H}$ *for any simple hypergraph* $\mathcal{H}$*.*

Thus, for two simple hypergraphs $\mathcal{G}$ and $\mathcal{H}$, testing $\mathcal{H} = \mathrm{Tr}(\mathcal{G})$ is equivalent to testing $\mathcal{G} = \mathrm{Tr}(\mathcal{H})$ and hence the roles of $\mathcal{G}$ and $\mathcal{H}$ in the definition of TransHyp can be exchanged.

Again, for those who prefer computation, the appropriate version is the following computational variant TransHyp′.

TransHyp′:	input:	simple hypergraph $\mathcal{H}$
	output:	$\mathrm{Tr}(\mathcal{H})$

Note that TransHyp′ is often called *transversal hypergraph generation* and that even for simple hypergraphs the size of the transversal hypergraph may be exponential.

Example 2.3.9. The hypergraph $\mathcal{H} = \{\{v_1, v_2\}, \{v_3, v_4\}, \ldots \{v_{2n-1}, v_{2n}\}\}$ has size $2n$ but 2^n minimal transversals.

Hence, there cannot be an algorithm computing the transversal hypergraph in time polynomial in the input size. Thus, a suitable notion of fast solvability for TransHyp′ is that of output-polynomial time as described in Section 2.1.2.

2.3.3 Further hypergraph notation

Let us finish this section on hypergraphs by defining some further notation.

Definition 2.3.10 (Unions)**.** *For simple hypergraphs* $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$ *and* $\mathcal{H} = \{h_1, h_2, \ldots, h_{m'}\}$ *we have the following "union" operators*

$$\begin{aligned} \mathcal{G} \cup \mathcal{H} &= \{g_1, g_2, \ldots, g_m, h_1, h_2, \ldots, h_{m'}\} \qquad \textit{and} \\ \mathcal{G} \vee \mathcal{H} &= \{g_i \cup h_j : i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, m'\}. \end{aligned}$$

An important property using these unions is the following.

Proposition 2.3.11 ([Ber89])**.** *Let* $\mathcal{G}$ *and* $\mathcal{H}$ *be simple hypergraphs. Then* $\mathrm{Tr}(\mathcal{G} \cup \mathcal{H}) = \min(\mathrm{Tr}(\mathcal{G}) \vee \mathrm{Tr}(\mathcal{H}))$*.*

Finally, we name the complements of transversals.

Definition 2.3.12 ((Maximal) Independent Set)**.** *Let* $\mathcal{H}$ *be a hypergraph. A subset of vertices of* $\mathcal{H}$ *is* independent *if it does not contain an edge. An independent set is* maximal *if no proper superset is independent.*

Note that the complement of an independent set in a hypergraph is a transversal. The complements of the maximal independent sets are the minimal transversals.

2.4 Equivalence of MONET and TRANSHYP

We now show the folklore result that MONET and TRANSHYP are polynomial time equivalent and thus are essentially the same problem.

Lemma 2.4.1. *MONET $\equiv_m^p$ TRANSHYP.*

Proof. In order to show the equivalence of MONET and TRANSHYP we first establish "equivalence" of minterms of monotone CNFs and minimal transversals of simple hypergraphs.

Note that a minterm m of an irredundant, monotone CNF ψ is monotone and intersects every clause of ψ since otherwise the assignment m that clearly satisfies m does not satisfy ψ. As minterms are minimal with respect to set inclusion we cannot leave out any variable from m and still have a minterm of ψ.

Now consider the simple hypergraph that corresponds to ψ by viewing it as a set of clauses that itself are sets of variables. Clearly, a minterm m of ψ, interpreted as a set of variables, is a minimal transversal of that hypergraph.

Vice versa, consider a minimal transversal t of the hypergraph corresponding to ψ. As t is a subset of variables that intersects each clause and is minimal with respect to set inclusion, t interpreted as monomial clearly is a minterm of ψ.

We are now ready to show MONET $\leq_m^p$ TRANSHYP. Therefore, we have to give an appropriate reduction function f that maps a MONET instance (φ, ψ) to a TRANSHYP instance $(\mathcal{G}, \mathcal{H})$. We define f to be the (identity) function that constructs hypergraphs from normal forms by viewing normal forms as families of variable subsets. Hence, each monomial of φ forms an edge of $\mathcal{G}$ and each clause of ψ forms an edge of $\mathcal{H}$. From our above discussion of the relationship between minterms and minimal transversals it now follows that $(\varphi, \psi) \in$ MONET iff $\mathcal{G} = \mathrm{Tr}(\mathcal{H})$ which is equivalent to $(\mathcal{G}, \mathcal{H}) \in$ TRANSHYP. As f clearly is polynomial time computable, this establishes MONET $\leq_m^p$ TRANSHYP.

As for the other way around, we analogously construct a reduction function g that maps a TRANSHYP instance $(\mathcal{G}, \mathcal{H})$ to a MONET instance (φ, ψ). We define g to construct a DNF φ from $\mathcal{G}$ in which each edge of $\mathcal{G}$ corresponds to a monomial of φ and, analogously, the clauses of the CNF ψ correspond to the edges of $\mathcal{H}$. Again, it is straightforward that $(\mathcal{G}, \mathcal{H}) \in$ TRANSHYP iff $(\varphi, \psi) \in$ MONET. As g clearly is polynomial time computable, this establishes TRANSHYP $\leq_m^p$ MONET.

Hence, MONET and TRANSHYP are polynomial time equivalent. $\square$

Another example of polynomial time equivalent problems is TRANSHYP$'$ and the computation of all maximal independent sets of a given hypergraph. Here the straightforward reduction just exploits the complement relation between transversals and independent sets. This yields that also TRANSHYP and the problem of deciding whether a given set of independent sets contains all the maximal independent sets of a given hypergraph are polynomial time equivalent using an analogous reduction.

Hence, we have two other ways of formulating Monet, namely in terms of minimal transversals as in the problem TransHyp and in terms of maximal independent sets.

2.5 Basic computational tasks

2.5.1 Preprocessing steps

Before analyzing Monet itself, it is crucial to be able to check several trivial necessary conditions of the input to be a Monet instance. As we later on will show some problems to be logarithmic space solvable, we take special care on the resources needed for the preprocessing and show it only needs logarithmic space. Note that if one of the conditions described in the following is violated, we can immediately reject the input as it cannot be a Monet instance.

We say that an input is *well-formed* if and only if it can be interpreted as a monotone Boolean formula. That means that it is a correct formula with respect to the syntax (Definition 2.2.1). In particular, only variable symbols x_i, "$\wedge$", "$\vee$", "(", and ")" are allowed symbols, the parenthesis structure has to be correct, and the sequence of variables and connectives must not include situations like "$x_1 \vee \vee x_2$" or "$(\vee$" or "$x_1 \wedge x_2 x_3 \vee x_4$" etc.

Lemma 2.5.1. *Whether an input is well-formed can be decided in logarithmic space.*

Proof. Let n be the number of symbols in the input. An appropriate machine checks in a first scan whether only allowed symbols are involved. No additional space is needed.

A second scan of the input is used to examine the parenthesis structure. This can be done using a counter that is initially 0. Whenever a parenthesis is found, the counter is incremented by 1 in case of a left and decremented by 1 in case of a right parenthesis. The parenthesis structure of the input is correct if and only if the counter always contains a value ≥ 0 and the counter equals 0 when the scan is completed. Clearly, the counter is bounded logarithmically.

In a third and last scan of the input a few syntactic rules have to be checked. After the occurrence of a variable only "$\vee$", "$\wedge$" or ")" are allowed. After a left parenthesis only a variable is allowed. After a right parenthesis only "$\wedge$" and "$\vee$" are allowed. After a "$\wedge$" or a "$\vee$" only a variable is allowed. No additional space is needed while scanning the input and checking these syntactic rules. $\square$

We also need a test of being in normal form.

Lemma 2.5.2. *Whether a given well-formed formula is in normal form can be decided without using any additional space.*

Algorithm 1 The irredundancy test

Input: well-formed formula α in normal form with variable set V
Output: Yes, if α is irredundant, and No, otherwise

1: **for all** terms t_i of α **do**
2:     **for all** other terms t_j, $j \neq i$ of α **do**
3:         $\mathit{count} \leftarrow 0$
4:         **for all** variables $x \in t_i$ **do**
5:             **if** $x \in t_j$ **then**
6:                 $\mathit{count} \leftarrow \mathit{count} + 1$
7:         **if** $\mathit{count} \geq |t_i|$ **then**
8:             **output** No and **stop**
9: **output Yes**

Proof. An appropriate machine has to perform two tests. First, it checks whether the parenthesis structure of the input formula is "$()()\dots()$." Secondly, the correct usage of the connectives is tested. The machine checks whether the connectives within a pair "()" are "$\wedge$" in case of checking DNFs (respectively "$\vee$" for a CNF) and whether between any two consecutive pairs of parentheses "$()()$" the connectives are "$\vee$" (respectively "$\wedge$" for a CNF). For both tests the machine does not need any additional space. □

As we only deal with irredundant normal forms, we need a test for irredundancy.

Lemma 2.5.3. *Whether a given well-formed formula α in normal form is irredundant can be decided in logarithmic space.*

Proof. First we check whether no term of α contains a variable twice. This can be implemented in logarithmic space in a straightforward way.

Then we have to check whether no term is contained in another term. A procedure that solves this problem and uses logarithmic space is given as Algorithm 1. The correctness of Algorithm 1 is straightforward. We have to analyze the space requirement.

Let the size n of α be the number of variable occurrences. We assume that α consists of m terms. The first two for-loops need the index of the current terms. Therefore, we assume that the algorithm numbers the terms of α by $1, \dots, m$ and the for-loops test the terms by increasing indices. Hence, the for-loops can count the already tested terms and know the current term. The terms of α need not to be copied to be compared. Both counters are logarithmically space bounded in n.

The third for-loop has to know the index of the current variable. We analogously assume that the variables have indices $1, \dots, |V|$. Hence, the index is logarithmically bounded in n. Two other logarithmically in n space bounded counters are needed for *count* and to evaluate the size $|t_i|$ needed in line 7. □

Finally, we check whether both formulas contain the same variables.

Lemma 2.5.4. *Whether a given irredundant, monotone DNF φ and a given irredundant, monotone CNF ψ have the same variable set V can be decided in logarithmic space.*

Proof. Let the input size n be the number of variable occurrences in φ and ψ. We assume that the variables have indices $1, \ldots, |V|$ and that the formulas are given on two tapes.

The machine just cycles through the DNF φ and tests for each variable occurrence whether this variable is present in ψ (via looking for the index of the variable in ψ from the beginning to the end). When finished, it exchanges the roles and cycles through ψ in the same way. □

Note that in the course of the thesis we do not explicitly check all the conditions of this section as we now know that they can be managed by a logarithmic space preprocessing. Hence, from now on, we assume that MONET instances satisfy all of the above conditions without mentioning it every time.

2.5.2 Basic checks

We now describe other basic checks needed in several of the equivalence test algorithms in the later sections. For example, we often need to check a maxterm condition for clauses. This is important as from the maxterm Definition 2.2.16 it follows that a clause is a *maxterm* of a DNF φ iff it is contained in the irredundant, monotone CNF of φ.

Lemma 2.5.5. *Given an irredundant, monotone DNF φ and an irredundant, monotone clause c, it can be decided in logarithmic space whether c is a maxterm of φ.*

Proof. Let $V = \{x_1, \ldots, x_{|V|}\}$ be the set of variables in φ and c, and $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ be the set of monomials of φ. Let the input size n be the number of variable occurrences in φ and c. We give an algorithm with the desired properties as Algorithm 2. It has to be checked whether c has a non-empty intersection with every monomial of φ (lines 1 to 7). Thereafter, it has to be tested whether $c \setminus \{x\}$ has a non-empty intersection with all monomials for every variable $x \in c$ (lines 8 to 19). If one such variable can be found, then c is not a maxterm. The correctness of Algorithm 2 thus is straightforward. We have to analyze the space requirement.

To know the current monomial, the for-loops in lines 1 and 10 can manage counters that give the number of already checked monomials. These counters have to count till $|M_\varphi|$. Hence, they are logarithmically bounded in n. An analogous argumentation holds for the for-loops in lines 3 and 12. To know the current variable, they manage counters that count till $|m|$ for the current monomial m, which is clearly logarithmic in n. And again, the for-loop in line 8 is handled analogously. Here, the counter has to count till $|c|$, which also is logarithmic in n.

Algorithm 2 The maxterm test

Input: irredundant, monotone DNF φ and clause c
Output: Yes, if c is a maxterm of φ, and No, otherwise

```
for all monomials m_i of φ do
    count ← 0
    for all all variables x_j ∈ m_i do
        if x_j ∈ c then
            count ← count + 1
    if count = 0 then
        output No and stop
for all variables x_i ∈ c do
    hit ← |M_φ|
    for all monomials m_j ∈ φ do
        count ← 0
        for all all variables x_k ∈ m_j do
            if x_k ∈ c and k ≠ i then
                count ← count + 1
        if count > 0 then
            hit ← hit − 1
    if hit = 0 then
        output No and stop
output Yes
```

It remains to check the variables *count* and *hit*. The maximal value of *count* is the size of a largest monomial of φ. Hence, *count* remains logarithmic in n. The maximal value of *hit* is $|M_\varphi|$. Hence, it is also logarithmic in n. ☐

Another test we will use is that of evaluating a DNF under a given assignment.

Lemma 2.5.6. *Given an irredundant, monotone DNF φ and an assignment $\mathcal{A}$, it can be decided in logarithmic space whether $\mathcal{A}(\varphi) = 1$.*

Proof. Let $V = \{x_1, \ldots, x_{|V|}\}$ be the set of variables and $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ be the set of monomials of φ. It has to be tested whether $\mathcal{A}$ contains at least one monomial of φ. An appropriate algorithm is given as Algorithm 3. The correctness of Algorithm 3 is straightforward. We have to analyze the space requirement.

Both for-loops could manage counters that contain the number of the currently tested monomial and the index of the current variable to know which are the current monomial and variable. Such counters stay logarithmic in the input size. The if-test in line 4 does not need any additional space, since it is just a search in $\mathcal{A}$ for the index of the variable. The *eval*-variable only needs constant space. ☐

Finally, we establish the complexity of testing whether a given set is contained in a set family.

Algorithm 3 The DNF evaluation

Input: irredundant, monotone DNF φ and an assignment $\mathcal{A}$
Output: Yes, if $\mathcal{A}(\varphi) = 1$, and No, otherwise

for all $m_i \in M_\varphi$ **do**
    eval $\leftarrow 1$
    for all $x_j \in m_i$ **do**
        if $x_j \notin \mathcal{A}$ **then**
            eval $\leftarrow 0$
    if *eval* $= 1$ **then**
        output Yes and **stop**
output No

Algorithm 4 `isIn`

Input: set $S = \{s_1, \dots, s_{|S|}\}$ of subsets of V and subset $t \subseteq V$
Output: Yes, if $t \in S$, and No, otherwise

1: **for all** $s_i \in S$ **do**
2:     **if** $|s_i| = |t|$ **then**
3:         *isin* $\leftarrow 0$
4:         **for all** $x \in t$ **do**
5:             **if** $x \in s_i$ **then**
6:                 *isin* $\leftarrow$ *isin* $+ 1$
7:         **if** *isin* $= |s_i|$ **then**
8:             **output** Yes and **stop**
9: **output** No

Lemma 2.5.7. *Given a set S of subsets of V and a subset $t \subseteq V$, it can be decided in logarithmic space whether t is contained in S.*

Proof. The algorithm is given as Algorithm 4. The correctness of Algorithm 4 is straightforward.

The for-loops need two logarithmic counters to count till $|s_i|$ and $|t|$. The if-test in line 2 can be implemented using two other logarithmic counters. The *isin*-variable needs logarithmic space as well, since the largest value stored is $|t|$. □

Chapter 3

Applications

In this chapter we give an overview of problems that are equivalent to Monet or that are strongly related to Monet in the sense that Monet techniques can be applied to solve them. We also discuss generalizations of Monet and its variants.

Our idea is to give some further motivation for the importance of Monet by showing the broad range of problems where results on Monet are applicable and, hence, what different fields can benefit from new insights. Thereby, the chapter is not intended to contain all the details and proofs nor do we formally define all the terms from the respective fields. Instead, we would like the chapter to be a bibliography of possible application areas of Monet algorithms and thus we prefer just giving pointers to the original literature where any details can be found. We think this chapter's compilation of applications might be very useful as a new brief survey consisting of a state-of-the-art reference list to Monet applications that complements older lists such as the ones in [BEGK02a, CH07, EG95, EG02, Eit91, Got04, FGBS96].

The chapter is organized as follows. It contains alphabetically ordered sections for each field where Monet applications can be found. In each section, the problems themselves are also ordered alphabetically. Of course, sometimes the classification of the problems is a matter of taste as some fields may overlap, e. g., artificial intelligence and logic or artificial intelligence and machine learning.

3.1 Artificial intelligence

Abduction Abduction is a tool of common-sense reasoning that helps in finding explanations or missing knowledge. More formally, given the characteristic set of a Horn theory Σ—the *background theory*—, a literal q — the *query*—, and a subset A of all literals, the problem is to find all explanations for q with respect to A. Here, a logical theory is a set of formulas. It is *Horn*, if it is a set of clauses having at most one positive literal each. An *explanation* is a minimal set E of literals from A such that $\Sigma \cup E$ is satisfiable and implies q, which means that any model of $\Sigma \cup E$ also is a model of q. Here, a *model* of Σ is a satisfying assignment.

The above described variant of abduction can be shown to be equivalent to the computational variant of Monet in its hitting set formulation [EG02, EM07] and hence algorithms based on hitting set computation exist [SU06].

Horn envelope The *Horn envelope* of a propositional theory Σ is the strongest (with respect to implication) Horn theory Σ' such that Σ implies Σ'. It can be shown that, given the models of Σ, computing all prime clauses of its Horn envelope is equivalent to the computational variant of Monet in its transversal hypergraph formulation [EG02, Got04, Kha95, Kha96, KPS93].

Knowledge recompilation It can be shown that the task of computing all maximal subsets of a Horn theory that are consistent with a new Horn formula is a generalization of the computational variant of Monet in its transversal hypergraph formulation [GPS98].

Knowledge representation Traditionally, knowledge in artificial intelligence is represented by formulas and the notion is that all logical conclusions are accessible to some agent knowing the formulas. In [KKS93, KR96] it was proposed to use as an alternative the representation by characteristic models instead of formulas. *Characteristic models* are models that are not the intersection of other models. In case of Horn formulas, Khardon showed that switching between the representations of a Horn CNF by its characteristic models and by all its prime implicates is equivalent to the computational variant of Monet in its transversal hypergraph formulation [Kha95, Kha96].

Another means of knowledge representation is the formalism of conceptual graphs that emerged from semantic networks. These structures can be extended to conceptual graph assemblies by adding hypergraphs to the conceptual graphs. In case that the hypergraphs have to be given implicitly, it is shown in [CC07, Cro06] that this can be modeled by hypergraph transversals.

A further notion related to knowledge representation is that of version spaces. For a class C of concepts, the subset of concepts consistent with a set of given positive and negative examples is called a *version space*. The version space is *converged* if there is exactly one concept from C consistent with the examples. Deciding convergence of monotone version spaces is equivalent to Monet [HMP04].

In case that only approximation of knowledge in form of a theory is possible, there are several papers relating model-preference default reasoning to the computational variant of Monet in its transversal hypergraph formulation [KPS93, SK90, SK96, Zan02].

Model-based diagnosis Roughly speaking, model-based diagnosis is the question why and when some system may not behave as it should. Given some system description, the set of components of the system and some observations on the systems behavior, model-based diagnosis tries to find minimal consistent states of the system with respect to the observations. This problem and a first solution based on hitting set computation was first reported in [Rei87]. Note that this first solution was improved later on [GSW89, Wot01]. Other hitting set enumeration algorithms for model-based diagnosis can be found in [Fer05a, FV04, FV05, Hae98, Rym92].

Actually, the computational variant of MONET in its hitting set formulation is equivalent to model-based diagnosis [EG91, EG95, Nic02]. A more general version of diagnosis is strongly related to prime implicant computation [KMR92].

Examples of hitting set based methods for diagnostics can be found in reliability estimations [AK03], configuration problems [FFJS04], fault diagnosis for earth movers [FVB+02, FVB+03a, FVB+03b, FVBM02], debugging [PDA93a, PDA93b], and disorder diagnosis [VO00a].

Propositional circumscription The decision version of propositional circumscription asks, given a CNF and a model, whether the model is minimal for the CNF. This problem was first studied by Cadoli [Cad92]. As for monotone inputs it can be shown to be equivalent to MONET in its transversal hypergraph formulation [DH08].

Type error diagnosis Type error diagnosis is the task of generating an explanation for some error. It requires finding all minimal unsatisfiable subsets of a given set of constraints (representing the error) which can be managed via solving the computational variant of MONET in its hitting set formulation [BS05, LS08, KLM06].

Universality of cognitive structures Given a cognitive structure, the universality problem is to decide whether the structure responds to specific instances. This problem is important to decide the response-ability of the structure. It can be solved using the dualization formulation (cf. the corresponding paragraph in Section 3.14) of MONET [PCPO02].

3.2 Combinatorial optimization

Covering problems Associate with a simple hypergraph $\mathcal{H}$ consisting of m edges and v vertices the incidence matrix $M_{\mathcal{H}}$ that contains as rows the characteristic vectors of the edges of $\mathcal{H}$. Then the computational variant of MONET in its transversal hypergraph formulation is equivalent to finding feasible solutions of the following set-covering problem [BS87, BS88]:

$$\min\{x \ : \ M_{\mathcal{H}} \cdot x \geq e_m, \quad x_j \in \{0,1\}, \quad j = 1, \dots, v\},$$

where e_m is the vector consisting of m ones. The characteristic vectors of the transversals of $\mathcal{H}$ are the feasible solutions of the above set-covering variant.

A similar problem of computing minimal coverings can also be found in [Law66].

3.3 Computational biology

Computing cut sets Cut sets in metabolic networks are strategies that block a given set of reactions. Thereby a reaction is *blocked* if it cannot operate in a steady state. Minimal cut sets can be computed from the elementary modes of the network which turns out to be equivalent to transversal hypergraph generation [HKS08, IB07, KG04, Kla06, KSG07]. Of course, one can imagine similar tasks for different types of networks (not necessarily metabolic) where again transversal computation proves useful [Hau08].

Modeling reaction pathways In [EL03] methods related to evaluating networks of chemical reactions are described. Several of these methods require computation of minimal sets and the authors suggest to use algorithms solving the computational variant of Monet for this purpose.

Phylogeny reconstruction Methods for a restricted computational Monet variant in the setting of computing all minimal transversals up to a given size can be applied to almost-perfect phylogeny reconstruction [Dam06]. Almost-perfect phylogeny reconstruction is the problem of computing a matrix that describes a phylogenetic tree from a given matrix using row deletions, column deletions, or bit flips in the given matrix.

Reconstruction of unknown mixtures of proteins Damaschke describes a method for concluding a set of causes from an observed set of effects [Dam07]. His method is based on the computational variant of Monet in its hitting set formulation and may for instance be applied to the task of determining which proteins are present in a mixture of proteins using peptide mass fingerprinting and a database of mass spectra.

3.4 Computational geometry

Bodies Let $A \subseteq \mathbb{R}^n$ be a subset of points in $\mathbb{R}^n$ and z be a point in $\mathbb{R}^n$. A *body* of A then is a minimal subset $X \subseteq A$ containing z in its convex hull. Generating all bodies can be shown to be a generalization of the computational variant of Monet in its transversal hypergraph formulation [KBEG08b]. A related question on the number of bodies compared to the number of maximal sets that do not have z in their convex hull is discussed in [CKK02] for the special case of $n = 2$.

Maximal k-boxes Given an anti-monotone property π and a subset of the maximal elements of some vector space satisfying π, the problem of computing a new maximal element or state that the given set contains all of the maximal elements can be used to compute maximal k-boxes. Such k-boxes are useful in several data mining scenarios as well as in computational geometry. Solving the maximal vector problem can be shown to be equivalent to the computational variant of Monet in an independent vector formulation [KBE+07].

3.5 Computational medicine

Optimal vaccination strategies Given a subset of initially infected individuals from a population of individuals and assumptions about disease transmission, the task of computing inclusion minimal vaccination strategies can be solved using the computational variant of Monet in its transversal hypergraph formulation [BG07b].

3.6 Cryptography

Attacking anonymity Anonymity protocols are useful to hide peer partners in communications. In [KP04] an attack on a generic anonymity protocol is used to identify all the peer partners of one participant. The attack uses the FK-algorithm B (cf. Section 5.3) for the computational variant of Monet in its hitting set formulation.

3.7 Databases

Armstrong relations A *relational schema* R consists of a set U of *attributes* and a set F of *functional dependencies* between the attributes. The *closure* F^+ of F contains all dependencies that follow from F. We say that a relational instance r over U is an *Armstrong relation* for R iff the functional dependencies holding in r are exactly the functional dependencies in F^+. Armstrong relations are especially useful in database design as they show, on the instance level, which functional dependencies hold and provide this information in a human-readable format to a database designer. There is a close relationship of computing Armstrong relations to Monet-like problems [DT95, GL90, KMR99, MR86]. Computing an Armstrong relation for a given set of functional dependencies in so-called *Boyce-Codd Normal Form (BCNF)*, where all the left hand sides of the functional dependencies are keys, can be shown to be equivalent to the computational variant of Monet in its hypergraph saturation (cf. the corresponding paragraph in Section 3.12) formulation [EG91, EG95].

Identify keys Keys play an important role in databases as they minimally identify tuples stored in the relations. Identification of whether a given relation is in a desirable normal form for relations can often be done knowing all the keys for the relation, e. g., deciding if it is in BCNF. Again the problem of finding keys is, just like identifying Armstrong relations, strongly related to Monet [Dem80, DT87, DT99, GMS97, Thi86, TS05] and the decision problem—given a relation instance and a set of keys for it, decide if there is another key—can be shown to be equivalent to the complement of Monet in its hypergraph saturation formulation [EG91, EG95].

Inclusion dependencies Inclusion dependencies are a generalization of foreign keys in the relational model. They convey data semantics in an interrelational manner, e. g., describing the fact that all values of some attributes in one relation are a subset of the values of some attributes of another relation. Computing such inclusion dependencies can be done using the computational variant of Monet in the transversal hypergraph formulation [MP03].

Inferring functional dependencies The problem of inferring functional dependencies from a given relation instance is equivalent to generating Armstrong relations as it is just the opposite problem. Thus, it also has many applications in database design and is strongly related to Monet [DT95, GL90, Got04, IKM99, IKM03, KM95, Koe05, Koe08, LPL00, MR86, MR87, MR92a, MR92b, MR94, WGR01]. The decision version—given relation instance r and a set F of functional dependencies in BCNF, decide if the functional dependencies holding in r equal F^+—is equivalent to Monet in its hypergraph saturation formulation [EG91, EG95].

Query processing If a query to a database fails, the user might gain more information by not just getting the empty set as an answer but having some information about what causes the query to fail. An appropriate method to solve this issue is to look for minimally failing subqueries of the original queries. This problem then turns out to be solvable by using the computational variant of Monet in the transversal hypergraph formulation [God97].

Query rewriting Knowing what causes a query to fail in the sense of computing the minimally failing subqueries (discussed in the previous paragraph) also helps in rewriting the query [God97].

Another technique for rewriting queries based on database views also is equivalent to the computational variant of Monet in its transversal hypergraph formulation [JBPT06].

Reducts in rough sets Rough sets theory aims at approximating concepts from data. An important tool are *reducts* that can be informally described as subsets of attributes in a data base that do not introduce additional inconsistencies in the data. They can therefore be used to detect redundant attributes and thus to achieve reduction in the number of attributes. As shown by Popova, reducts can be derived as transversals from a special matrix associated with the data [Pop04]. Thus, computing reducts is equivalent to the computational variant of Monet in its transversal hypergraph formulation. Another possibility of showing the equivalence is via difference functions as described in [Jär00].

View update Views are an important means of presenting data from databases to users. In case of database updates also views have to be updated. This can be managed using the computational variant of Monet in its hitting set formulation [AB00, Tom88].

3.8 Data mining

Association rules Association rules in databases are statements of the form "if item x is present in a tuple than in 95% of the tuples also y is present" and, thus, are very important for different data mining tasks. A crucial step in mining association rules is computing frequent and infrequent sets [AMS+96, AIP04] and as this task is equivalent to the computational Monet variant (see the corresponding paragraph on frequent item sets below), also association rule mining is strongly related to Monet. Even more general rules than just association rules can be found using frequent sets where again Monet techniques apply [MT96a, RC06].

Closed itemsets The *closure* of an itemset I is the set of all items that appear together with I in all tuples of a database. *Closed itemsets* are itemsets that equal their closure; the minimal itemsets whose closure equals an itemset Z are the *generators* of Z. Association rules where the left hand side is the generator of the right hand side are especially interesting. Minimal generators of closed itemsets are computable using the computational variant of Monet in its transversal hypergraph formulation [BG07a, Gar06, PT02].

Coloring data clusterings To usefully visualize information gained by different clustering results in data mining applications, one approach is to attribute colors to the clusters in a way that similar clusters get the same color. The authors of [DCS06] describe a method of color assignment based on the computational variant of Monet in its transversal hypergraph formulation.

Dualization of discrete functions Discrete functions are an important means for analyzing multi-attribute data sets. Bioch shows that for dualizing positive discrete functions techniques for solving the computational variant of Monet in its dualization formulation (cf. the corresponding paragraph in Section 3.14) can be applied [Bio98].

Emerging patterns Given two data sets, *emerging patterns* are those itemsets whose *support* (the number of occurrences in the data sets) differs substantially between the given data sets. Emerging patterns are useful for, e. g., classification. Computing emerging patterns can be done applying the computational variant of Monet in its transversal hypergraph formulation [BMR03, DL05, Man04, RB03, RF07]

Extracting semantics from data cubes The cube lattice framework is a search space for multidimensional data mining. One way of extracting semantics from cube lattices is that of cube transversals that turn out to be a special case of hypergraph transversals [CCL03]. Hence, techniques for the computational variant of MONET in its transversal hypergraph formulation are applicable to the computation of cube transversals.

Formal concept analysis *Concept lattices* are complete lattices generated by closure operators (cf. the above paragraph on closed itemsets). They play a role in knowledge discovery as a whole [PT02], especially in mining causal dependencies [Pfa06], and generating rule based world-views from data given in a relational database [Pfa07]. Due to its relation to closed sets also formal concept analysis can benefit from MONET solving approaches.

Frequent itemsets Given a binary matrix M, a subset C of its columns is called *t-frequent* if at least t rows of M only contain non-zero entries in the C-columns. Otherwise, C is *t-infrequent*. Of special interest are the maximal t-frequent and the minimal t-infrequent subsets. Computing both, the maximal t-frequent and the minimal t-infrequent subsets is equivalent to the computational variant of MONET in its transversal hypergraph formulation [BGKM03]. As the notion of frequent sets can be easily extended to arbitrary databases, MONET solving techniques are very important for computing frequent and infrequent itemsets in data mining [Afr06, Bay98, FMP04, GKM+03, GKMT97, GMS97, Got04, KISI00, Man05, Mis02, MT98, STT98, SU03, Toi96a, Toi96b].

A generalization of maximal frequent sets to so-called intervals can also be shown to be solvable using MONET techniques [Elb06a].

Interesting sentences A generalization of frequent sets are *interesting itemsets*, where there is some interestingness predicate instead of just frequency. The most important interesting itemsets are the maximal interesting itemsets. Computing these maximal interesting sets can be managed using algorithms for the computational variant of MONET in its transversal hypergraph formulation [MFP04, MT96b]. Actually, the problems can be shown to be equivalent [GKMT97, MT97].

Sequential patterns Sequential pattern mining can be seen as a way of generalizing association rules. An example is to derive rules like "50% of the customers that buy a bike and biking clothes, then buy clipless pedals and appropriate biking shoes, then buy a GPS, then buy ..." that cannot be gained using association rule mining. An ordered list of itemsets forms a *sequence*. All the transactions performed by a customer form this customer's *customer sequence*. A customer *supports* a sequence iff this sequence appears as a subsequence in his customer sequence. The problem of mining sequential patterns is the problem of finding all the maximal sequences that have at least a given support. Thus, this problem is similar to mining frequent itemsets. And not too surprisingly, it can also be solved using the computational variant of Monet in its transversal hypergraph formulation [Li06, LLT07].

3.9 Distributed systems

Synchronization Synchronizing a distributed system without guaranteed communication is an important task that can be solved by mutual exclusion. A key concept to realize mutual exclusion is that of *coteries*, which are families of incomparable subsets such that every pair of contained subsets has at least one element in common. In case of distributed systems, the subsets are subsets of nodes of the system or, in case of distributed databases, sites of the database. Very desirable in this setting are so-called *non-dominated* coteries, for which no other coterie exists such that each set is contained in some set of the other coterie. Deciding whether a coterie is non-dominated is equivalent to Monet in its self-transversality/self-duality formulation (cf. the corresponding paragraphs in Sections 3.12 and 3.14) and so there is a rich field of applications of Monet [BI95b, BI95c, EG95, GB85, HY05, IK93, JPY88, MK01].

3.10 E-commerce

Best cover The problem of, given a concept description and an allowed terminology, rewrite the concept using only names from the allowed terminology, is known as the *best cover problem*. It has rich applications in e-commerce and can be solved using the computational variant of Monet in its transversal hypergraph formulation [BHL+05, HLRT02a, HLRT02b, HLRT03].

Query rewriting in e-catalogs E-catalogs are an important source of information in the web. Querying them may result in a failure, as it may in the case of "real" databases (cf. the corresponding paragraph in Section 3.7) and again the problem of rewriting the query can be shown to be equivalent to the computational variant of Monet in its transversal hypergraph formulation [BHP+06, BHRT03].

Web service discovery A very important task in the field of web services is to automatically find services that fit user specific interestingness constraints. As this problem is very similar to set-covering in a constrained form, approaches for solving Monet in its transversal hypergraph generation formulation are applicable [Sir04, ZB06].

3.11 Game theory

Nash equilibrium Computing so-called *NE-theorems* and *NE-examples* in bimatrix games is equivalent to deciding whether the game has a Nash equilibrium. Computing NE-theorems and NE-examples can be shown to be equivalent to the computational variant of MONET in its transversal hypergraph formulation [BEG+08].

Nash solvability Given two player's strategies and the possible outcomes, a *game form* is a mapping of the cross-product of the strategies to the outcomes. Thus, a game form can be seen as game without specified payoffs. Nash solvability of a game form is equivalent to its tightness (see paragraph below) and thus is equivalent to MONET in its duality (cf. the corresponding paragraph in Section 3.14) formulation [BGM07, Gur75, Gur88].

Tight game forms A game form can be associated to two DNFs that represent the mapping of the two player's possible strategies to the outcomes of the game. A game form is called *tight* iff these two DNFs are dual and thus deciding tightness is equivalent to MONET in its duality (cf. the corresponding paragraph in Section 3.14) formulation [BGM07, Gur75, Gur88]

3.12 Graph theory

2-colorable intersecting hypergraphs A hypergraph *k-coloring* is an assignment of one of k colors to each vertex such that no edge consists of vertices all having the same color (this means that the sets consisting of vertices having the same color are independent). As for 2-coloring, we obviously have to concentrate on hypergraphs having no singleton edges. 2-coloring a simple, *intersecting* hypergraph (no disjoint edges) can be shown to be equivalent to the complement of MONET in its self-transversality (see paragraph below) formulation [EG95].

Bicritical clutters Clutter is just another name for a simple hypergraph. A clutter $\mathcal{H}$ is *bicritical* iff for every edge e of $\mathcal{H}$, the clutter consisting of the edges of $\mathcal{H}$ disjoint from e, can be colored with two colors less than $\mathcal{H}$. It can be shown that deciding whether a non-trivial bicritical clutter has chromatic number 3 is equivalent to MONET in its self-transversality (see paragraph below) formulation [Ben99].

Convex generators A *convexity space* for a set V is a pair (V, C) where C is a family of subsets of V. For a subset $A \subseteq V$ the *convex hull* of A with respect to (V, C) is the intersection of all sets in C that contain A. A *convex generator* of a graph $G = (V, E)$ is a minimal subset of V such that its convex hull is V. It can be shown that findind the convex generators of a graph is equivalent to the computational variant of Monet in its transversal hypergraph formulation [JD00].

Independent sets A subset of vertices is *independent* in a hypergraph iff it does not contain an edge. Thus, independent sets are the complements of transversals and, hence, computing all the maximal independent sets of a hypergraph is equivalent to Monet in its transversal hypergraph formulation [BEGK00, BEGK04, JPY88, LLK80, MP97].

Minimal contrast subgraphs Given two collections of graphs, a *contrast subgraph* is a subgraph appearing in one collection but not in the other. Especially interesting are the minimal contrast subgraphs. Finding them has applications in graph classification and the like and can be solved using the computational variant of Monet in its transversal hypergraph formulation [TB06].

Multiple and partial transversals Multiple and partial transversals are generalizations of transversals. As for *multiple transversals*, we assign weights to the edges and to hit an edge of weight b a multiple transversal has to contain at least b vertices of that edge. As for *partial transversals*, we have a threshold k given for a hypergraph and "transversals" are allowed to have an empty intersection with at most k edges. The problems of computing all minimal multiple/partial transversals are equivalent to the computational variant of Monet in its transversal hypergraph formulation [BGKM01].

Saturation A hypergraph $\mathcal{H}$ is *saturated* iff every vertex subset is contained in or contains some edge of $\mathcal{H}$. As for simple hypergraphs, it can be shown that testing saturation is equivalent to Monet in its transversal hypergraph formulation [BI95a, EG95].

Self-transversality A hypergraph is *self-transversal* or *strange* iff $\mathcal{H} = \mathrm{Tr}(\mathcal{H})$. Testing whether a hypergraph is self-transversal is equivalent to Monet in its transversal hypergraph formulation [EG95, Sey74].

Spanning subgraphs A graph G is *k-vertex connected* iff every subgraph of G obtained by removing at most $k-1$ vertices is *connected* (there exist paths between any two vertices). A subgraph of G is *spanning* iff it has the same vertex set as G. Given a graph G, generating all its minimal k-vertex connected spanning subgraphs can be managed using techniques solving an easy class (cf. Chapter 4) of Monet in its transversal hypergraph formulation [BBE+07, Bor06].

Total dominating sets Given a graph G, a *total dominating set* is a subset D of vertices such that each vertex in G has at least one neighbor in D. Constructing a hypergraph $\mathcal{H}$ from G containing as edges the neighborhoods of the vertices in $\mathcal{G}$

every minimal transversal of $\mathcal{H}$ is a minimal total dominating set of G. Thus computing all minimal total dominating sets can be managed using the computational variant of MONET in its transversal hypergraph formulation [TY07].

Two-layer planarization A graph is *bipartite* iff its vertex set can be partitioned into two parts such that there are no edges between vertices from the same part. A bipartite graph is *biplanar* if its vertices can be placed on two parallel lines such that no edges cross if drawn straight. The problem of two-layer planarization asks whether a given graph can be biplanarized by deleting k edges. Techniques that can be seen to use the computational variant of MONET in its hitting set formulation can solve two-layer planarization [DFH+01, Fer05b].

Weighted transversals Weighted transversals generalize partial and multiple transversals. Given non-negative weight and threshold vectors for each edge, a *weighted transversal* is a minimal vertex subset that intersects every edge except for a sub-family of total weight not exceeding the given threshold. Generating all weighted transversals is equivalent to the computational variant of MONET in its transversal hypergraph formulation [BGKM04].

3.13 Lattice theory

Implicational basis of lattices Computing a minimum implicational basis of a lattice that is given by the poset of its irreducible elements can be shown to be equivalent to the computational variant of MONET in its database formulation of computing functional dependencies [CM03, JN06]. The key idea is that the implicational basis then forms a functional dependency cover. There exist several restrictions where the basis computation can be done in output-polynomial time, namely locally distributive lattices [Duq91], $\wedge$-semidistributive lattices [JN06], and modular lattices [Wil00].

Independent elements in products of lattices Given the product $\mathcal{L}$ of n lattices, a set A in this product, and a partial list of maximal independent elements of A in $\mathcal{L}$, the task is to either find a new maximal independent element of A or to decide that there is none. This problem is a generalization of MONET in its transversal hypergraph formulation [Elb02a] but MONET techniques may also be useful for appropriate algorithms.

Very similar is the following problem. We are given the product $\mathcal{P}$ of n posets, where the precedence graph of each poset is acyclic and either the in-degree or the out-degree of each element is bounded. Given a set $A \subseteq \mathcal{P}$, it can be shown that computing the set of maximal independent elements of A in $\mathcal{P}$ is a generalization of the computational variant of MONET in its independent set formulation [Elb02c].

3.14 Logic

Bidual Horn extensions A mapping $f : \{0,1\}^n \to \{0,1\}$ is said to be a *Boolean function* f on n variables. Note that any Boolean function can be expressed by a Boolean formula and that each vector from $\{0,1\}^n$ straightforwardly corresponds to an assignment for any Boolean formula expressing f. The *dual* f^d of a function f is defined to satisfy $\mathcal{A}(f^d) = \overline{\mathcal{A}}(\neg f)$ for every assignment $\mathcal{A}$. A function f is *bidual Horn* iff f and f^d are Horn. A *partially* Boolean function is given by a disjoint pair (T, F) of subsets of $\{0,1\}^n$ that contain vectors where the function is supposed to be `true` resp. `false`. The extension problem for a given partially defined Boolean function is checking whether it is interpolated by a function f from a given class of total Boolean functions, and computing a formula for f. It can be shown that bidual Horn extension is output-polynomial if and only if the computational variant of MONET in its dualization formulation (see below) is output-polynomial [EIM99].

Canonical decomposition A monotone function f (which can be expressed by a monotone formula) is *dual-minor* iff for all assignments f's truth-value is at most equal to the truth-value of f^d. It is *self-dual* iff $f = f^d$. As for a dual-minor function, it is interesting to decompose it into a conjunction of monotone self-dual functions. If all these self-dual functions satisfy special properties, we have a so-called *canonical decomposition*. It can be shown that finding canonical decompositions of positive dual-minor functions is equivalent to MONET in its duality formulation (see below) [BIM99].

Dualization Given two irredundant, monotone DNFs, decide if they are dual, is equivalent to MONET [EG02, FK96, Got04]. As follows, the computational variant—given an irredundant, monotone DNF, compute the dual irredundant, monotone DNF—, known as dualization, is equivalent to the computational variant of MONET.

There is also a generalization of this MONET version. Namely dualization of disguised bidual Horn formulas. A formula is *disguised bidual Horn* iff it becomes bidual Horn after renaming some variables. Dualization of disguised bidual Horn formulas clearly is a generalization of the computational variant of MONET in its dualization formulation. It can also be managed in output-quasi-polynomial time [EIM02].

Inner-core and outer-core functions Given sets T and F of a partial function and an integer k the *k-inner-core* and *k-outer-core* functions describe the functions that are immune against flipping at most k assignments from T to F or vice versa. For the class of monotone functions, computing the maxterms and the minterms of inner-core and outer-core-functions in output-polynomial time is equivalent to finding an output-polynomial algorithm for the computational variant of MONET in its dualization formulation [MI99].

Interior and exterior functions The *k-neighborhood* of an assignment $\mathcal{A}$ for a Boolean function contains all assignments that differ from $\mathcal{A}$ in at most k variables. For a non-negative integer k the *k-interior* function of a Boolean function f is defined to be `true` for each assignment whose k-neighborhood only contains satisfying assignments of f. Analogously, the *k-exterior* function is defined to be `true` for each assignment whose k-neighborhood contains a satisfying assignment of f. Given the irredundant DNF of a monotone function f and an integer k, the problems of computing all min- and maxterms of the k-exterior function of f or all minterms of the k-interior function of f in output-polynomial time is equivalent to give a polynomial algorithm for MONET in its duality formulation [MI96, MOI03].

Maximal models Given a CNF consisting only of negative literals, computing all its maximal models can be shown to be equivalent to the computational variant of MONET in its transversal hypergraph formulation [KSS00, Sta01].

Proof systems A *proof system* for a language L is a verifying algorithm V running in polynomial time and for all x we have $x \in L$ iff there is some advice string p such that V on input x and p accepts. Associated to a proof system V is a function $f_V(n)$, which is defined as the maximum of the minimal advice lengths for any $x \in L$ with $|x| = n$. A proof system is *polynomially bounded* iff $f_V(n)$ is polynomial. Associated to the open complexity of MONET in its transversal hypergraph formulation is the question of whether or not tree proofs are a polynomially bounded proof system for the transversal hypergraph language [Pit02].

Satisfiability variants The satisfiability problem can be formulated as the problem of deciding whether a given CNF is satisfiable. This problem is NP-complete. There are several restricted variants related to MONET.

IMSAT (intersecting monotone satisfiability) is a satisfiability variant where each clause of the CNF is restricted to either contain only positive or only negative literals and each positive clause has an nonempty intersection with each negative clause. This problem is equivalent to the complement of MONET in its transversal hypergraph formulation [EG95, EG02, Got04].

Associated with a clause set is its *conflict multigraph* that has a vertex for each clause and as many (parallel) edges connecting two vertices as the clauses have *conflicts* (variables that appear positive in one and negative in the other clause). A CNF is *bi-hitting* iff the conflict multigraph associated to the CNF's clause set is a complete bipartite multigraph (where every pair of vertices from different parts is connected by at least one edge). It can be shown that satisfiability of bi-hitting clause sets is equivalent to the complement of MONET in its transversal hypergraph formulation [GK04a].

NAESPI (not all equal satisfiability with positive literals and intersection) is a satisfiability variant where the CNF is restricted to be monotone and all clauses must have a nonempty intersection. But in this case we do not just ask for a satisfying assignment (which would be a trivial task) but for a satisfying assignment not completely containing any clause. This problem is equivalent to the complement of MONET in its self-duality (see paragraph below) formulation [GK04b].

Self-duality An irredundant, monotone DNF is self-dual iff it is identical to its dual irredundant, monotone DNF. It can be shown that deciding self-duality of an irredundant, monotone DNF is equivalent to MONET [BI95a, Dom97, EG95, GK04b, GK07, GM08].

Unknown assignment Given subsets T of the minterms and F of the maxterms of a monotone function f, the problem of finding an *unknown* assignment that is not superset of any $t \in T$ nor subset of any $f \in F$ can be shown to be equivalent to the computational variant of MONET [BI95a, GK99, MI97].

The corresponding decision version asks for a given partial function whether in fact it is a total function.

3.15 Machine learning

Classification Finding minimal feature sets that can be used as classification rule templates and preserve a given classification of objects is an interesting task in machine learning. In [Vin99b, VØ00b] a method for computing such classification rules is described that uses the computational variant of MONET in its hitting set formulation. These techniques can also be applied to rough sets (cf. the corresponding paragraph in Section 3.8).

Kernel rules The input of a typical machine learning scenario is a training set of class-labeled examples where examples are described by the assignment of values to a set of attributes. A *partial description* is an instantiation of a subset of attributes. A *rule* is a partial description such that all examples in the training set that agree with the partial descriptions' instantiated variables have the same class-label. If no subset of a rule also is a rule, we say that it is a *kernel rule*. It can be shown that kernel rules can be computed using prime implicant generation and thus in monotone settings algorithms for the computational variant of MONET are applicable [Rym94b].

Learning monotone formulas Given an oracle for a monotone formula that answers questions on formula values, learning the formula's irredundant, monotone DNF and CNF is equivalent to the computational variant of MONET [BHIK97, BI95a, DMP99, GKMT97]. Using other oracles or slighly changing the setting, there are many more MONET like problems in the field of learning monotone formulas [Ang88, BCG+96, BD96, Tor01, TT01, TT02].

3.16 Mathematical programming

Enumerate solutions We are given a system $Ax \geq b$ of r linear inequalities in n integer variables, where A is a real $r \times n$-matrix, b is a real r-vector, and $0 \leq x \leq c$ for some non-negative n-vector c. A vector x is a *feasible* solution iff $Ax \geq b$. The system is monotone iff for every feasible solution x all $y \geq x$ are also feasible. The problem of generating all minimal feasible solutions of a monotone system can be solved using techniques for the computational variant of MONET in its transversal hypergraph formulation [KBEG08a]. In case of a binary matrix A and all-ones vectors b and c even equivalence can be established [BEG+02].

Feasibility The chromatic number of a hypergraph can be computed by a linear relaxation of an integer program. Determining the feasibility of a certain point in the polytope associated with this linear program can be shown to be equivalent to MONET in its self-duality formulation [GM08].

Irreducibly infeasible subsystems Given an infeasible system $Ax \leq b$, finding all minimal infeasible subsystems is the question of computing all subsystems whose deletion would cause the resulting system to be feasible. This problem can be shown to be solvable using techniques for the computational variant of MONET in its transversal hypergraph formulation [Pfe02].

Polyhedral cones Given a family $\mathcal{K}$ of polyhedral cones and a vector b in their sum, computing all the minimal subsets of $\mathcal{K}$ whose sum contains b and all the maximal subsets of $\mathcal{K}$ whose sum does not contain b is equivalent to the computational variant of MONET in its transversal hypergraph formulation [Kha00].

Polyhedron representation The problem of, given a bounded polyhedron P by a system of linear inequalities $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ and a subset X of P's vertex set, decide whether X is the complete vertex set of P, is known as the *polytope-polyhedron problem*. This problem is essential in switching between polyhedron representation from inequalities to vertices. The polytope-polyhedron problem is very similar to MONET in its transversal hypergraph formulation [Lov92]. For some (restricted) computational variants—find all polyhedron vertices—algorithms for the computational variant of MONET in its transversal hypegraph formulation can be applied [BEGM07].

3.17 Matroid theory

Cut-conjunctions Given a binary matroid M on ground set E and a subset $B \subseteq E$, generating all maximal sets $X \subseteq S \setminus B$ that span no element in B can be shown to be a generalization of the computational variant of Monet in its transversal hypergraph formulation [KBB+08]. Here, $X \subseteq E$ *spans* a $b \in B$ if $r(X) = r(X \cup \{b\})$, where $r : E \to \mathbb{Z}^+$ is the rank function of M. A special case of this matroid problem is generating cut-conjunctions in graphs that is finding egde sets whose deletion causes a set of source-sink pairs to be disconnected.

Independent sets Given matroids $M_1, \ldots, M_m$ on a ground set V, computing all maximal subsets of V that are independent in all the matroids is a special case of a problem for polymatroid functions that then turns out to be solvable using Monet techniques.

Let V be a finite set and f be a function mapping subsets of V to natural numbers. We say that f is *monotone* if $f(X) \leq f(Y)$ for $X \subseteq Y$; *submodular* if $f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$ holds for all subsets $X, Y \subseteq V$; and *polymatroid* if f is monotone, submodular and $f(\emptyset) = 0$. Given a system $f_i(X) \geq t_i, i = 1, ..., m$ of polymatroid inequalities, computing all minimal feasible solutions can be shown to be solvable using techniques for the computational variant of Monet in its transversal hypergraph formulation [BEGK02b, BEGK03a, BEGK03b].

Non-spanning subsets For a subset X of the ground set E of a matroid M let $span(X)$ denote the set of all elements of E spanned by X. Given a matroid M on E and two nonempty disjoint subsets $A, D \subset E$, the task of computing all maximal subsets $X \subseteq D$, such that $span(X) \cap A = \emptyset$, can be shown to be a generalization of the computational variant of Monet in its transversal hypergraph formulation [KBE+05].

Spanning and connected subsets A subset X of the ground set E of a matroid M is said to be *connected* iff for every pair of distinct elements x, y of X there is a circuit C of M such that $X \supseteq C \supseteq \{x, y\}$. X *spans* the matroid M if $r(X) = r(E)$, where $r : E \to \mathbb{Z}^+$ is the rank function of M. Given a matroid M, the task of enumerating all minimal spanning and connected subsets can be shown to be a generalization of the computational variant of Monet in its transversal hypergraph formulation [KBB+06].

3.18 Mobile communication systems

Channel assignment Given a cellular system, cells that are within a certain range to each other may not use the same channel due to interference problems. Contrariwise, cells that are sufficiently apart from each other may use the same channels to reduce used frequencies. The problem of assigning channels to cells can be shown to be solvable using the computational variant of MONET in its independent set formulation [MS94, SS98].

3.19 Reliability theory

Safeguard sensitive data Given a sensitive database, the task of safeguarding it consists of ensuring its privacy and its longevity. The search for a “best” system to safeguard sensitive data under certain metrics can be managed involving a step that uses the computational variant of MONET [MGM06].

3.20 Semantic web

Semantic composition in e-learning Given a repository of lecture subparts and a user’s request, the problem of personalized learning involves retrieving subparts that match the user’s query. Solving this problem involves computing concept coverings that are closely related to the notion of best covers (cf. the corresponding paragraph in Section 3.10). And again, computing concept covers can be managed using an algorithm for the computational variant of MONET in its transversal hypergraph formulation [KLM07].

3.21 Software engineering

Debugging UML diagrams UML is the standard modeling language in software engineering. In debugging UML diagrams important tasks are automated contradiction detection and repair. As for computing minimal sets of changes in UML diagrams in order to remove contradictions, an algorithm for the computational variant of MONET in its hitting set formulation can be used [SKU06].

Revising specifications In software projects the development of code is based on specifications that state how the final programs should operate. Such specifications are not static as customers may change their mind during the project. Whenever a new specification is added to the current specification and causes conflicts, these conflicts have to be resolved. The task of enumerating minimally revised specifications can be shown to be solvable using an algorithm for the computational variant of MONET in its hitting set formulation [SU05].

3.22 Topology

Homology groups A *simplicial complex* (V, Δ) can be seen as a special type of hypergraph with the property that when $e \in \Delta$ is an edge, then so are all of e’s subsets. Edges are called *faces* in this setting. It can be shown that computing homology groups of finite simplicial complexes from their duals is equivalent to the computational variant of MONET in its transversal hypergraph formulation [DHSW03].

Non-faces The *non-faces* of a simplicial complex are subsets of vertices not contained in Δ. For special simplicial complexes, namely so-called shellable simplicial complexes, computing all minimal non-faces can be solved using methods for an easy class of the computational variant of MONET in its dualization formulation [BCE+00, Pfe02].

3.23 XML

XML functional dependencies Just like in case of relational databases, functional dependencies can also be defined for XML. As such XML functional dependencies are closely related to relational functional dependencies, computing XML functional dependencies can be solved using algorithms for the computational variant of MONET in its transversal hypergraph formulation [Tri08].

Chapter 4

Easy Classes

In this chapter we examine easy classes of Monet. These are restrictions of the input DNF that allow for polynomial time algorithms solving respective Monet instances. We focus on restrictions of the DNF only as they can be easily transformed to restrictions of the CNF by exchanging the connectives $\wedge$ and $\vee$ and thus a role change of the formulas. The study of easy classes is an important branch of research concerning Monet. One obvious reason is that it allows specific classes to be solved faster than with any known algorithm for the general problem. Thereby, the easy classes reveal information about the really hard parts of the problem. Another reason why easy classes are interesting is the question for hardness or lower bounds for the problem Monet itself. Unfortunately, no such results are known yet. But a first step in this direction is a thorough analysis of the resources needed to solve even easy classes.

Our intention is twofold. First, of course, we want to demonstrate the wide range of polynomial solvable subclasses of Monet. Our second goal then is to examine the question of how easy the easy classes really are. All that is known so far are polynomial time bounds for all the easy classes as this was sufficient to prove their "easiness." But there are no known lower bounds (not even logarithmic space). This situation is not satisfying in the course of hardness analysis of the easy classes or Monet itself.

We show that some of the easy classes are "easier" than expected, as we will show them to be solvable with logarithmic space only, improving the previously known polynomial time bounds.

The discussed restrictions can be divided into two groups. The first group comprises restrictions on the size of the DNF in the broader sense, like the number of monomials or their size. The second group contains rather structural restrictions of the DNF, like being regular, aligned or 2-monotonic.

The chapter is organized as follows. In Section 4.1 we examine restrictions on the size of the DNF, whereas Section 4.2 is dedicated to the more structural restrictions. Some concluding remarks follow in Section 4.3.

4.1 Restrictions on the size of the DNF

In this section we examine restrictions of the DNF φ of a MONET-instance (φ, ψ) that concern the size of φ in a broad sense.

4.1.1 The DNF contains only a constant number of monomials

First, we restrict the DNF to contain only a constant number of monomials. The corresponding restricted MONET version is $\text{MONET}_{\text{cnm}}$ (MONET with a constant number of monomials).

$\text{MONET}_{\text{cnm}}$:	*instance:*	irredundant, monotone φ in DNF and ψ in CNF, where φ contains only c monomials for constant c
	question:	are φ and ψ equivalent?

$\text{MONET}_{\text{cnm}}$ is known to be decidable in polynomial time $O(n^c)$ [EG91]. We improve this bound to logarithmic space.

Theorem 4.1.1. $\text{MONET}_{\text{cnm}}$ *is decidable in logarithmic space.*

Proof. Let (φ, ψ) be a MONET-instance of size n and $V = \{x_1, x_2, \dots, x_{|V|}\}$ be its variable set. We describe the work of an appropriate machine.

Note that checking whether φ contains only a constant number of monomials requires only logarithmic space as monomial counting suffices. Let c be the constant bounding the number of monomials of φ.

The machine now has to perform the equivalence test of φ and ψ. It systematically generates candidates for clauses of a CNF equivalent to φ (the first candidate consists of the first variables from each monomial; the second candidate consists of the second variable from the last monomial and the first variables from all the other monomials; [...]; the last candidate consists of the last variables from all monomials). There are at most $|V|^c$ possible candidates and the machine counts the already tested ones. This counter is logarithmic in n. By counting the already tested candidates the machine knows which is the next candidate because of the systematic generation. Since a candidate consists of at most c variables and since an index of one variable has size $\log |V|$, the machine could write down the indices of the variables forming the current candidate in space $\leq c \log |V|$ which is clearly logarithmic in n. For each such candidate the machine checks whether it is a maxterm of φ using Algorithm 2 as a subprocedure. For candidates that are maxterms, the machine has to ensure that they are included in ψ, since otherwise φ and ψ cannot be equivalent. This is done using Algorithm 4 from Lemma 2.5.7 as a subprocedure.

If all candidates, that are maxterms, can be verified to be contained in ψ, the machine tests for each clause of ψ (systematically one after the other) if it is a maxterm of φ using Algorithm 2 where a pointer gives the current clause to ensure that clauses have not to be copied to be compared. If a clause is found that is not a maxterm of φ, then $(\varphi, \psi) \notin \text{Monet}_{\text{cnm}}$ with this clause as a counter-example. Otherwise, the machine can conclude $(\varphi, \psi) \in \text{Monet}_{\text{cnm}}$. □

4.1.2 The DNF contains only monomials of constant size

In this section we restrict the DNF to be a k-DNF, that is to contain only monomials of size at most k for a constant k. The corresponding restricted Monet version is $\text{Monet}_{\text{cupp}}$ (Monet with a constant upper bound for the monomial size).

$\text{Monet}_{\text{cupp}}$:	*instance:*	irredundant, monotone Boolean formulas φ in k-DNF, for a constant k, and ψ in CNF
	question:	are φ and ψ equivalent?

$\text{Monet}_{\text{cupp}}$ is known to be solvable in polynomial time $O(n^{k+1})$ [BGH98, DK88, EG95, EGM03, Elb02b, JPY88, MP97], it is even placed in RNC [BEGK00]. Note that RNC contains NL and thus L. If $k = 2$, which corresponds to the case that the hypergraph associated to the DNF in fact is a graph, there are some good old (and new) polynomial time results [Epp05, JPY88, LLK80, TIAS77]. We improve these results by showing that $\text{Monet}_{\text{cupp}}$ can be decided in logarithmic space.

Therefore, we use the following property of transversal hypergraphs originally proven in [EG95] (part of Theorem 5.2 there). Note that we only changed notation to better fit in the Monet setting (remember that $\text{Monet}_{\text{cupp}}$ is equivalent to the problem TransHyp with bounded edge-size where φ and ψ have to be transversal hypergraphs of each other).

Lemma 4.1.2 ([EG95])**.** *Let φ in k-DNF with the set of monomials M_φ and ψ in CNF with the set of clauses C_ψ be two irredundant, monotone Boolean formulas. If $k \geq 2$, then:*

$$(\varphi, \psi) \in \text{Monet}_{\text{cupp}} \iff C_\psi \subseteq \text{Tr}(M_\varphi) \wedge E_1 \wedge E_2, \tag{4.1}$$

with

$$\begin{aligned} E_1 &\equiv \neg \exists m \subseteq V, |m| \leq k : m \in \text{Tr}(C_\psi) \wedge m \notin M_\varphi, \\ E_2 &\equiv \neg \exists C'_\psi \subseteq C_\psi, |C'_\psi| = k+1 : \forall c \in C_\psi : c \not\subseteq \{x \in V : d(x, C'_\psi) > 1\}, \end{aligned}$$

where $d(x, C'_\psi)$ denotes the number of sets in C'_ψ that contain the variable x.

With Lemma 4.1.2 at hand we are now ready to tighten the complexity bound for $\text{Monet}_{\text{cupp}}$.

Theorem 4.1.3. $\text{Monet}_{\text{cupp}}$ *is decidable in logarithmic space.*

Proof. Let n be the size of the MONET$_{\text{cupp}}$ instance (φ, ψ). We assume that every monomial of φ has size at most k for a constant k. An appropriate machine is able to determine this k by counting in logarithmic space. We will show that the right hand side of (4.1) can be verified in logarithmic space. Therefore, we describe the work of an appropriate machine T. The machine uses the logspace procedures `isIn` from Lemma 2.5.7 and Algorithm 2 from Lemma 2.5.5 as subroutines. Note that procedure calls can be space-efficiently simulated by using pointers to cells on input or working tapes of T, where parameters needed for the procedure call start.

$C_\psi \subseteq \mathrm{Tr}(M_\varphi)$: T calls the maxterm test (Algorithm 2) systematically for φ and every clause in C_ψ. To know which clause is currently tested, T counts the number of tested clauses. This counter can be managed in logarithmic space in the size of C_ψ.

E_1: Every constant-sized m has to be checked. To do this, T systematically generates the candidates. To know which candidate is the actual candidate, T counts the number of already checked candidates. The number of possible candidates is bounded by $1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{k} = O(n^k)$. Hence the counter needs $k \cdot \log(n)$ bits. Because of the constant size of m, the machine T can write down the whole current candidate. For every candidate m a procedure analogous to Algorithm 2 from Lemma 2.5.5 answers the question whether m is a prime implicant of ψ. If the answer is Yes, then T calls `isIn` (Algorithm 4) to know whether m is in φ. Altogether, E_1 can be verified in logarithmic space.

E_2: Only a constant number of clauses form the current candidate set for the E_2-test. By systematically generating the candidate sets, T is able to know the monomials that form the current candidate by counting the candidates. The counter must count to $\binom{n}{k+1} = O(n^{k+1})$, hence, logarithmic space suffices. Because of the constant size of the candidates C'_ψ, the machine T can manage pointers to each clause in the current candidate set C'_ψ on some working tape. Hence, using `isIn` (Algorithm 4), T is able to check, for every variable in every clause, if the variable is contained in more than one element of C'_ψ.

Altogether, logarithmic space suffices to decide MONET$_{\text{cupp}}$. □

4.1.3 The DNF contains only very large monomials

In this section we restrict the DNF with variable set V to contain only monomials of size at least $|V| - c$ for a constant c. The corresponding restricted MONET version is MONET$_{\text{clow}}$ (MONET with a "<u>c</u>onstant" <u>low</u>er bound for the monomial size).

MONET$_{\text{clow}}$:	*instance:*	irredundant, monotone φ in DNF and ψ in CNF with variable set V, where monomials in φ have size at least $\lvert V\rvert - c$ for a constant c
	question:	are φ and ψ equivalent?

$\textsc{Monet}_{\text{clow}}$ is known to be decidable in polynomial time $O(n^{c+1})$ [BGH98, EG95]. We improve this bound to logarithmic space. First, we need some technical definitions. The complement $\overline{V'}$ of a subset V' of V and the complement $\overline{\mathcal{F}}$ of a family $\mathcal{F}$ of subsets of V are defined as $\overline{V'} = V \setminus V'$ and $\overline{\mathcal{F}} = \{\overline{F} : F \in \mathcal{F}\}$. For an irredundant, monotone DNF φ with the set M_φ of monomials we define the operator τ as $\tau(M_\varphi) = \{m \setminus \{x\} : m \in M_\varphi, x \in m\}$. The following important fact is due to Eiter and Gottlob [EG95].

Proposition 4.1.4 ([EG95]). *Let φ be an irredundant, monotone DNF with the set M_φ of monomials. Every clause of the irredundant, monotone CNF ψ equivalent to φ is contained in $\overline{\tau(M_\varphi)}$.*

With Proposition 4.1.4 at hand, we can give an algorithm deciding $\textsc{Monet}_{\text{clow}}$ in logarithmic space.

Theorem 4.1.5. $\textsc{Monet}_{\text{clow}}$ *is decidable in logarithmic space.*

Proof. Let n be the size of the $\textsc{Monet}_{\text{clow}}$-instance (φ, ψ) and M_φ be the set of monomials of φ. Whether (φ, ψ) is a $\textsc{Monet}_{\text{clow}}$-instance can be tested in logarithmic space. Counting the variables in each monomial of φ suffices and the counter clearly stays logarithmic in n. Let the lower bound for the monomial size be $|V| - c$ for constant c.

It remains to check the equivalence of φ and ψ. From Proposition 4.1.4 it follows that the only candidates for clauses of a CNF equivalent to φ are contained in $\overline{\tau(M_\varphi)}$. The machine performs a candidate generation and check procedure very analogous to the one from the proof of Theorem 4.1.1. Each candidate arises from a monomial and includes all variables that are not included in the monomial plus one variable from the monomial. Hence, the candidate size is bounded by $c + 1$. Each such candidate can be written down on an extra tape due to the constant size. For each candidate the machine works like the one from Theorem 4.1.1 and checks whether they are maxterms and contained in ψ. To know the next candidate, the machine systematically generates them and counts the number of already generated candidates. It starts by generating all candidates from the first monomial m_1 of φ. The first candidate is the set of all variables not contained in m_1 and the first variable from m_1. The second candidate is the set of all variables not contained in m_1 and the second variable from m_1, etc. After finishing the generation of all candidates from the first monomial, the machine generates all candidates from the second monomial in the same way. After that, all candidates from the third monomial, etc. Altogether, there are at most $|V| \cdot |M_\varphi|$ many candidates. Hence, the counter stays logarithmic in n. If a candidate, that is a maxterm of φ, cannot be found in ψ, the machine rejects. After finishing the candidate generation, the machine has to test for all clauses of ψ whether they are maxterms of φ like the machine in the proof of Theorem 4.1.1 does. $\square$

4.1.4 Polynomial time size restrictions

Besides the logspace solvable classes from the previous sections, there are also size restrictions for which the best known algorithms are polynomial and thus better than for arbitrary MONET instances.

Logarithmic number of variables Suppose that a MONET instance (φ, ψ) of size n only contains $O(\log n)$ variables. Note that then the number of assignments for (φ, ψ) is bounded by a polynomial in n and thus a brute-force test checking $\mathcal{A}(\varphi) = \mathcal{A}(\psi)$ for every possible assignment $\mathcal{A}$ is polynomial in the input size.

Logarithmic "lower" bound on monomial size This class is a generalization of the problem $\text{MONET}_{\text{clow}}$, where the k in the $\text{MONET}_{\text{clow}}$ definition now is allowed to be logarithmic. Using an Apriori approach (cf. Sections 5.2 or 7.4), this class (in the transversal hypergraph setting) can be solved in polynomial time [GKM+03, GKMT97]. Note that our parameterized result for the unions of hypergraph transversals (cf. Section 7.4) further generalizes this class.

Logarithmic number of monomials This class is a generalization of $\text{MONET}_{\text{cnm}}$ where we now allow not only a constant but a logarithmic number of monomials. It can be shown to be solvable in polynomial time [Dom97, Mak03].

4.2 Structural restrictions on the DNF

Having examined the size restrictions, we now address more structural restrictions. We focus on MONET-instances with a DNF φ that is regular, aligned, or 2-monotonic. For all three classes polynomial time algorithms are known [BS87, Bor94, BHIK97]. We improve the resource bounds by giving logarithmic space algorithms.

4.2.1 The DNF is regular

Definition 4.2.1 (Regular). *A formula α with the set $V = \{x_1, \dots, x_{|V|}\}$ of variables is regular, if for every pair of variable indices $i < j$ and every assignment $\mathcal{A}$ with $x_i \notin \mathcal{A}$ and $x_j \in \mathcal{A}$ it holds that $\mathcal{A}(\alpha) \leq \mathcal{A}'(\alpha)$, where $\mathcal{A}' = (\mathcal{A} \backslash \{x_j\}) \cup \{x_i\}$.*

Example 4.2.2. As an example consider the regular DNF

$$\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4).$$

In this section, we examine the following special class of MONET.

$\text{MONET}_{\text{reg}}$:	*instance:*	irredundant, monotone formulas φ in DNF and ψ in CNF with variable set V, where φ is regular
	question:	are φ and ψ equivalent?

Algorithm 5 The regularity test

Input: irredundant, monotone DNF φ
Output: Yes, if φ is regular, and No, otherwise
1: **for all** monomials m_i **do**
2:   **for all** variables $x_j \in V$ **do**
3:     **if** $x_j \notin m_i$ and $x_{j+1} \in m_i$ **then**
4:       $\mathcal{A} \leftarrow (m_i \setminus \{x_{j+1}\}) \cup \{x_j\}$
5:       **if** $\mathcal{A}(\varphi) = 0$ **then**
6:         **output** No and **stop**
7: **output Yes**

Regularity testing of φ can be managed in logarithmic space. We use the following observation due to Muroga.

Proposition 4.2.3 ([Mur71])**.** *Let $V = \{x_1, \ldots, x_{|V|}\}$ be the variable set of a monotone formula α. Then α is regular if and only if for all prime implicants m of α and all variables $x_i \notin m$ and $x_{i+1} \in m$ the assignment $(m \setminus \{x_{i+1}\}) \cup \{x_i\}$ satisfies α.*

Lemma 4.2.4. *The regularity test for an irredundant, monotone DNF φ can be implemented to run in logarithmic space.*

Proof. Note that the irredundant, monotone DNF φ already consists of all prime implicants of φ. Let $V = \{x_1, \ldots, x_{|V|}\}$ be the variable set of φ and let $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ be the set of φ's monomials (prime implicants). A regularity test for irredundant, monotone DNFs is given in Algorithm 5. The correctness of Algorithm 5 is straightforward as the algorithm implements the test of the property stated in Proposition 4.2.3. We have to analyze the space requirement.

Both for-loops can manage counters that contain the number of the currently tested monomial and the index of the current variable to know which are the current monomial and variable. Such counters stay logarithmic in the input size.

The two containedness tests of the first if in line 3 require only one additional counter to store the index $j+1$. This counter is logarithmic in the input size. The containedness tests just have to search the indices j and $j+1$ in m_i. They need no additional storage other than the two logarithmic counters of the for-loops and the logarithmic index $j+1$ to know the current monomial and the current variables. The second if in line 5 is answered by a subprocedure for evaluating a DNF under a given assignment. Therefore, the machine gives the indices i, j and $j+1$ to the DNF evaluation procedure from Lemma 2.5.6 that then knows the assignment to be tested. As the evaluation is logspace, this procedure call does not increase the resource requirements. $\square$

It is known that $\textsc{Monet}_{\text{reg}}$ is decidable in polynomial time [BS87, Cra87, HPP79, MI98, PS85, PS94], the best bound being quadratic. We show that already logarithmic space suffices.

Theorem 4.2.5. $\textsc{Monet}_{\mathrm{reg}}$ *is decidable in logarithmic space.*

Proof. Let n be the size of the instance (φ, ψ) and $V = \{x_1, \ldots, x_{|V|}\}$ be the set of variables.

As for the equivalence test, we slightly adapt the procedure RSC of Bertolazzi and Sassano [BS87]. RSC computes all the maxterms of a given regular, irredundant, monotone DNF φ. Bertolazzi and Sassano have found the following coherence between a regular DNF and its maxterms. For each monomial m of φ and every variable $x_j \in m$, $j > l$ for some value l, the set $F_j(m) \cup \{x_j\}$ is a maxterm of φ. Thereby, l is the smallest index of a variable contained in monomial m of φ but not in the lexicographic predecessor of m; and $F_j(m) = \{x_k \notin m : k < j\}$. Here lexicographic ordering means that the monomials are ordered lexicographically by their characteristic vectors. The characteristic vector of monomial m is $|V|$-dimensional and contains a 1 at position k iff $x_k \in m$. Monomial m_i is lexicographically larger than monomial m_j, $m_i >_{lex} m_j$, if and only if m_i's vector has a 1 in the first position, where the characteristic vector of m_i and m_j differ.

The above property for the maxterms of φ is used in lines 5 and 6 of our equivalence test given as Algorithm 6. It computes the maxterms of φ, one after the other, and checks whether they are contained in ψ. Afterwards we have to check whether each clause of the given CNF ψ really is a maxterm of φ.

RSC precomputes a lexicographic ordering of the monomials and an $|M|$-dimensional vector γ containing the smallest variable indices that distinguish lexicographically adjacent monomials. Our algorithm does not have enough space to store such precomputations. Instead, it processes the monomials in the ordering they are given and computes the index (in the given ordering) of the predecessor in a lexicographic ordering (function `pred`, cf. Algorithm 7) every time it is needed. Analogously, the smallest variable index that distinguishes the current monomial from its predecessor (function `leastDiff`, cf. Algorithm 9) is computed every time it is needed. The listings of `pred` and `leastDiff` are given as Algorithms 7 and 9.

We first show that `pred` runs in logarithmic space. The for-loop can be managed via a logarithmic counter and since p contains monomial indices, it is also logarithmic. Hence, it is obvious that `pred` works correctly and in logarithmic space if `leqLex` does. The function `leqLex` is intended to return the index of the lexicographically smaller of two monomials. We give an appropriate algorithm as Algorithm 8. The correctness of Algorithm 8 is straightforward and hence, `pred` is correct.

The for-loop in Algorithm 8 can be managed via a logarithmic counter. The "$x \in m$"-tests can also be managed using counters. Hence, `leqLex` runs in logarithmic space and so does `pred`.

We now turn to `leastDiff` given as Algorithm 9. Given two monomials, the function `leastDiff` returns the smallest index l of a variable that is contained in only one of the monomials. The correctness of Algorithm 9 is straightforward. As for the space requirement, the for-loop requires a logarithmic counter. The

Algorithm 6 The equivalence test for regular inputs

Input: $\textsc{Monet}_{\mathrm{reg}}$-instance (φ, ψ) with the set M_φ of monomials and the set C_ψ of clauses

1: **for all** $m_i \in M_\varphi$ **do**
2:     $p \leftarrow$ `pred`(m_i, M_φ)
3:     $l \leftarrow$ `leastDiff`(m_i, m_p)
4:     **for all** variables x_j **do**
5:         **if** $x_j \in m_i$ and $j > l$ **then**
6:             **if** $F_j(m_i) \cup \{x_j\} \notin C_\psi$ **then**
7:                 **reject**
8: **for all** $c_i \in C_\psi$ **do**
9:     **if** c_i is not a maxterm of φ **then**
10:         **reject**
11: **accept**

Algorithm 7 `pred`

Input: monomial m_i of an irredundant, monotone DNF φ with the set M_φ of monomials and variable set V

Output: index of the lexicographic predecessor monomial of m_i in M_φ

$p \leftarrow i$
for all $m_j \in M_\varphi$ **do**
    if `leqLex`$(m_i, m_j) = j$ **then**
        if `leqLex`$(m_j, m_p) = p$ **then**
            $p \leftarrow j$
return p

"$x \in m$"-tests can also be managed using a logarithmic counter.

Correctness of `pred` and `leastDiff` implies correctness of Algorithm 6 as RSC was proven to be correct in [BS87] and our algorithm is just a slight adaption.

We have to examine the space requirement of Algorithm 6. All three for-loops could manage counters that contain the number of the already tested monomials in the original ordering, the index of the current variable, or the number of already tested clauses to know which are the current monomial, variable or clause. Such counters stay logarithmic in n. Both, p and l, store indices that remain logarithmic in n.

The "$\notin C_\psi$"-tests in line 6 of Algorithm 6 are answered as follows. We have to avoid writing down $F_j(m_i) \cup \{x_j\}$ each time as we have to be careful on the space requirements. But note that instead the machine knows F_j implicitly, given i and j. Hence, by slightly adapting the `isIn` procedure from Lemma 2.5.7 we can just provide pointers in the form of i and j and get the correct answer by a call to this subprocedure. Each subprocedure call only needs logarithmic space.

The maxterm-test in line 9 of Algorithm 6 is implemented as a subprocedure

Algorithm 8 `leqLex`

Input: two monomials m_i, m_j on variable set $V = \{x_1, \dots, x_{|V|}\}$
Output: index of the lexicographically smaller monomial

```
for all x_k ∈ V in ascending index order do
    if x_k ∉ m_j and x_k ∈ m_i then
        return j
    else if x_k ∈ m_j and x_k ∉ m_i) then
        return i
return i
```

Algorithm 9 `leastDiff`

Input: two monomials $m_i \geq_{lex} m_j$ of an irredundant, monotone DNF φ with the set M_φ of monomials and variable set V
Output: smallest index k, such that $x_k \in m_i$ and $x_k \notin m_j$

```
if i = j then
    return 0
for all x_k ∈ V in ascending index order do
    if x_k ∉ m_j and x_k ∈ m_i then
        return k
```

call as well. Therefore, the index i is given to Algorithm 2 that then can know the DNF and the clause. □

4.2.2 The DNF is aligned

In this section we turn to a generalization of regular formulas.

Definition 4.2.6 (Aligned). *Let $V = \{x_1, \dots, x_{|V|}\}$ be the variable set of a monotone formula α. Then α is aligned, if for all prime implicants m of α and all variables $x_i \notin m$ with $i \leq max_m = \max\{j : x_j \in m\}$ the assignment $(m \setminus \{x_{max_m}\}) \cup \{x_i\}$ also satisfies α.*

Every regular formula is aligned (compare Proposition 4.2.3 and the definition of aligned). But the converse does not hold, as can be seen by the following example.

Example 4.2.7.

$$\begin{aligned}\varphi \quad = \quad & (x_1) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_3 \wedge x_4) \vee \\ & (x_3 \wedge x_5 \wedge x_6) \vee (x_4 \wedge x_5 \wedge x_6 \wedge x_7) \vee (x_5 \wedge x_6 \wedge x_7 \wedge x_8).\end{aligned}$$

The DNF φ is aligned but it is not regular, since for $\mathcal{A} = (\{x_5, x_6, x_7, x_8\} \setminus \{x_7\}) \cup \{x_4\}$ we have $\mathcal{A}(\varphi) = 0$.

Hence, aligned formulas are a generalization of regular ones. In this section, we consider the following special class of Monet.

Algorithm 10 The alignedness test

Input: irredundant, monotone DNF φ with variable set V
Output: Yes, if φ is aligned, and No, otherwise

1: **for all** $m_i \in M_\varphi$ **do**
2:     **for all** $x_j \in V$ **do**
3:         **if** $x_j \notin m_i$ **then**
4:             $\mathcal{A} \leftarrow (m_i \setminus \{x_{max_{m_i}}\}) \cup \{x_j\}$
5:             **if** $\mathcal{A}(\varphi) = 0$ **then**
6:                 **output** No and **stop**
7: **output** Yes

$\textsc{Monet}_{\mathrm{ali}}$:	*instance:*	irredundant, monotone formulas φ in DNF and ψ in CNF with variable set V, where φ is aligned
	question:	are φ and ψ equivalent?

Testing whether φ is aligned can be managed in logarithmic space.

Lemma 4.2.8. *Whether an irredundant, monotone DNF φ is aligned can be decided in logarithmic space.*

Proof. We slightly adapt the algorithm of the regularity test given in Lemma 4.2.4, since we do not have to test all what we have tested there (compare the definition of an aligned formula with Proposition 4.2.3).

Note that the irredundant, monotone DNF φ, that is input for the test, already consists of all prime implicants of φ. Let $V = \{x_1, \ldots, x_{|V|}\}$ be the variable set and $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ be the set of monomials (prime implicants) of φ. The largest variable index appearing in a monomial m is denoted by max_m. An algorithm, testing whether φ is aligned, is given as Algorithm 10. The correctness proof of Algorithm 10 is straightforward, since the algorithm just tests the property given in the definition of aligned formulas. We have to analyze the space requirement.

Both for-loops could manage counters that contain the number of the currently tested monomial and the index of the current variable to know which are the current monomial and variable. Such counters stay logarithmic in the input size. The variable index max_{m_i} can be stored using logarithmic space, too.

The containment test of the if in line 3 just has to search the index j in m_i, which can be done with logarithmic space as described in the proof of Lemma 4.2.4. Analogously to the regularity testing algorithm, the if in line 5 is answered by a subprocedure for evaluating a DNF under a given assignment. Therefore, the machine gives the indices i, j and $j+1$ to the DNF evaluation procedure from Lemma 2.5.6 that then knows the assignment to be tested. □

It is known that $\textsc{Monet}_{\mathrm{ali}}$ is decidable in quadratic time [Bor94]. We show that logarithmic space suffices.

Theorem 4.2.9. $\text{MONET}_{\text{ali}}$ *is decidable in logarithmic space.*

Proof. Let (φ, ψ) be a MONET-instance of size n and $V = \{x_1, \ldots, x_{|V|}\}$ be its variable set. As for the equivalence test, we use an approach of Boros [Bor94] but modify the algorithm to show that it works in logarithmic space.

For an assignment $\mathcal{A}$ let $max_{\mathcal{A}}$ denote the largest variable index that is included in $\mathcal{A}$. An assignment $\mathcal{A}$ satisfying a monotone formula α is called *leftmost*, if for $\mathcal{A}' = \mathcal{A} \setminus \{x_{max_{\mathcal{A}}}\}$ we have $\mathcal{A}'(\alpha) = 0$. Boros has shown that the irredundant, monotone DNF of an aligned formula α exactly comprises of all leftmost assignments of φ [Bor94]. In the proof, Boros uses a special type of binary decision tree (BDT) representation of aligned formulas, shows this representation to be polynomial size bounded, and gives a polynomial time algorithm for $\text{MONET}_{\text{ali}}$ based on the BDT. We will modify his algorithm to achieve a logarithmic space bound.

A *binary decision tree (BDT)* T is a directed binary tree. The nodes of the tree have either two or no outgoing edges. The nodes reachable from node v are the *successors* of v and together with v they form the subtree $T(v)$. The nodes $w \neq v$ for which $v \in T(w)$ are the *predecessors* of v. There is only one node without predecessors, the *root* r. The nodes with two outgoing edges are the *inner nodes* of T. The nodes with no outgoing edges are the *leaves* of T. The leaves of T have labels 0 (*false leaves*) or 1 (*true leaves*) such that there is no inner node v for which $T(v)$ only contains leaves with the same label. Let L_0 (L_1) be the set of all false (true) leaves of T. The set of predecessors of node v forms a directed path $D(v) = \{v_1 = r, v_2, \ldots, v_{d(v)} = v\}$, where $d(v)$ denotes the depth of v (the distance from the root). Each inner node gets a variable as label. In our case, the label of node v is $x_{d(v)}$. Let u be an inner node with the outgoing edges (u, v) and (u, w). We say that v and w are the *sons* of u and u is their *father*. One son is the *true son* $\text{ts}(u)$ and the other the *false son* $\text{fs}(u)$. For the path $D(v)$ we define the sets $\mathit{true}(v) = \{x_{d(v_k)} : v_{k+1} = \text{ts}(v_k), k = 1, \ldots, d(v)\}$ and $\mathit{false}(v) = \{x_{d(v_k)} : v_{k+1} = \text{fs}(v_k), k = 1, \ldots, d(v)\}$ of nodes appearing as true sons respectively false sons. Each such BDT T represents a DNF of a Boolean formula α and its dual in the following way,

$$\alpha = \bigvee_{v \in L_1} \bigwedge_{x_i \in \mathit{true}(v)} x_i \bigwedge_{x_i \in \mathit{false}(v)} \neg x_i \qquad \text{and} \qquad \alpha^d = \bigvee_{v \in L_0} \bigwedge_{x_i \in \mathit{false}(v)} x_i \bigwedge_{x_i \in \mathit{true}(v)} \neg x_i.$$

For the dual α^d of a formula α it holds that $\mathcal{A}(\alpha) = \neg\mathcal{A}(\neg\alpha^d)$. Note that the irredundant CNF of an irredundant, monotone DNF φ can be produced by switching the roles of $\wedge$ and $\vee$ in the irredundant DNF of φ^d. This will be the key for our algorithm. Namely, Boros has proven that for each monotone formula α there exists an unique BDT T_α whose true leaves (false leaves) correspond one-to-one to the leftmost assignments of α (α^d) [Bor94]. We have

$$\alpha = \bigvee_{v \in L_1} \bigwedge_{x_i \in \mathit{true}(v)} x_i \qquad \text{and} \qquad \alpha^d = \bigvee_{v \in L_0} \bigwedge_{x_i \in \mathit{false}(v)} x_i.$$

Algorithm 11 The equivalence test for aligned inputs

```
Input: MONET_ali-instance (φ, ψ) with the set M_φ of monomials and the set
       C_ψ of clauses
for all m_i ∈ M_φ do
    for all x_j ∈ m_i do
        if m_i^{j-1} ∪ {x_{j+1}} is not a subimplicant of φ then
            if overline(m_i^{j-1}) ∩ {x_1, ..., x_j} is not a superclause of ψ then
                reject
for all c_i ∈ C_ψ do
    if c_i is not a maxterm of φ then
        reject
accept
```

Boros has shown that for aligned formulas the BDT is only polynomial in size and constructs an algorithm that computes the BDT and from it the CNF of φ [Bor94].

Our algorithm cannot compute the whole BDT since it would require polynomial space to store it. But remember that our input is the DNF φ and it is aligned. Hence, from the results of Boros it follows that the monomials of φ are in a one-to-one relation with the true leaves of the BDT for φ. We only have to search all false leaves v and check whether $\mathit{false}(v)$ is contained in ψ since no other maxterms exist. But how do we go through all false leaves? It can be easily proven that they are sons of nodes lying on the path to a true leave.

Claim 4.2.10. There is no false leave in a BDT T that is not the false son of a father contained in $D(v)$ for a true leave v.

Proof. Assume that we could find a false leave u that is the true son of a node w. Then $\mathit{true}(u)$ is a leftmost assignment of a formula represented by T. But then u cannot be a false leave. A contradiction. Hence, false leaves are false sons of their fathers.

Assume now that the father w of the false leave u is not contained in any $D(v)$ for a true leave v. Hence, the leaves in $T(w)$ all are false leaves. Again, a contradiction. □

We will test each node in the BDT described by the monomials of φ as a potential father of a false leave. Therefore, we check for each branch on the path described by a monomial whether the false son is a false leave and if so whether the corresponding maxterm is contained in ψ. In a second step we check whether all clauses of ψ are maxterms of φ.

Let $m^j = m \cap \{x_1, \dots, x_j\}$ for a monomial m and $\overline{s} = V \setminus s$ for a subset s of V. Using this notation, an appropriate algorithm deciding $\textsc{Monet}_{\mathrm{ali}}$ is given as Algorithm 11. The correctness of Algorithm 11 is straightforward, since it just implements the techniques described above.

Algorithm 12 Subimplicant test

Input: irredundant, monotone DNF φ with the set $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ of monomials and a subset s of the set $V = \{x_1, \ldots, x_{|V|}\}$ of variables
Output: Yes, if s is a subimplicant of φ, and No, otherwise

```
1: for all m_i ∈ M_φ do
2:     test ← 1
3:     for all x_j ∈ s do
4:         if x_j ∉ m_i then
5:             test ← 0
6:     if test = 1 then
7:         output Yes and stop
8: output No
```

We analyze the space requirement. All three for-loops know the current monomial, variable or clause by using logarithmically space bounded counters. The if-tests in lines 3, 4 and 7 are answered by three subprocedures.

In line 3 the algorithm gives the indices i and j to the procedure given in Algorithm 12 that then implicitly knows $m_i^{j-1} \cup \{x_{j+1}\}$. A monotone monomial m is a subimplicant of monotone formula α if m is subset of a term of the irredundant, monotone DNF of α.

Claim 4.2.11. Whether a subset s of the set $V = \{x_1, \ldots, x_{|V|}\}$ of variables of an irredundant, monotone DNF φ is a subimplicant of φ can be decided in logarithmic space.

Proof. Note that the DNF φ contains all prime implicants of φ. Let the input size n be the number of variable occurrences in φ and s. An algorithm with the desired properties is given as Algorithm 12. The correctness of Algorithm 12 is straightforward. We have to analyze the space requirement.

Both for-loops can be managed using logarithmic counters to know the current monomial or variable. The *test*-variable needs constant space. □

In line 4, Algorithm 11 gives the indices i and j to the procedure given in Algorithm 13 that then implicitly knows $\overline{m_i^{j-1}} \cap \{x_1, \ldots, x_j\}$. A monotone clause c is a superclause of a monotone formula α if c contains a term of the irredundant, monotone CNF of α.

Claim 4.2.12. Whether a subset s of the set $V = \{x_1, \ldots, x_{|V|}\}$ of variables of an irredundant, monotone CNF ψ is a superclause of ψ can be decided in logarithmic space.

Proof. Note that the CNF ψ contains all maxterms of ψ. Let the input size n be the number of variable occurrences in ψ and s. An algorithm with the desired properties is given as Algorithm 13. Note that Algorithm 13 is analogous to Algorithm 12. Hence, an analogous argumentation gives the logarithmic space bound. □

Algorithm 13 Superclause test

Input: irredundant, monotone CNF ψ with the set $C_\psi = \{c_1, \dots, c_{|C_\psi|}\}$ of clauses and a subset s of the set $V = \{x_1, \dots, x_{|V|}\}$ of variables
Output: Yes, if s is a superclause of ψ, and No, otherwise

for all $c_i \in C_\psi$ **do**
    $test \leftarrow 1$
    for all $x_j \in c_i$ **do**
        if $x_j \notin s$ **then**
            $test \leftarrow 0$
    if $test = 1$ **then**
        output Yes and **stop**
output No

Finally, Algorithm 11 gives the index i to the subprocedure given as Algorithm 2 that then implicitly knows c_i and can manage the maxterm test in line 7 of Algorithm 11. All three subprocedures are logarithmically space bounded and so is Algorithm 11. □

4.2.3 The DNF is 2-monotonic

Another generalization of regular formulas are 2-monotonic formulas.

Definition 4.2.13 (2-Monotonic)**.** *A monotone formula α is 2-monotonic if there exists a permutation π of the variables such that $\pi(\alpha)$ is regular.*

In this section, we consider the following version of MONET.

MONET$_{\text{2m}}$:	*instance:*	irredundant, monotone formulas φ in DNF and ψ in CNF with variable set V, where φ is 2-monotonic
	question:	are φ and ψ equivalent?

It is known that MONET$_{\text{2m}}$ is decidable in polynomial time [BHIK97, MI97, MI98, PB88], the best bound being cubic [MI98]. We show that already logarithmic space suffices. Therefore, we use the following result about 2-monotonic formulas.

Proposition 4.2.14 ([Mak02, Win62])**.** *Let α be a 2-monotonic formula with the set $V = \{x_1, \dots, x_{|V|}\}$ of variables. For every $x_j \in V$ let position k of a $|V|$-dimensional vector $\alpha^{(j)}$ be*

$$\alpha_k^{(j)} = |\{m \text{ is a prime implicant of } \alpha : x_j \in m, |m| = k\}|.$$

Let $\alpha^{(j_1)} \geq_{lex} \alpha^{(j_2)} \geq_{lex} \cdots \geq_{lex} \alpha^{(j_{|V|})}$, where $\geq_{lex}$ denotes the lexicographic order between $|V|$-dimensional vectors, and let π be a permutation of variables such that $\pi(x_{j_i}) = x_i$ for all i. Then $\pi(\alpha)$ is regular.

Algorithm 14 Writing the w-permuted DNF on an oracle tape

1: $\pi(\varphi) \leftarrow$ "("
2: **for all** $m_i \in M_\varphi$ in ascending order **do**
3: $\quad c_1 \leftarrow 0$
4: $\quad$ **if** $i = 1$ **then**
5: $\quad\quad \pi(\varphi) \leftarrow \pi(\varphi) \circ$ "("
6: $\quad$ **else**
7: $\quad\quad \pi(\varphi) \leftarrow \pi(\varphi) \circ$ " $\vee$ ("
8: $\quad$ **for all** $x_j \in V$ **do**
9: $\quad\quad$ **if** $x_j \in m_i$ and $c_1 \neq 0$ **then**
10: $\quad\quad\quad \pi(\varphi) \leftarrow \pi(\varphi) \circ$ " $\wedge$ "
11: $\quad\quad$ **if** $x_j \in m_i$ **then**
12: $\quad\quad\quad c_1 \leftarrow c_1 + 1$
13: $\quad\quad\quad max \leftarrow$ `getMax`(φ)
14: $\quad\quad\quad c_2 \leftarrow 1$
15: $\quad\quad\quad$ **while** $max \neq i$ **do**
16: $\quad\quad\quad\quad max \leftarrow$ `getNext`(φ, max)
17: $\quad\quad\quad\quad c_2 \leftarrow c_2 + 1$
18: $\quad\quad\quad \pi(\varphi) \leftarrow \pi(\varphi) \circ$ "x_{c_2}"
19: $\quad \pi(\varphi) \leftarrow \pi(\varphi) \circ$ ")"
20: $\pi(\varphi) \leftarrow \pi(\varphi) \circ$ ")"

We refer to the permutation π from Proposition 4.2.14 as the w-permutation and show that it can be computed using logarithmic space.

Lemma 4.2.15. *Let φ be an irredundant, monotone DNF with the variable set $V = \{x_1, \ldots, x_{|V|}\}$ and the set $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ of monomials. The w-permuted $\pi(\varphi)$ can be written on an oracle tape using logarithmic space only.*

Proof. Note that we view an oracle tape to be write-only and hence we do not count space used on it as a machine is not able to reuse the information. By $s_1 \circ s_2$ we denote the operation of adding string s_2 on the oracle tape at the end of string s_1.

We give an algorithm with the desired properties as Algorithm 14. It writes the string $\pi(\varphi)$ on an oracle tape, recomputing new variable indices each time they are needed. The algorithm does not store already computed indices, since that would need more than logarithmic space. For each variable occurrence x_i in φ the algorithm counts where in the lexicographic ordering of the α-vectors the vector $\alpha^{(i)}$ appears. A variable with the corresponding index is written on the oracle tape instead of x_i. To derive the new index of x_i the algorithm computes the lexicographically last α-vector (`getMax` in line 13). As long as $\alpha^{(i)}$ is not found, the algorithm computes the next element in the ordering of the α-vectors (`getNext` in line 16) and adds one to the counter c_2 that should contain the number of $\alpha^{(i)}$ in

Algorithm 15 `getMax`

Input: irredundant, monotone DNF φ with the set $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ of monomials and the set $V = \{x_1, \ldots, x_{|V|}\}$ of variables

Output: index *max* of the variable with the lexicographically largest α-vector

1: $max \leftarrow 1$
2: **for all** $x_i \in V, i \neq 1$ **do**
3:     $k \leftarrow 0$
4:     **while** $k \leq |V|$ **do**
5:         $k \leftarrow k + 1$
6:         $c_3 \leftarrow |\{m \in M_\varphi : x_{max} \in m, |m| = k\}|$
7:         $c_4 \leftarrow |\{m \in M_\varphi : x_i \in m, |m| = k\}|$
8:         **if** $c_4 > c_3$ **then**
9:             $max \leftarrow i$
10:             $k \leftarrow |V| + 1$
11:         **else if** $c_4 < c_3$ **then**
12:             $k \leftarrow |V| + 1$
13: **return** *max*

the lexicographic ordering of Proposition 4.2.14. When $\alpha^{(i)}$ is found, the counter c_2 contains the number of $\alpha^{(i)}$ in the ordering of Proposition 4.2.14. The formula $\pi(\varphi)$ is composed as a string on the oracle tape (lines 1, 5, 7, 10, 18, 19, and 20).

The counters c_1 (largest value is the size of a largest monomial) and c_2 (largest value is $|V|$) stay logarithmic in n. Both for-loops can also be managed via logarithmically space bounded counters that contain the number of the current monomial or the index of the current variable. And last but not least, the variable *max* is logarithmically space bounded, since it only contains variable indices.

We have to analyze the functions `getMax` and `getNext` to fully describe the algorithm computing $\pi(\varphi)$. The function `getMax` should return the index i of the lexicographically largest of the $\alpha^{(i)}$. An appropriate algorithm is given as Algorithm 15. Each variable is a candidate for having the lexicographically largest α-vector. Hence, all variables are tested systematically by Algorithm 15. In the while-loop (line 4), the vector $\alpha^{(max)}$ which is so far the lexicographically largest vector and the vector $\alpha^{(i)}$ of the current variable are tested componentwise to decide which one is lexicographically larger. If it is $\alpha^{(i)}$, then i is the new maximum so far (line 9). The correctness is straightforward.

As for the space requirement, both counters c_3 and c_4 remain logarithmic in n, since their largest value is $|M_\varphi|$. They can be computed by checking the monomials systematically whether they contain the tested variable. If so, another counter is used to get the size of the current monomial. This counter is compared to k. The largest value stored in variable k is $|V| + 1$ which is logarithmic in n. Another logarithmically space bounded counter is used for the for-loop. The variable *max* contains variable indices. Hence, it is logarithmically space bounded. Altogether, the function `getMax` can be computed using logarithmic space.

The function `getNext` should return the index of the variable whose α-vector is the successor, in the lexicographic ordering of Proposition 4.2.14, of the current $\alpha^{(max)}$. An appropriate algorithm is given as Algorithm 16. In lines 1

Algorithm 16 `getNext`

Input: irredundant, monotone DNF φ with the set $M_\varphi = \{m_1, \ldots, m_{|M_\varphi|}\}$ of monomials and the set $V = \{x_1, \ldots, x_{|V|}\}$ of variables, and a variable index max

Output: index of the variable whose α-vector is the successor of $\alpha^{(max)}$ in the lexicographic ordering of Proposition 4.2.14

1: $next \leftarrow 0$
2: **for all** $x_i \in V$ **do**
3:     $k \leftarrow 0$
4:     **while** $k \leq |V|$ **do**
5:         $k \leftarrow k + 1$
6:         $c_5 \leftarrow |\{m \in M_\varphi : x_{max} \in m, |m| = k\}|$
7:         $c_6 \leftarrow |\{m \in M_\varphi : x_i \in m, |m| = k\}|$
8:         **if** $c_6 < c_5$ **then**
9:             $next \leftarrow i$
10:             $k \leftarrow |V| + 1$
11:         **else if** $c_6 > c_5$ **then**
12:             $k \leftarrow |V| + 1$
13: **for all** $x_i \in V$ **do**
14:     $k \leftarrow 0$
15:     **while** $k \leq |V|$ **do**
16:         $k \leftarrow k + 1$
17:         $c_5 \leftarrow |\{m \in M_\varphi : x_{max} \in m, |m| = k\}|$
18:         $c_6 \leftarrow |\{m \in M_\varphi : x_{next} \in m, |m| = k\}|$
19:         $c_7 \leftarrow |\{m \in M_\varphi : x_i \in m, |m| = k\}|$
20:         **if** $c_6 < c_7$ and $c_7 < c_5$ **then**
21:             $next \leftarrow i$
22:             $k \leftarrow |V| + 1$
23:         **else if** $c_6 > c_7$ or $c_7 > c_5$ **then**
24:             $k \leftarrow |V| + 1$
25: **return** $next$

to 12, `getNext` searches a variable whose α-vector is lexicographically smaller than $\alpha^{(max)}$. All variables whose α-vector is lexicographically smaller than $\alpha^{(max)}$ are candidates for the variable having the lexicographically next largest vector. Our algorithm simply checks all candidates. In the while-loop of line 15 the algorithm tries to find a variable whose α-vector is smaller than $\alpha^{(max)}$ but larger than $\alpha^{(next)}$, the successor so far of $\alpha^{(max)}$. If $\alpha^{(i)}$ lies lexicographically in between $\alpha^{(max)}$ and $\alpha^{(next)}$, then i is a new candidate for the successor (lines 20 to 22). The correctness of `getNext` is straightforward.

The space needed for the three counters c_5, c_6, and c_7 is logarithmically bounded in n, since their largest value is $|M_\varphi|$. The values of these counters are derived analogously to the counters c_3 and c_4 in Algorithm 15. The for-loops in lines 2 and 13 manage two other logarithmic counters to know the current variable. The variable k is also logarithmically space bounded, since the largest value to store is $|V| + 1$. The variables *next* and *max* contain variable indices which are logarithmically space bounded in n. Hence, the function `getNext` can be computed using logarithmic space.

Altogether, we can conclude that the w-permutation $\pi(\varphi)$ of φ can be written on an oracle tape using logarithmic space. $\square$

With the w-permutation at hand we can show the following.

Theorem 4.2.16. $\textsc{Monet}_{2m}$ *is decidable in logarithmic space.*

Proof. Let (φ, ψ) be a Monet-instance of size n and $V = \{x_1, \ldots, x_{|V|}\}$ be its variable set.

Note that in order to test whether the DNF φ of a Monet-instance is 2-monotonic, we can test whether $\pi(\varphi)$ is regular, where π is the w-permutation from Proposition 4.2.14. Since $\pi(\varphi)$ can be written on an oracle tape using logarithmic space only (Lemma 4.2.15), we can test 2-monotonicity in logarithmic space using the logarithmic space regularity test from Lemma 4.2.4 as an oracle.

As for the equivalence test, we again use the algorithm writing $\pi(\varphi)$ on the oracle tape and it is obvious that a slight adaption could afterwards also write $\pi(\psi)$ on the oracle tape. Then the logarithmic space algorithm for $\textsc{Monet}_{reg}$ is invoked as an oracle. A conjunction of the answers of both oracle calls—the regularity test and the equivalence check—yields the result. Since $\mathsf{L}^{\mathsf{L}} = \mathsf{L}$ for conjunctive usage of two oracles, the oracles do not increase the resource requirements. $\square$

4.2.4 Polynomial time structural restrictions

Besides the logspace solvable structural restrictions of Monet mentioned in the last sections, there are many more structural restrictions that are known to be polynomial time solvable and thus “easier” than arbitrary Monet instances. In the following we give a brief survey.

α-acyclic Associate with a hypergraph $\mathcal{H}$ the graph $G_{\mathcal{H}}$ whose vertex set is the vertex set of $\mathcal{H}$ and in which vertices are adjacent if they appear together in some edge of $\mathcal{H}$. A hypergraph $\mathcal{H}$ is *α-acyclic* iff $G_{\mathcal{H}}$ is chordal (every cycle of length at least 4 has a chord) and each clique of $G_{\mathcal{H}}$ is contained in some edge of $\mathcal{H}$. This definition can be shown to be equivalent to the case that the repetition of the following two rules (known as GYO-reduction) yields the empty hypergraph [BFMY83, GS83]:

1. if vertex v appears in only one edge, remove v from that edge,

2. if $e \subseteq e'$ for two distinct edges e, e', remove e.

Whether a hypergraph is α-acyclic can be tested in polynomial [Fag83] and even in linear [TY84] time. MONET restricted to instances where the DNF has an associated α-acyclic hypergraph is polynomial time solvable [EGM03]. Note however, that α-acyclicity is not immune against edge deletions as an α-acyclic hypergraph may have subhypergraphs that are α-cyclic.

β-acyclic The non-immunity against edge deletions of α-acyclicity leads to the notion of β-acyclicity. A hypergraph is *β-acyclic* iff each of its subhypergraphs is α-acyclic. This property can be tested in polynomial time [Fag83]. MONET restricted to instances where the DNF has an associated β-acyclic hypergraph can be shown to be polynomial time solvable [EG95].

Bounded clause-monomial intersections A hypergraph $\mathcal{H}$ is *r-exact* iff any minimal transversal of $\mathcal{H}$ intersects any edge in at most r vertices. Recently, it was shown that MONET restricted to instances where the DNF has an associated r-exact hypergraph—and thus the DNF's monomials have bounded intersections with clauses of the equivalent CNF—is polynomial time solvable [ER08].

Note that this class also includes μ-equivalent DNFs (see the corresponding paragraph below) and DNFs whose associated binary matrix (each row is the characteristic vector of a monomial) has the circular ones property [HM03]. The μ-equivalent DNFs are clearly 1-exact and the DNFs having a circular ones matrix are 2-exact.

Bounded conformality A hypergraph $\mathcal{H}$ is *bounded conformal* iff for every subset X of $\mathcal{H}$'s vertices, X is contained in an edge of $\mathcal{H}$ whenever all of X's subsets of size at most δ, for a constant δ, are contained in an edge. It is *bounded dual conformal* if its transversal hypergraph is bounded conformal for some constant. MONET restricted to instances where the DNF has an associated bounded conformal or bounded dual conformal hypergraph can be shown to be polynomial time solvable [KBEG07a, KBEG07b].

Bounded degree (read-k) A DNF has *bounded degree* (is *read-k*) iff every variable appears in at most k monomials. MONET restricted to instances where the DNF is read-k can be shown to be solvable in polynomial time $O(n^{k+3})$ for constant k and we even have polynomial time for logarithmic k [DMP99, Dom97, EGM03, KBEG07a, KBE+05, MP97]. Note that in Chapter 7 we show MONET to be fixed-parameter tractable with respect to the degree as a parameter and that this result also implies polynomial time solvability for bounded degree.

Bounded monomial intersections MONET restricted to instances where the DNF satisfies the property that any k monomials intersect in at most r variables, $k + r \leq c$, for some constant c, can be shown to be polynomial time solvable [KBEG07b].

Bounded treewidth A *tree decomposition* (of type 1) of an irredundant, monotone DNF φ with variable set V is a tree $T = (W, E)$, where each vertex $w \in W$ is labeled with a subset $S(w) \subseteq V$. The following properties have to hold:

- $\bigcup_{w \in W} S(w) = V$,
- for every monomial $m \in \varphi$ there is a $w \in W$ with $m \subseteq S(w)$,
- for each variable $x_i \in V$ the induced set $\{w \in W : x_i \in S(w)\}$ is a connected subtree of T.

The *width* of T is $\max_{w \in W} |S(w)| - 1$ and the type 1 *treewidth* of φ is the minimal width of all tree decompositions of φ.

There is also a type 2 treewidth, which is the usual graph treewidth of the incidence graph of φ. Thereby, the incidence graph $G(\varphi)$ of φ is defined as follows. It has a vertex for each monomial and each variable of φ and edges between monomial and variable vertices if the variable appears in that monomial. Monet restricted to instances where the DNF has constantly bounded treewidth of either type 1 or 2 can be shown to be polynomial time solvable [EGM03]. The reason is that for type 1 a bounded treewidth of k implies that φ is a k-DNF and, thus, has bounded degree (see above), and for type 2 it can be shown that φ then is 2^k-degenerated (see below) [EGM03].

Degenerated A monotone DNF is *k-degenerated* if there exists a variable ordering $x_1, \ldots, x_n$ such that, for $i = 1, 2, \ldots, n$, the number of monomials which contain x_i and, apart from it, only variables from $x_1, \ldots, x_{i-1}$ is at most k. Monet restricted to instances where the DNF on variable set V is k-degenerated for some constant k can be shown to be solvable in polynomial time $O(n^{k+3})$; even if the DNF is $\log |V|$-degenerated a polynomial time algorithm can be given [EGM03].

Note that this class also includes DNFs whose associated binary matrix (each row is the characteristic vector of a monomial) has the consecutive ones property [McC04]. Such DNFs are 1-degenerated.

Δ-partial threshold A monotone function f is *Δ-partial threshold* iff it can be represented as:

$$f(x) = \begin{cases} 1, & \text{if } \sum(w_i \cdot x_i) \geq t + \alpha, \\ 0, & \text{if } \sum(w_i \cdot x_i) < t - \alpha, \\ 0 \text{ or } 1, & \text{otherwise,} \end{cases}$$

for $\alpha = \Delta \cdot \min\{w_i\}$ and nonnegative real numbers w_i, t, and Δ. Monet restricted to instances where the DNF is Δ-partial threshold for some constant Δ can be shown to be solvable in polynomial time $O(n^{\Delta+4})$ [MI97].

Ideal Given a simple hypergraph $\mathcal{H}$, a vertex v_i is said to be *last* iff for each edge e containing v_i and each edge f not containing v_i we have

$$\{v_k \notin e : \exists e' \in \mathcal{H} : e' \subseteq (e \setminus \{v_i\}) \cup \{v_k\}\} \cap f \neq \emptyset.$$

Given a simple hypergraph $\mathcal{H}$ and a vertex ordering $\sigma = (v_1, v_2, \ldots, v_n)$ we say that σ is *ideal* iff v_i is last in $\mathcal{H}_i = \mathcal{H}/\{v_{i+1}, \ldots, v_n\}$ where $\mathcal{H}/\{v_{i+1}, \ldots, v_n\}$ is obtained from $\mathcal{H}$ by deleting the vertices $v_{i+1}, \ldots, v_n$ and simplifying the result. A hypergraph is ideal iff there is an ideal ordering for its vertices. Ideal hypergraphs have the property that the number of their minimal transversals is bounded polynomially in the number of their edges and vertices. Monet restricted to instances where the DNF has an associated ideal hypergraph can be shown to be polynomial time solvable [BS88].

***k*-degree threshold** A monotone function f is *k-degree threshold* iff it can be represented as:

$$f(x) = \begin{cases} 1, & \text{if } \sum(w_i \cdot x_i) + \cdots + \sum_{i_1 < \cdots < i_k}(w_{i_1 \ldots i_k} \cdot x_{i_1} \cdot \ldots \cdot x_{i_k}) \geq t, \\ 0, & \text{otherwise,} \end{cases}$$

for nonnegative weights $w_{i_1 \ldots i_{k'}}$ $(1 \leq k' \leq k)$ and threshold t. Monet restricted to instances where the DNF is k-degree threshold for some constant k and that furthermore satisfy

$$2 \cdot \Delta \cdot w_{\min} \geq \sum_{l=2}^{k} \binom{n}{l} \max_{i_1 < i_2 < \cdots < i_l} w_{i_1, i_2, \ldots, i_l},$$

where $w_{\min} = \min_i w_i$, can be shown to be Δ-partial threshold (see above) [MI97]. Hence, for constant Δ, such instances are polynomial time solvable [MI97].

***k*-tight** An irredundant, monotone DNF φ is *k-tight* if for a positive integer k we have:

$$\max\{|m \setminus c| : m \in \varphi,\ c \text{ is maxterm of } \varphi \text{ with } m \setminus \{x\} \subseteq c \text{ for some } x \in m\} \leq k.$$

Monet restricted to instances where the DNF is k-tight for some constant k can be shown to be solvable in polynomial time $O(n^{k+3})$ [MI97].

Note that almost all monotone functions can be shown to be 4-tight [SKA01].

Matroid An irredundant, monotone DNF φ is *matroid* iff for any two monomials m, m' of φ and any $x \in m \setminus m'$ there is a $x' \in m' \setminus m$ such that $m \setminus \{x\} \cup \{x'\}$ is in φ. Monet restricted to instances where the DNF is matroid can be shown to be solvable in polynomial time $O(n^5)$ [MI97, PB88]

μ-equivalent A DNF is μ-equivalent iff it has an equivalent read-1 formula (in which each variable occurs only once). This property can be tested in polynomial time [Mun89]. MONET restricted to instances where the DNF is μ-equivalent can be shown to be polynomial time solvable [Eit94].

Shellable Let $\mathcal{F}$ be a family of subsets of $V = \{x_1, \ldots, x_n\}$, π be a permutation of the elements of $\mathcal{F}$ und $f_i \in \mathcal{F}$. Then the *shadow* of f_i in $\mathcal{F}$ with respect to π is $\mathrm{Sha}_{\mathcal{F},\pi}(f_i) = \{x_j \in V : \exists f_k \in \mathcal{F} : \pi(f_k) < \pi(f_i) \wedge f_k \setminus f_i = \{x_j\}\}$. We say that a monotone DNF φ with monomial set M_φ and variable set $V = \{x_1, \ldots, x_n\}$ is *shellable* iff there is a permutation of the monomials such that for all $m_i, m_j \in M_\varphi$ with $\pi(m_i) < \pi(m_j)$ we can find a $x_k \in m_i \cap \mathrm{Sha}_{M_\varphi,\pi}(m_j)$. An equivalent formulation is that there has to be a monomial $m_l \in M_\varphi$, such that $x_k \in m_i$ and $\pi(m_l) < \pi(m_j)$ and $m_l \setminus m_j = \{x_k\}$. Note that any aligned formula also is shellable and note that the shellability property of a DNF might get lost through irredundantization. Note that there is no known polynomial algorithm for testing shellability yet. However, if the appropriate permutation is given, MONET restricted to instances where the DNF is shellable, can be shown to be polynomial time solvable [BCE+00].

Stable For a function f let $\mathrm{minT}(f)$ and $\mathrm{maxT}(f)$ denote the sets of its min- and maxterms. For subsets $min \subseteq \mathrm{minT}(f)$ and $max \subseteq \mathrm{maxT}(f)$ we can approximate f by two functions g, h with $\mathrm{minT}(g) = min$ and $\mathrm{maxT}(h) = max$. A monotone function f is *stable* iff for all pairs of functions g, h defined by subsets of f's min- and maxterms as above we have

$$\max\{|\mathrm{maxT}(g)|, |\mathrm{minT}(h)|\} \leq |\mathrm{minT}(f)| + |\mathrm{maxT}(f)|.$$

MONET restricted to instances where the DNF is stable can be shown to be polynomial time solvable [Bio98].

Threshold A monotone DNF φ with variable set V is *threshold* iff there are $|V| + 1$ real numbers $w_1, \ldots, w_{|V|}, t$, such that

$$\mathcal{A}(\varphi) = 1 \Leftrightarrow \sum_{x_i \in \mathcal{A}} w_i \cdot x_i \geq t$$

holds for every assignment $\mathcal{A}$. Note that this is a special case of 2-monotonic formulas. Using the permutation in which $i < j$ for variables x_i, x_j holds iff $w_i > w_j$, yields a regular formula. Hence, MONET restricted to instances where the DNF is threshold can be solved in polynomial time.

Uniformly δ-sparse A hypergraph $\mathcal{H}$ is *uniformly δ-sparse* iff for every nonempty subset X of $\mathcal{H}$'s vertices, the average degree of the subhypergraph of $\mathcal{H}$ induced by X (the edges that only consist of vertices from X) is at most δ. MONET restricted to instances where the DNF has an associated uniformly δ-sparse hypergraph can be shown to be polynomial time solvable [BEGK04].

4.2.5 A structural restriction that does not help

Hypertree-width 2 A restriction for Monet in its transversal hypergraph formulation is to require one of the input hypergraphs to have bounded hypertree-width, which is a generalization of treewidth. This is similar to require, in the original Monet setting, that the DNF's associated hypergraph has bounded hypertree-width. Unfortunately, it can be shown that even restricting the inputs to hypertree-width 2 does not ease the problem [EG02]—note that in this case not the test for hypertree-width 2 is difficult as it can be implemented in polynomial time [GLS02].

4.3 Concluding remarks

"Easy" classes of Monet are restrictions of the DNF that admit a polynomial time solution of the corresponding restricted version of Monet. Many such classes are known but how easy are they? We showed that many easy classes actually are solvable using logarithmic space only, improving the already known polynomial time bounds. Among our results are $\text{Monet}_{\text{cnm}}$, where the DNF is allowed to contain only a constant number of monomials; $\text{Monet}_{\text{cupp}}$, where the DNF is allowed to contain only monomials of constant size; and $\text{Monet}_{\text{clow}}$, where each monomial of the DNF is only allowed not to contain a constant number of variables. As for the more structural restrictions, we have shown that Monet with a regular, a 2-monotonic, or an aligned DNF is decidable in logarithmic space.

Nevertheless, it would be very interesting to find logarithmic space algorithms for other easy classes of Monet. We conjecture that at least instances with β-acyclic or μ-equivalent DNFs are solvable in logarithmic space. Both classes are known to be polynomial time solvable [EG95, Eit94]. Especially the class of μ-equivalent DNFs—that have an equivalent formula in which each variable appears only once—is interesting, as in all known lower bound results [Hag07a, Tak07] (cf. Section 5) for algorithms solving the general problem Monet the instances used are μ-equivalent.

Another issue is to prove lower bounds for easy classes. Such lower bounds for special classes could be useful when proving hardness of Monet. This should be addressed in future research.

Chapter 5

Algorithms

In this chapter, we examine several algorithms for Monet or its computational variant. But note that discussing decision or computation algorithms is no big difference here. In fact, finding a polynomial algorithm for Monet is equivalent to finding an output-polynomial algorithm for Monet′ [BI95a]. This can be roughly seen as follows.

A Monet′ algorithm computes the equivalent irredundant, monotone CNF of a given irredundant, monotone DNF. To serve as an algorithm for Monet it can afterwards just compare its computed CNF and the given one.

The other direction is also not too involved. Note that a Monet algorithm can be easily transformed to give a witness if it answers "Not equivalent." This witness then can be used for the computation as follows. To compute the irredundant, monotone CNF of a given irredundant, monotone DNF, we start the decision algorithm on input the given DNF and an empty CNF, ask for equivalence and probably get a witness for non-equivalence. But this witness has to contain a maxterm of the DNF that can be found straightforwardly by trying to exclude variables. We include the computed maxterm in the CNF and again start the decision algorithm on the input DNF and the so far computed CNF. If still non-equivalent we use the witness again to find a new maxterm and so on.

As the computational variant of Monet and the hypergraph transversal generation problem are equivalent, we also have algorithms in this chapter solving the transversal hypergraph formulation of Monet. We decided not to transform the hypergraph algorithms to the Monet setting as often the hypergraph notation is easier to understand.

The aim of this chapter is to give an overview of the major Monet solving techniques implemented in the different algorithms and to give lower bounds for several of them showing that they are not "fast" in the sense of (output-)polynomial time.

The chapter is organized as follows. In Section 5.1 we examine one of the earliest approaches for the computational variant of Monet in its transversal hypergraph formulation—the Berge-multiplication algorithm—and several improvements thereof—the DL-, BMR-, and KS-algorithms—showing that not one of them is output-polynomial. In Section 5.2 we discuss a levelwise approach—the HBC-algorithm—similar to the well known Apriori technique and we again show that it is not output-polynomial. Afterwards, in Section 5.3 we introduce the

Algorithm 17 Berge-multiplication

```
1: Tr(H_1) ← {{v} : v ∈ e_1}
2: for i ← 2, ..., m do
3:     Tr(H_i) ← min(Tr(H_{i-1}) ∨ {{v} : v ∈ e_i})
4: output Tr(H_m)
```

FK-algorithms to have the theoretical background for the experiments in Chapter 6. The FK-algorithms are the MONET algorithms having the currently best theoretical upper bound. Some concluding remarks follow in Section 5.4.

5.1 Berge-multiplication and its improvements

In this section we focus on a method to generate transversal hypergraphs known as Berge-multiplication. We also include the discussion of several improvements of Berge-multiplication. Namely, the DL-, the BMR-, and the KS-algorithm.

5.1.1 Berge-multiplication

The Berge-multiplication algorithm [Ber89] generates transversal hypergraphs using Proposition 2.3.11 as follows. For a hypergraph $\mathcal{H} = \{e_1, e_2, \ldots, e_m\}$ let $\mathcal{H}_i = \{e_1, e_2, \ldots, e_i\}$, $i = 1, 2, \ldots, m$. We then have

$$\mathrm{Tr}(\mathcal{H}_i) \quad = \quad \min(\mathrm{Tr}(\mathcal{H}_{i-1}) \vee \mathrm{Tr}(\{e_i\})) = \min(\mathrm{Tr}(\mathcal{H}_{i-1}) \vee \{\{v\} : v \in e_i\})$$

and $\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\mathcal{H}_m)$. This implies a straightforward iterative computation process—the Berge-multiplication algorithm. A pseudocode listing is given in Algorithm 17. Despite the simplicity of Berge-multiplication, it took a couple of years until Takata [Tak07] presented a nontrivial lower bound using the following inductively defined family of hypergraphs.

Definition 5.1.1 (Takata's Hypergraphs)**.**

$$\begin{aligned} &\mathcal{G}_0 = \{\{v_1\}\} && \textit{and} \\ &\mathcal{G}_i = (\mathcal{A} \cup \mathcal{B}) \vee (\mathcal{C} \cup \mathcal{D}), && \textit{where } \mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D} \textit{ are vertex-disjoint copies of } \mathcal{G}_{i-1}. \end{aligned}$$

Takata showed the Berge-multiplication algorithm not to be output-polynomial based on the following observations.

Lemma 5.1.2 ([Tak07])**.** *We have* $|V_{\mathcal{G}_i}| = 4^i$, $\quad |\mathcal{G}_i| = 2^{2(2^i-1)}$, $\quad |\mathrm{Tr}(\mathcal{G}_i)| = 2^{2^i-1}$. *For* $i \geq 2$ *and any* $e \in \mathcal{G}_i$, *it holds that* $|\mathrm{Tr}(\mathcal{G}_i \setminus \{e\}) \setminus \mathrm{Tr}(\mathcal{G}_i)| \geq 2^{(i-2)\cdot 2^i+2}$.

From Lemma 5.1.2 it follows that, independent of the edge ordering, the penultimate (intermediate) result computed by Berge-multiplication on input $\mathcal{G}_i$ is superpolynomial in the size of the input and output (cf. the original paper [Tak07] for more details).

Very recently, Boros et. al. [BEM08] proved a subexponential $n^{\sqrt{n}}$ upper bound on the running time of Berge-multiplication.

Algorithm 18 The DL-algorithm

$\mathrm{Tr}(\mathcal{H}_1) \leftarrow \{\{v\} : v \in e_1\}$
for $i \leftarrow 2, \dots, m$ **do**
  $Tr_{\text{guaranteed}} \leftarrow \{t \in \mathrm{Tr}(\mathcal{H}_{i-1}) : t \cap e_i \neq \emptyset\}$
  $e_i^{\text{covered}} \leftarrow \{v \in e_i : \{v\} \in Tr_{\text{guaranteed}}\}$
  $\mathrm{Tr}(\mathcal{H}_{i-1})' \leftarrow \mathrm{Tr}(\mathcal{H}_{i-1}) \setminus Tr_{\text{guaranteed}}$
  $e_i' \leftarrow e_i \setminus e_i^{\text{covered}}$
  for all $t' \in \mathrm{Tr}(\mathcal{H}_{i-1})'$ in increasing cardinality order **do**
    for all $v \in e_i'$ **do**
      if $t' \cup \{v\}$ is not superset of any $t \in Tr_{\text{guaranteed}}$ **then**
        $Tr_{\text{guaranteed}} \leftarrow Tr_{\text{guaranteed}} \cup \{t' \cup \{v\}\}$
  $\mathrm{Tr}(\mathcal{H}_i) \leftarrow Tr_{\text{guaranteed}}$
output $\mathrm{Tr}(\mathcal{H}_m)$

5.1.2 The algorithm of Dong and Li

The border-differential algorithm of Dong and Li [DL05] comes from the data mining field and is intended for mining emerging patterns (cf. Section 3.8). The analogy to the generation of hypergraph transversals was already pointed out by Bailey, Manoukian, and Ramamohanarao [BMR03]. A pseudocode listing of the DL-algorithm is given in Algorithm 18.

The algorithm was experimentally evaluated on many instances from data mining settings [DL05] whereas a theoretical analysis of the running time was left open. For this purpose the conversion of the algorithm to the hypergraph setting is very fruitful. The only observable difference between Berge-multiplication and the DL-algorithm is that the DL-algorithm takes special care on how to perform the minimization of $\mathrm{Tr}(\mathcal{H}_{i-1}) \vee \{\{v\} : v \in e_i\}$. But as Takata's analysis showed, the minimization is not the bottleneck of Berge-multiplication. Thus, we can extend Takata's analysis of Berge-multiplication in a straightforward way to the DL-algorithm and get the same lower bound.

Theorem 5.1.3. *The DL-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log \log n)}$, where n denotes the size of the input and output.*

5.1.3 The algorithm of Bailey, Manoukian, and Ramamohanarao

As we have seen in Theorem 5.1.3, the DL-algorithm is not output-polynomial. Nevertheless, for hypergraphs with only a few edges of small size the DL-algorithm has been shown experimentally to perform well [DL05]. This property is exploited by the BMR-algorithm [BMR03] (cf. Algorithm 19 for the listing) as it uses the DL-algorithm as a subroutine that computes all minimal transversals for small hypergraphs (line 14 of the listing). The BMR-algorithm on input $\mathcal{H}$ is invoked

Algorithm 19 The BMR-algorithm

Input: a simple hypergraph, given by the set E of its hyperedges, and a set $V_{\text{partition}}$ of partitioning vertices

1: $V \leftarrow$ set of all vertices in E
2: order vertices by increasing number of occurrences in $E \Rightarrow [v_1, \ldots, v_k]$
3: **for** $i \leftarrow 1, \ldots, k$ **do**
4:     $E_{\text{partition}} \leftarrow \emptyset$
5:     $V \leftarrow V \setminus \{v_i\}$
6:     **for all** $e \in E$ **do**
7:         **if** $v_i \notin e$ **then**
8:             $E_{\text{partition}} \leftarrow \min(E_{\text{partition}} \cup \{e \setminus V\})$
9:     $V_{\text{partition}} \leftarrow V_{\text{partition}} \cup \{v_i\}$
10:     $a \leftarrow$ average edge cardinality of $E_{\text{partition}}$ multiplied by $|E_{\text{partition}}|$
11:     **if** $|E_{\text{partition}}| \geq 2$ and $a \geq 50$ **then**
12:         recursively call the BMR-algorithm on input $E_{\text{partition}}, V_{\text{partition}}$
13:     **else**
14:         compute $\mathrm{Tr}(E_{\text{partition}})$ via the DL-algorithm
15:         $Tr' \leftarrow \mathrm{Tr}(E_{\text{partition}}) \vee \{V_{\text{partition}}\}$
16:         $Tr \leftarrow \min(Tr \cup Tr')$
17:     $V_{\text{partition}} \leftarrow V_{\text{partition}} \setminus \{v_i\}$
18: **return** Tr

by the top-level call with the set E of edges of $\mathcal{H}$ and an empty set $V_{\text{partition}}$. The global variable Tr is initially empty.

Before calling the DL-algorithm, the BMR-algorithm ensures that the hypergraph has only few edges of small size. If this is not yet the case, the BMR-algorithm reduces the number of edges and their size by recursively deriving smaller hypergraphs from $\mathcal{H}$ (line 12). This is achieved by partitioning the edge set and masking out vertices that are more frequent than the actual partitioning vertex v_i (lines 5 to 8). If the hypergraph is small, the DL-algorithm computes all minimal transversals (line 14). These transversals are expanded by the current partitioning vertices $V_{\text{partition}}$ (line 15) since the result is a transversal of $\mathcal{H}$. The global variable Tr contains all the minimal transversals of the hypergraph $\mathcal{H}$ when the algorithm stops.

A bottleneck for the running time of the BMR-algorithm is that possibly many of the recursively computed transversals—the set Tr' in the listing—actually are not minimal for the input hypergraph $\mathcal{H}$. We concentrate on this issue and construct a family $\mathcal{G}'_i$ of hypergraphs for which the BMR-algorithm computes too many such non-minimal transversals to run in output-polynomial time. Let $\mathcal{G}'(i) = \{e_i, f_i\}$, where $e_i = \{v_{i^2-i+1}, \ldots, v_{i^2}\}$ and $f_i = \{v_{i^2+1}, \ldots, v_{i^2+i}\}$. We inductively define

$$\begin{aligned} \mathcal{G}'_1 &= \{\{v_1\}, \{v_2\}\}, \text{ and} \\ \mathcal{G}'_i &= (\mathcal{G}'_{i-1} \cup \{\{w_i\}\}) \vee \mathcal{G}'(i), \text{ for } i \geq 2. \end{aligned}$$

Note that $\mathcal{G}'_{i-1}$, $\{\{w_i\}\}$, and $\mathcal{G}'(i)$ are pairwise vertex-disjoint simple hypergraphs for $i \geq 2$. To calculate the size of $\mathcal{G}'_i$ and of $\mathrm{Tr}(\mathcal{G}'_i)$ we have to solve the recurrences $|\mathcal{G}'_i| = 2 \cdot |\mathcal{G}'_{i-1}| + 2$ and $|\mathrm{Tr}(\mathcal{G}'_i)| = |\mathrm{Tr}(\mathcal{G}'_{i-1})| + i^2$. With the initial conditions $|\mathcal{G}'_1| = 2$ and $|\mathrm{Tr}(\mathcal{G}'_1)| = 1$ we obtain

$$|\mathcal{G}'_i| = 2^{i+1} - 2 \quad \text{and} \quad |\mathrm{Tr}(\mathcal{G}'_i)| \; = \; \frac{2i^3 + 3i^2 + i}{6}$$

by iteration. For the number $|V_{\mathcal{G}'_i}|$ of vertices of $\mathcal{G}'_i$ we have $|V_{\mathcal{G}'_i}| = i^2 + 2i - 1$. As we will especially need $\mathcal{G}'_3$ later on, we explicitly give $\mathcal{G}'_2$ and $\mathcal{G}'_3$.

$$\begin{aligned}
\mathcal{G}'_2 \;=\; & \{\{v_1, v_3, v_4\}, \{v_2, v_3, v_4\}, \{w_2, v_3, v_4\},\\
& \{v_1, v_5, v_6\}, \{v_2, v_5, v_6\}, \{w_2, v_5, v_6\}\},\\
\mathcal{G}'_3 \;=\; & \{\{v_1, v_3, v_4, v_7, v_8, v_9\}, \{v_2, v_3, v_4, v_7, v_8, v_9\}, \{w_2, v_3, v_4, v_7, v_8, v_9\},\\
& \{v_1, v_5, v_6, v_7, v_8, v_9\}, \{v_2, v_5, v_6, v_7, v_8, v_9\}, \{w_2, v_5, v_6, v_7, v_8, v_9\},\\
& \{w_3, v_7, v_8, v_9\},\\
& \{v_1, v_3, v_4, v_{10}, v_{11}, v_{12}\}, \{v_2, v_3, v_4, v_{10}, v_{11}, v_{12}\},\\
& \{w_2, v_3, v_4, v_{10}, v_{11}, v_{12}\}, \{v_1, v_5, v_6, v_{10}, v_{11}, v_{12}\},\\
& \{v_2, v_5, v_6, v_{10}, v_{11}, v_{12}\}, \{w_2, v_5, v_6, v_{10}, v_{11}, v_{12}\},\\
& \{w_3, v_{10}, v_{11}, v_{12}\}\}.
\end{aligned}$$

The BMR-algorithm iteratively partitions the input hypergraph to obtain smaller hypergraphs where the transversal generation is feasible. The partitioning depends on the vertex frequencies. Hence, we first have to analyze the frequencies of the vertices in $\mathcal{G}'_i$.

Lemma 5.1.4. *For $i \geq 2$ let $\#_v(i,j)$ and $\#_w(i,j)$ respectively denote the number of occurrences of vertices v_j and w_j in $\mathcal{G}'_i$. Then*

$$\begin{array}{ll}
\#_w(i,j) = 0, & \text{for } j > i,\\
\#_w(i,j) > \#_w(i,j+1), & \text{for } 2 \leq j < i,\\
\#_w(i,2) = \#_v(i,1) = \#_v(i,2), & \\
\#_v(i,j) = 0, & \text{for } j > i^2 + i,\\
\#_v(i,j) = \#_v(i,k), & \text{for } l^2 - l + 1 \leq j \leq k \leq l^2 + l, \quad 1 \leq l \leq i,\\
\#_v(i,j) < \#_v(i,k), & \text{for } 1 \leq j < l^2 - l + 1 \leq k \leq l^2 + l, \quad 2 \leq l \leq i.
\end{array}$$

Proof. • We have the obvious equations

$$\begin{aligned}
\#_w(i,j) &= 0, && \text{for } j > i, \text{ and} && (5.1)\\
\#_v(i,j) &= 0, && \text{for } j > i^2 + i, && (5.2)
\end{aligned}$$

as neither w_j, for $j > i$, nor v_j, for $j > i^2 + 1$, are vertices of $\mathcal{G}'_i$.

- Another easy case is
$$\#_w(i,2) \;=\; \#_v(i,1) \;=\; \#_v(i,2), \tag{5.3}$$
as it is not difficult to show that all three values are equal to 2^{i-1}.

- The next inequality
$$\#_w(i,j) \;>\; \#_w(i,j+1), \qquad \text{where } 2 \le j < i. \tag{5.4}$$
is also straightforward as we have $\#_w(i,j) = 2^{i-j+1}$ for $2 \le j \le i$.

- We next consider
$$\#_v(i,j) \;=\; \#_v(i,k), \quad \text{for } l^2-l+1 \le j \le k \le l^2+l,\ 1 \le l \le i. \tag{5.5}$$
The proof is by induction on i. Let $i=2$. In this case, from the definition of $\mathcal{G}'_2$ we have $2 = \#_v(2,1) = \#_v(2,2)$, and $3 = \#_v(2,3) = \#_v(2,4) = \#_v(2,5) = \#_v(2,6)$. So let the equation hold for $i = m-1$. We will show it for $i=m$. From the definition of $\mathcal{G}'_m$ we have $\#_v(m,j) = 2 \cdot \#_v(m-1,j)$ for every $j < m^2-m+1$. Hence, for $2 \le l < m$ the equation follows from our assumption.

  From the definition of $\mathcal{G}'_m$ we also have $\#_v(m,j) = |\mathcal{G}'_{m-1}|+1$ for $m^2-m+1 \le j \le m^2+l$. Hence, the equation follows for $l=m$. Both cases together yield Equation (5.5).

- The last inequality to prove is
$$\#_v(i,j) < \#_v(i,k), \quad \text{for } 1 \le j < l^2-l+1 \le k \le l^2+l,\ 2 \le l \le i. \tag{5.6}$$
Again, the induction is on i. Let $i=2$. From the definition of $\mathcal{G}'_2$ we have $\#_v(2,1) = \#_v(2,2) = 2 < 3 = \#_v(2,3) = \#_v(2,4) = \#_v(2,5) = \#_v(2,6)$. Let us assume that the inequality holds for $i=m-1$. We have to prove it for $i=m$.

  First, we consider the case $l < m$. From the definition of $\mathcal{G}'_m$ we have $\#_v(m,j) = 2\cdot\#_v(m-1,j)$ for all $j < l^2-l+1$ and $\#_v(m,k) = 2\cdot\#_v(m-1,k)$ for all $l^2-l+1 \le k \le l^2+l$. Together with the assumption this yields the inequality for the case $l<m$.

  Secondly, we have to examine the case $l=m$. Let us consider the vertex v_{m^2-m}, the vertex from $\mathcal{G}'_{m-1}$ in $\mathcal{G}'_m$ with the largest index. From the case $l<m$ we know that v_{m^2-m} is one of the most frequent vertices of $\mathcal{G}'_{m-1}$ in $\mathcal{G}'_m$. To complete the proof it suffices to show $\#_v(m,m^2-m) < \#_v(m,m^2-m+1)$ as we know from Equation (5.5) and the already established "$l<m$"-case. From the definition of $\mathcal{G}'_m$ and $\mathcal{G}'_{m-1}$ we have
$$\begin{aligned}\#_v(m,m^2-m) &= 2\cdot(|\mathcal{G}'_{m-2}|+1), \text{ and}\\ \#_v(m,m^2-m+1) &= |\mathcal{G}'_{m-1}|+1.\end{aligned}$$

With $|\mathcal{G}'_i| = 2^{i+1} - 2$ this gives

$$\begin{aligned} \#_v(m, m^2 - m) &= 2^m - 2, \text{ and} \\ \#_v(m, m^2 - m + 1) &= 2^m - 1. \end{aligned}$$

Hence, we have $\#_v(m, m^2 - m) < \#_v(m, m^2 - m + 1)$, which completes the proof of Equation (5.6).

Thus, the proof of Lemma 5.1.4 is completed. □

From Lemma 5.1.4 it follows that the vertices from $\mathcal{G}'(i)$ are the last vertices in the vertex ordering computed by the BMR-algorithm on input $\mathcal{G}'_i$. This is crucial for the next step of our analysis in which we examine the recursive calls produced by the BMR-algorithm on input $\mathcal{G}'_i$.

Lemma 5.1.5. *For $i \geq 4$, the BMR-algorithm on input $\mathcal{G}'_i$ recursively calls the BMR-algorithm at least $2i$ times with a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ as input. Here, modified means that all edges of $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ may additionally include at most half of the vertices of $\mathcal{G}'(i)$.*

Proof. We only examine the last $2i$ vertices processed by the BMR-algorithm. From Lemma 5.1.4 we know that these are exactly the vertices from $\mathcal{G}'(i)$—contained in the edges e_i and f_i. Let $v'_1, v'_2, \ldots, v'_{2i}$ be any ordering of these vertices. We consider the BMR-algorithm on that ordering.

Let the j-th vertex v'_j, $1 \leq j \leq 2i$, from the above ordering be the current partitioning vertex (line 3 of the BMR-algorithm). After partitioning (lines 5 to 8), the remaining hypergraph has the form

$$(\mathcal{G}'_{i-1} \cup \{\{w_i\}\}) \vee \{\{v'_1, \ldots, v'_{j-1}\} \cap x_i\},$$

where $x_i = f_i$ if $v'_j \in e_i$, and $x_i = e_i$ if $v'_j \in f_i$. Hence, the remaining hypergraph always is a $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ with at most half of the vertices from $\mathcal{G}'(i)$ in every edge.

Altogether, for each of the last $2i$ vertices the minimal transversals of a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ have to be computed. Note that a modified $\mathcal{G}'_3 \cup \{\{w_4\}\}$ has 15 edges of average size at least 5.4 and, thus, $a \geq 81$ (line 10). Hence, for $i \geq 4$ the last $2i$ vertices invoke recursive calls of the BMR-algorithm with a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ as input. □

With Lemma 5.1.5 at hand we can analyze the number of non-minimal transversals computed by the BMR-algorithm.

Lemma 5.1.6. *Let $i \geq 4$. For the number $\eta(i)$ of non-minimal transversals computed during a run of the BMR-algorithm on input $\mathcal{G}'_i$ we have $\eta(i) \geq 2^{i-1} \cdot i!$.*

Proof. From Lemma 5.1.5 it follows that there are $2i$ recursive calls with a modified $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ as input. Such a recursive call produces at least all of the minimal and some non-minimal transversals of $\mathcal{G}'_{i-1} \cup \{\{w_i\}\}$ augmented by the current partitioning vertex as transversals for $\mathcal{G}'_i$. But since at least the partitioning vertex is dispensable in these transversals, not one of them is minimal for $\mathcal{G}'_i$ and thus will not be part of the final output. There are at least $\eta(i-1) + |\mathrm{Tr}(\mathcal{G}'_{i-1})|$ such non-minimal transversals per recursive call. Hence, we have to solve the recurrence

$$\begin{aligned} \eta(i) &\geq 2i \cdot (\eta(i-1) + |\mathrm{Tr}(\mathcal{G}'_{i-1})|) \\ &\geq 2i \cdot \eta(i-1). \end{aligned}$$

As for the initial condition we have the following.

Claim 5.1.7. $\eta(3) = 34$.

Proof. We examine the BMR-algorithm on input $\mathcal{G}'_3$. Without loss of generality we assume that the BMR-algorithm processes the vertices in the order $w_3, w_2, v_1, v_2, \ldots, v_{12}$. When using w_3, w_2, or v_1 as partitioning vertex, nothing happens since the resulting hypergraph is empty.

The next partitioning vertex is v_2 and there remains the hypergraph with the three edges $\{w_3\}$, $\{w_2\}$, and $\{v_1\}$. The DL-algorithm is invoked and outputs one minimal transversal, which is augmented by v_2. The resulting transversal $\{w_3, w_2, v_1, v_2\}$ is minimal for $\mathcal{G}'_3$.

When using v_3 or v_4 as partitioning vertex, there remains the hypergraph with the four edges $\{w_3\}$, $\{w_2\}$, $\{v_1\}$, and $\{v_2\}$. The DL-algorithm computes the minimal transversal of this hypergraph, which is augmented by v_3 and respectively v_4. Obviously, the resulting transversals are not minimal since they contain the minimal transversal $\{w_3, w_2, v_1, v_2\}$. Hence, the BMR-algorithm has computed two non-minimal transversals of $\mathcal{G}'_3$.

When using v_5 or v_6 as partitioning vertex, there remains the hypergraph with the four edges $\{w_3\}$, $\{w_2, v_3, v_4\}$, $\{v_1, v_3, v_4\}$, and $\{v_2, v_3, v_4\}$. The DL-algorithm computes the three minimal transversals $\{w_3, v_3\}$, $\{w_3, v_4\}$, and $\{w_3, w_2, v_1, v_2\}$ and augments them by v_5 and respectively v_6. This yields four minimal transversals of $\mathcal{G}'_3$ and another two non-minimal transversals of $\mathcal{G}'_3$.

When using v_7, v_8, or v_9 as partitioning vertex, there remains a $\mathcal{G}'_2 \cup \{\{w_3\}\}$. Each time, the DL-algorithm is invoked to compute all five minimal transversals of $\mathcal{G}'_2 \cup \{\{w_3\}\}$. Each such computed minimal transversal of $\mathcal{G}'_2 \cup \{\{w_3\}\}$ is augmented by the current partitioning vertex. The resulting transversal is not minimal for $\mathcal{G}'_3$ since already the minimal transversals of $\mathcal{G}'_2 \cup \{\{w_3\}\}$ are minimal for $\mathcal{G}'_3$. Hence, the algorithm produces 15 non-minimal transversals for the vertices v_7, v_8, and v_9.

As for the vertices v_{10}, v_{11}, and v_{12} there remains a $(\mathcal{G}'_2 \cup \{\{w_3\}\}) \vee \{v_7, v_8, v_9\}$ after partitioning. For each such modified $\mathcal{G}'_2 \cup \{\{w_3\}\}$ the DL-algorithm as a subroutine is invoked to compute the minimal transversals since $a = 40 < 50$ (line 10 of the BMR-algorithm) for a modified $\mathcal{G}'_2 \cup \{\{w_3\}\}$. For each such call the DL-algorithm produces all five minimal transversals of $\mathcal{G}'_2 \cup \{\{w_3\}\}$ plus the three minimal transversals $\{v_7\}$, $\{v_8\}$, $\{v_9\}$. Each such computed transversal is augmented by the current partitioning vertex. This yields nine minimal transversals of $\mathcal{G}'_3$ and another 15 non-minimal transversals.

Altogether, the BMR-algorithm with input $\mathcal{G}'_3$ computes 34 non-minimal transversals. This yields $\eta(3) = 34$ and completes the claim. □

Hence, $\eta(3) \geq 2^2 \cdot 3!$ and we get $\eta(i) \geq 2^{i-1} \cdot i!$ by iteration, which completes the proof of Lemma 5.1.6. □

Putting all the pieces together we are able to give a superpolynomial lower bound on the running time of the BMR-algorithm.

Theorem 5.1.8. *The BMR-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log\log n)}$, where n denotes the size of the input and output.*

Proof. We consider the BMR-algorithm on input $\mathcal{G}'_i$. By $m_i = |V_{\mathcal{G}'_i}| \cdot (|\mathcal{G}'_i| + |\mathrm{Tr}(\mathcal{G}'_i)|)$ we denote an upper bound on the size of the input and output. For $i \geq 22$ we have

$$m_i = (i^2 + 2i - 1) \cdot \left(2^{i+1} - 2 + \frac{2i^3 + 3i^2 + i}{6}\right) \leq 2^{3i}.$$

The running time of the BMR-algorithm on input $\mathcal{G}'_i$ is at least $\eta(i)$, the number of generated non-minimal transversals. Thus, to analyze the running time we will show that $\eta(i)$ is superpolynomial in m_i. It suffices to show that

$$2^{i-1} \cdot i! > (2^{3i})^c, \qquad \text{for any constant } c.$$

This is equivalent to $i - 1 + \log(i!) > c \cdot 3i$, for any constant c. Using Stirling's formula we have $\log(i!) \geq i \cdot \log i - i$ and thus it suffices to show $i - 1 + i \cdot \log i - i > c \cdot 3i$, for any constant c. This is equivalent to

$$\frac{\log i}{3} - \frac{1}{3i} > c, \qquad \text{for any constant } c.$$

Since the last equation obviously holds for sufficiently large i, we have proven that $\eta(i)$ is superpolynomial in m_i, namely $\eta(i) = m_i^{\Omega(\log\log m_i)}$. □

5.1.4 The algorithm of Kavvadias and Stavropoulos

A first drawback of Berge-multiplication or the BMR-algorithm observed by Kavvadias and Stavropoulos [KS05] is the memory requirement. Since newly computed transversals have to be checked for minimality against the previously computed minimal transversals, all the previously generated minimal transversals have to be stored. The KS-algorithm tries to overcome this potentially exponential memory requirement by two techniques. The first is to combine vertices that belong to exactly the same hyperedges.

Definition 5.1.9 (Generalized Vertex)**.** *Let $\mathcal{H}$ be a hypergraph with vertex set V. The set $X \subseteq V$ is a* generalized vertex *of $\mathcal{H}$ if all vertices in X belong to exactly the same hyperedges of $\mathcal{H}$.*

A transversal possibly containing generalized vertices will be referred to as *generalized* transversal. While adding edge e_i, and hence generating the minimal generalized transversals of $\mathcal{H}_i$ out of the minimal generalized transversals of $\mathcal{H}_{i-1}$, the generalized vertices have to be updated according to e_i. Kavvadias and Stavropoulos characterize the following three types of generalized vertices X of a minimal generalized transversal t of $\mathcal{H}_{i-1}$.

- type α: $X \cap e_i = \emptyset$. Hence, X is a generalized vertex of $\mathcal{H}_i$.
- type β: $X \subset e_i$. Hence, X is a generalized vertex of $\mathcal{H}_i$.
- type γ: $X \cap e_i \neq \emptyset$ and $X \not\subset e_i$. Here, X is divided into $X_1 = X \setminus (X \cap e_i)$ and $X_2 = X \cap e_i$. Both X_1 and X_2 are generalized vertices of $\mathcal{H}_i$.

Let $\kappa_\alpha(t,i)$, $\kappa_\beta(t,i)$, and $\kappa_\gamma(t,i)$ denote the number of generalized vertices of type α, β, and γ in t according to e_i. When edge e_i is added, the minimal generalized transversal t of $\mathcal{H}_{i-1}$ has to be split into $2^{\kappa_\gamma(t,i)}$ generalized transversals of $\mathcal{H}_{i-1}$—the so-called *offsprings* of t—since all combinations of newly generalized vertices have to be generated. If $\kappa_\beta(t,i) \neq 0$, all these newly generated offsprings are also minimal transversals of $\mathcal{H}_i$. But if $\kappa_\beta(t,i) = 0$, there is a special offspring t_0 of t that contains all the X_1-parts of the γ-type generalized nodes of t. Hence, $t_0 \cap e_i = \emptyset$ and t_0 has to be augmented by a vertex from e_i to be a transversal of $\mathcal{H}_i$. All the other offsprings of t already are minimal transversals of $\mathcal{H}_i$ since they contain at least one X_2-part of a generalized vertex from t.

The second technique to overcome the potentially exponential memory requirement is based on the observation that Berge-multiplication is a form of breadth-first search through a "tree" of minimal transversals. At the ith-level of the "tree" the nodes are the minimal transversals of the partial hypergraph $\mathcal{H}_i$. The descendants of a minimal transversal t at level i are the minimal transversals of $\mathcal{H}_{i+1}$ that include t. Note that, since a node at level $i+1$ may have several ancestors at level i, the structure is not really a tree but very tree-like. The bottom level consists of the minimal transversals of $\mathcal{H}$. When cycling through this "tree" breadth-first, one has to wait very long for the first minimal transversal to be output and some nodes are visited several times because they have more than one ancestor. To overcome the long time that may pass until the first minimal transversal is output, the KS-algorithm uses a depth-first strategy. And to really cycle through a tree and not a tree-like structure with some cycles, Kavvadias and Stavropoulos introduce the notion of so-called appropriate vertices.

Definition 5.1.10 (Appropriate Vertex)**.** *Let $\mathcal{H} = \{e_1, \ldots, e_m\}$ be a hypergraph with vertex set V and let t be a minimal transversal of the partial hypergraph $\mathcal{H}_i$ of $\mathcal{H}$. A generalized vertex $v \subseteq V \setminus t$ at level i is an* appropriate vertex *for t if no other vertex in $t \cup \{v\}$ except v can be removed and the remaining set still be a transversal of $\mathcal{H}_i$. The set $appr(t, e)$ contains all appropriate vertices for t in edge e.*

Note that the special offspring t_0 of a minimal generalized transversal t of $\mathcal{H}_{i-1}$ has to be augmented by a vertex from $appr(t, e_i)$ only. All the other vertices from

e_i can be skipped. Expanding only with appropriate vertices ensures that no non-minimal transversals are generated and avoids regenerations. Another advantage is that the previously described transversal "tree" structure becomes a real tree (cf. the original paper [KS05] for more details).

All the described techniques—generalized vertices, depth-first strategy, appropriate vertices—together with the main idea of Berge-multiplication—processing the edges one after the other—are used in the KS-algorithm (cf. Algorithm 20 for the listing).

Algorithm 20 The KS-algorithm

express e_1 as a set of one generalized vertex
compute the transversal $t = \mathrm{Tr}(e_1)$
`addNextHyperedge`(t, e_2)

procedure `addNextHyperedge`(t, e_i)
update the set of generalized vertices
express t and e_i as sets of generalized vertices of level i
$l \leftarrow 1$
while `generateNextTransversal`(t, l) **do**
    if e_i is the last hyperedge **then**
        output t' without using generalized vertices
    else
        `addNextHyperedge`(t', e_{i+1})
        $l \leftarrow l + 1$

function `generateNextTransversal`(t, l)
if $\kappa_\beta(t, i) \neq 0$ **then**
    if $l \leq 2^{\kappa_\gamma(t,i)}$ **then**
        $t' \leftarrow$ the l-th offspring of t
        return `true`
    else
        return `false`
else if $\kappa_\beta(t, i) = 0$ **then**
    if $l \leq 2^{\kappa_\gamma(t,i)} - 1$ **then**
        $t' \leftarrow$ the l-th offspring of t except t_0
        return `true`
    else if $2^{\kappa_\gamma(t,i)} \leq l \leq 2^{\kappa_\gamma(t,i)} - 1 + |appr(t, e_i)|$ **then**
        $t' = t_0$ augmented by the $(l - 2^{\kappa_\gamma(t,i)} + 1)$-th vertex of $appr(t, e_i)$
        return `true`
else
    return `false`

After computing a first transversal (only one since it consists of a generalized vertex), the recursive `addNextHyperedge` procedure is called (note that t' is a global variable). Due to the usage of generalized vertices, the expansion of t is divided into two parts according to the presence (line 15 of the listing) or absence (line 21) of a generalized vertex of type β in t. If the minimal transversal t of $\mathcal{H}_{i-1}$ contains a generalized vertex of type β, all its offsprings intersect e_i and hence are minimal transversals of $\mathcal{H}_i$ (lines 16 to 20). If t does not contain a type β vertex, all its offsprings except t_0 intersect e_i and hence are minimal for $\mathcal{H}_i$ (lines 22 to 24). The offspring t_0 has to be augmented by every appropriate vertex (line 26).

The effect as shown by Kavvadias and Stavropoulos is that a newly generated transversal is minimal and that regenerations are avoided [KS05]. Since the KS-algorithm uses a depth-first strategy, it does not have to store all the minimal transversals of the subhypergraph $\mathcal{H}_{i-1}$ to compute the minimal transversals of $\mathcal{H}_i$. This yields a space requirement of the KS-algorithm that is polynomial in the input size $|\mathcal{H}|$ [KS05].

As for the running time, the KS-algorithm is experimentally shown [KS05] to be competitive to Berge-multiplication, the BMR-algorithm, and an implementation of Algorithm A of Fredman and Khachiyan [FK96, KBEG06] (cf. Section 5.3). We will show that the KS-algorithm is not output-polynomial.

First, we note that there are situations in which the KS-algorithm cannot find an appropriate vertex.

Example 5.1.11. Consider for example the hypergraph

$$\mathcal{H} = \{\{v_1, v_5\}, \{v_2, v_5\}, \{v_3, v_6\}, \{v_4, v_6\}, \{v_5, v_6\}\}.$$

Having processed all but the last edge, there are no generalized vertices left. We concentrate on the path down the transversal tree that corresponds to choosing v_1, v_2, v_3, and v_4. The intermediate transversal is $t = \{v_1, v_2, v_3, v_4\}$. The only edge left is $\{v_5, v_6\}$. But the KS-algorithm cannot find an appropriate vertex for t in this edge.

Hence, there are dead ends in the recursion tree of the KS-algorithm, namely leaves that do not contain a minimal transversal of the input $\mathcal{H}$. The next step is to find hypergraphs with too many such dead ends.

Lemma 5.1.12. *For $i \geq 3$, the number of dead ends the KS-algorithm has to visit for any of Takata's hypergraphs $\mathcal{G}_i$ as input is at least $2^{(i-2)\cdot 2^i+1}$, independent of the edge ordering.*

Proof. Consider the hypergraph family $\mathcal{G}_i$ of Takata defined in Section 5.1.1. First note that when the KS-algorithm adds the last edge of $\mathcal{G}_i$, there are no *proper* generalized vertices left (generalized vertices that are not singleton sets). We want to argue that the same already holds for the penultimate step, hence, that $\mathcal{G}_i \setminus \{e\}$ has no proper generalized vertex, for any edge $e \in \mathcal{G}_i$. Assume otherwise that after processing all of $\mathcal{G}_i \setminus \{e\}$'s edges there remains a proper generalized vertex $X \subseteq V$. As $\mathcal{G}_i$ has no proper generalized vertices, we have $X \subseteq e$. Let

e be composed of the $\mathcal{A}$ and $\mathcal{C}$ component in $\mathcal{G}_i$'s definition (the argumentation is analogous for the other cases) and consider two different vertices $v, u \in X$. If both v and u are vertices in the $\mathcal{A}$ component we have a contradiction as already $\mathcal{A}$ contains an edge f that contains v but not u. This edge appears in $|\mathcal{C}| + |\mathcal{D}|$ edges of $\mathcal{G}_i \setminus \{e\}$. Hence, not both v and u can be vertices in X as they would have been split according to the f copies in prior steps of the KS-algorithm's run (an analogous argument shows that not both are in $\mathcal{C}$).

The remaining possibility (minus symmetry) is that v is from $\mathcal{A}$ and u is from $\mathcal{C}$. But note that $\mathcal{G}_i \setminus \{e\}$ contains an edge f with $v \in f$ but f is composed of the $\mathcal{A}$ and $\mathcal{D}$ part of $\mathcal{G}_i$. Again, this shows that v and u cannot both be vertices in X as they would have been split in a prior step.

Altogether, we now know that before processing the last edge, there cannot be proper generalized vertices in $\mathcal{G}_i \setminus \{e\}$. From Lemma 5.1.2 it follows that, whatever ordering of the edges is chosen, there are at least $2^{(i-2)\cdot 2^i+2}$ nodes in the penultimate level of the transversal tree described above Definition 5.1.10. The bottom level of the tree obviously contains $|\mathrm{Tr}(\mathcal{G}_i)|$ many nodes—one for each minimal transversal. Since $|\mathrm{Tr}(\mathcal{G}_i)| = 2^{2^i-1}$ (cf. Lemma 5.1.2), there is a decrease in the number of nodes from the penultimate level to the bottom level for $i \geq 3$. This decrease can only be caused by dead ends in the penultimate level. Hence, for $i \geq 3$ there are at least $2^{(i-2)\cdot 2^i+2} - 2^{2^i-1} \geq 2^{(i-2)\cdot 2^i+1}$ many dead ends in the penultimate level. □

Using Lemma 5.1.12 we can show that the KS-algorithm is not output-polynomial.

Theorem 5.1.13. *The KS-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log\log n)}$, where n denotes the size of the input and output.*

Proof. We consider the KS-algorithm on input $\mathcal{G}_i$. By $m_i = |V_{\mathcal{G}_i}| \cdot (|\mathcal{G}_i| + |\mathrm{Tr}(\mathcal{G}_i)|)$ we denote an upper bound on the size of $\mathcal{G}_i$ and $\mathrm{Tr}(\mathcal{G}_i)$. From Lemma 5.1.2 we have $m_i = 4^i \cdot (2^{2(2^i-1)} + 2^{2^i-1})$, which results in $m_i \leq 2^{2^{i+2}}$.

Let $\hat{\eta}(i)$ denote the number of dead end situations visited by the KS-algorithm on input $\mathcal{G}_i$. The time, the KS-algorithm needs to compute $\mathrm{Tr}(\mathcal{G}_i)$, is at least the number of dead end situations visited. Since the KS-algorithm visits the transversal tree depth-first, it visits all the dead end situations in the penultimate level of the tree. With Lemma 5.1.12 we have $\hat{\eta}(i) \geq 2^{(i-2)\cdot 2^i+1}$ for $i \geq 3$. Thus, to analyze the running time we will show that $\hat{\eta}(i)$ is superpolynomial in m_i. It suffices to show that $2^{(i-2)\cdot 2^i} > (2^{2^{i+2}})^c$, for any constant c. This is equivalent to $i - 2 > 4c$, for any constant c. Since this obviously holds for large enough i, we have proven that $\hat{\eta}(i)$ is superpolynomial in m_i, namely $\hat{\eta}(i) = m_i^{\Omega(\log\log m_i)}$. □

Algorithm 21 The HBC-algorithm

Input: a hypergraph $\mathcal{H}$ on vertex set V
1: $\mathit{Tr} \leftarrow \{\{v\} : v \in E$ for each $E \in \mathcal{H}\}$
2: $C_1 \leftarrow \{\{v\} : v \in V\} \setminus \mathit{Tr}$
3: $i \leftarrow 1$
4: **while** $C_i \neq \emptyset$ **do**
5:     **for all** $a, b \in C_i, |a \cap b| = i - 1$ **do**
6:         $c \leftarrow a \cup b$
7:         **if** $c \setminus \{v\} \in C_i$ for all $v \in c$ **then**
8:             **if** $c \setminus \{v\}$ hits fewer edges of $\mathcal{H}$ than c for all $v \in c$ **then**
9:                 **if** c is a transversal of $\mathcal{H}$ **then**
10:                     $\mathit{Tr} \leftarrow \mathit{Tr} \cup \{c\}$
11:                 **else**
12:                     $C_{i+1} \leftarrow C_{i+1} \cup \{c\}$
13:     $i \leftarrow i + 1$
14: **output** Tr

5.2 The algorithm of Hébert, Bretto, and Crémilleux

The HBC-algorithm [HBC07, Héb07] (cf. Algorithm 21 for the listing) is a level-wise approach for computing all minimal transversals similar to the classic Apriori algorithm [AS94, GKM+03, MT97] (cf. Section 7.4). Note that our presentation of the algorithm slightly differs from the original paper [HBC07] as we avoid using Galois connections and the like.

The HBC-algorithm generates transversal candidates in a level-wise manner. In the first step, one element subsets of V are candidates for being minimal transversals. Those that really are transversals are added to the set Tr (line 1 of the listing) that finally will contain all minimal transversals of $\mathcal{H}$. All other one element subsets of V are added (line 2) to the candidate set C_1 of level 1. In the $(i+1)$-th step, the HBC-algorithm combines candidates of the i-th step that have a large enough intersection (lines 5 and 6). For each candidate c it is then checked whether all its subsets are candidates at level i and whether c hits more edges than all its subsets (lines 7 and 8). If so, a final check determines whether c is a transversal or not and, hence, whether c has to be added to Tr or C_{i+1}, respectively (lines 9 to 12).

In [HBC07] it is claimed that the HBC-algorithm on input $\mathcal{H}$ runs in $O(2^{t(\mathcal{H})} \cdot |\mathrm{Tr}(\mathcal{H})|)$ time, where $t(\mathcal{H})$ denotes the size of a largest minimal transversal of $\mathcal{H}$. This bound seems to be intuitive as the HBC-algorithm checks all the subsets of each minimal transversal. It would also solve a longstanding open question. Namely, polynomial time decision of TransHyp when edges are logarithmically size-bounded. Unfortunately, as we will show, the claimed upper bound is wrong.

We shall give a lower bound that also shows the HBC-algorithm not to be output-polynomial.

To obtain our lower bound we will observe the behavior of the HBC-algorithm on the Takata hypergraphs (cf. Definition 5.1.1) that we also used in previous sections. The idea we use in our analysis is probably best described by an example.

Example 5.2.1. Consider Takata's hypergraph

$$\mathcal{G}_1 = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}\}$$

as input and note that $\mathrm{Tr}(\mathcal{G}_1) = \{\{v_1, v_2\}, \{v_3, v_4\}\}$. What may have led to the running time claim in [HBC07] is the impression that only subsets of sets that are finally part of the output (minimal transversals in our case) are processed by the algorithm as this is true in other scenarios where the Apriori technique is used. But consider the set $\{v_1, v_3\}$ in our example. It is an element of the candidate set C_2 in the HBC-algorithm as it hits more edges of $\mathcal{G}_1$ than its one element subsets which all are contained in C_1. But $\{v_1, v_3\}$ is not subset of any minimal transversal. The same holds for $\{v_1, v_4\}$, $\{v_2, v_3\}$, and $\{v_2, v_4\}$. All are candidates in C_2 but not contained in any minimal transversal.

As for $\mathcal{G}_1$, the O-notation assures that the claimed bound of [HBC07] holds. But for larger and larger i we show that the HBC-algorithm on input $\mathcal{G}_i$ generates too many candidates to run in $O(2^{t(\mathcal{G}_i)} \cdot |\mathrm{Tr}(\mathcal{G}_i)|)$ time. Thereby, we also show the HBC-algorithm not to be output-polynomial. A lower bound for the number of candidates produced is given in the following lemma.

Lemma 5.2.2. *The number of candidates generated by the HBC-algorithm on input $\mathcal{G}_i$ is at least $2^{(i-2)\cdot 2^i+2}$ for $i \geq 2$.*

Proof. Fix any edge e of $\mathcal{G}_i$ and note that each element of $\mathrm{Tr}(\mathcal{G}_i \setminus \{e\}) \setminus \mathrm{Tr}(\mathcal{G}_i)$ is generated as a candidate by the HBC-algorithm. This can be seen as follows. Each element t of $\mathrm{Tr}(\mathcal{G}_i \setminus \{e\}) \setminus \mathrm{Tr}(\mathcal{G}_i)$ is a minimal transversal of $\mathcal{G}_i \setminus \{e\}$ and thus hits more edges of $\mathcal{G}_i \setminus \{e\}$ than all of t's proper subsets. All of t's subsets are candidates in the course of the HBC-algorithm running on input $\mathcal{G}_i \setminus \{e\}$ as t finally is produced as output. But note that t then also hits more edges of $\mathcal{G}_i$ than all its proper subsets and that all these subsets then also are candidates in the course of the HBC-algorithm running on input $\mathcal{G}_i$. Hence, each element of $\mathrm{Tr}(\mathcal{G}_i \setminus \{e\}) \setminus \mathrm{Tr}(\mathcal{G}_i)$ hits more edges of $\mathcal{G}_i$ than all its subsets and all these subsets also are candidates at a lower level. But not one of the elements of $\mathrm{Tr}(\mathcal{G}_i \setminus \{e\}) \setminus \mathrm{Tr}(\mathcal{G}_i)$ is a minimal transversal of $\mathcal{G}_i$ and, thus, on input $\mathcal{G}_i$ these elements are produced as candidates only. From Lemma 5.1.2 we have $|\mathrm{Tr}(\mathcal{G}_i \setminus \{e\}) \setminus \mathrm{Tr}(\mathcal{G}_i)| \geq 2^{(i-2)\cdot 2^i+2}$ independent of the choice of e, which completes the proof of Lemma 5.2.2. □

In order to show that the bound given in [HBC07] is wrong, we need the following observation on the size of a largest minimal transversal of $\mathcal{G}_i$.

Lemma 5.2.3. *The size $t(\mathcal{G}_i)$ of a largest minimal transversal of $\mathcal{G}_i$ is 2^i.*

Proof. Note that by the definition of $\mathcal{G}_i$ we have $t(\mathcal{G}_i) = 2 \cdot t(\mathcal{G}_{i-1})$ and with the initial condition $t(\mathcal{G}_0) = 1$ we get $t(\mathcal{G}_i) = 2^i$ by iteration. □

We are now ready to give a lower bound on the running time of the HBC-algorithm showing it not to be output-polynomial and proving the upper bound claimed by Hébert et. al. [HBC07] to be wrong.

Theorem 5.2.4. *The HBC-algorithm is not output-polynomial. Its running time is at least $n^{\Omega(\log\log n)}$, where n denotes the size of the input and output. Furthermore, the $O(2^{t(\mathcal{H})} \cdot |\mathrm{Tr}(\mathcal{H})|)$ upper time bound stated in [HBC07] is wrong.*

Proof. We consider the HBC-algorithm on input $\mathcal{G}_i$. By $m_i = |V_{\mathcal{G}_i}| \cdot (|\mathcal{G}_i| + |\mathrm{Tr}(\mathcal{G}_i)|)$ we denote an upper bound on the size of $\mathcal{G}_i$ and $\mathrm{Tr}(\mathcal{G}_i)$. From Lemma 5.1.2 we have $m_i = 4^i \cdot (2^{2(2^i-1)} + 2^{2^i-1})$, which results in $m_i \leq 2^{2^{i+2}}$.

Let $\gamma(i)$ denote the number of candidates generated by the HBC-algorithm on input $\mathcal{G}_i$. The time, the HBC-algorithm needs to compute $\mathrm{Tr}(\mathcal{G}_i)$, is at least the number of candidates generated. With Lemma 5.2.2 we have $\gamma(i) \geq 2^{(i-2)\cdot 2^i+1}$ for $i \geq 2$. Thus, to analyze the running time we will show that $\gamma(i)$ is superpolynomial in m_i. It suffices to show that $2^{(i-2)\cdot 2^i} > (2^{2^{i+2}})^c$, for any constant c. This is equivalent to $i - 2 > 4c$, for any constant c. Since this obviously holds for large enough i, we have proven that $\gamma(i)$ is superpolynomial in m_i, namely $\gamma(i) = m_i^{\Omega(\log\log m_i)}$ which gives the stated lower bound.

To prove the second part, namely that $O(2^{t(\mathcal{H})} \cdot |\mathrm{Tr}(\mathcal{H})|)$ is not an upper time bound for the HBC-algorithm we shall show that$\gamma(i)$ is superpolynomial in $m'_i = 2^{t(\mathcal{G}_i)} \cdot |\mathrm{Tr}(\mathcal{G}_i)|$ for large enough i. From Lemmata 5.1.2 and 5.2.3 we have $m'_i = 2^{2^i} \cdot 2^{2^i-1} = 2^{2\cdot 2^i - 1}$. Hence, it suffices to show that $2^{(i-2)\cdot 2^i} > (2^{2\cdot 2^i})^c$, for any constant c. This is equivalent to $i - 2 > 2c$, for any constant c. Since this obviously holds for large enough i, we have proven that $\gamma(i)$ is superpolynomial in m'_i and hence the $O(2^{t(\mathcal{H})} \cdot |\mathrm{Tr}(\mathcal{H})|)$ upper time bound claimed in [HBC07] is wrong. □

5.3 The algorithms of Fredman and Khachiyan

In 1996 Fredman and Khachiyan developed two Monet algorithms, FK-algorithm A and the improved version FK-algorithm B [FK96]. There is a nice survey on the FK-algorithms and follow up work [EMG08].

Both FK-algorithms exploit the self-reducibility of Monet. Namely, φ and ψ are equivalent iff setting any variable x to `false` resp. `true` yields two respectively equivalent DNF/CNF-pairs.

In case of non-equivalence, both FK-algorithms return an assignment $\mathcal{A}$ with $\mathcal{A}(\varphi) \neq \mathcal{A}(\psi)$ as a witness for non-equivalence. Both algorithms work recursively, but before exploiting the self-reducibility they check some basic conditions that can easily guarantee non-equivalence in case of monotone normal forms.

5.3.1 Preconditions

Let (φ, ψ) be a pair of an irredundant, monotone DNF φ and an irredundant, monotone CNF ψ that are equivalent. Then the following three conditions hold. Firstly,

$$m \cap c \neq \emptyset \text{ for any monomial } m \in \varphi \text{ and any clause } c \in \psi. \tag{5.7}$$

Assume there exists a monomial $m \in \varphi$ and a clause $c \in \psi$ with $m \cap c = \emptyset$. Now consider the assignment $\mathcal{A} = m$ and note that $\mathcal{A}(\varphi) = 1$ and $\mathcal{A}(\psi) = 0$.

Secondly,

$$\varphi \text{ and } \psi \text{ must contain exactly the same variables.} \tag{5.8}$$

Assume that there is a variable x in φ that is not present in ψ (the argumentation is similar if ψ contains a "new" variable). Let m be a monomial of φ containing x and consider the assignment $\mathcal{A} = m \setminus \{x\}$. We have $\mathcal{A}(\varphi) = 0$ and $\mathcal{A}(\psi) = 1$ as $\mathcal{A}$ has a non-empty intersection with every clause of ψ due to condition (5.7).

Thirdly,

$$\max\{|m| : m \in \varphi\} \leq |\psi|, \quad \max\{|c| : c \in \psi\} \leq |\varphi|. \tag{5.9}$$

Assume that there is a monomial $m \in \varphi$ that contains more variables than clauses are contained in ψ (the argumentation is similar if ψ contains a clause that is "too large"). Now let $m' \subset m$ be a proper subset of m satisfying $m' \cap c \neq \emptyset$ for any clause c of ψ. Consider the assignment $\mathcal{A} = m'$ and note that $\mathcal{A}(\varphi) = 0$ and $\mathcal{A}(\psi) = 1$.

Conditions (5.7)–(5.9) can be easily tested in linear respectively quadratic time, which is done by both FK-algorithms as a preprocessing step. We now come to the description of FK-algorithm A. In Section 5.3.3 we then discuss the improvements of FK-algorithm B.

5.3.2 FK-algorithm A

FK-algorithm A uses an additional precondition that holds for any equivalent (φ, ψ) pair. Namely,

$$\sum_{m \in \varphi} 2^{v-|m|} + \sum_{c \in \psi} 2^{v-|c|} \geq 2^v, \text{ where } v \text{ is the number of variables.} \tag{5.10}$$

The left hand side of condition (5.10) sums up the assignments that satisfy φ and the assignments that do not satisfy ψ. Hence, if the left hand side of condition (5.10) is smaller than 2^v, there must be an assignment $\mathcal{A}^*$ that does not satisfy φ but ψ. Such an assignment $\mathcal{A}^*$ can then be iteratively found as follows. Start with the empty assignment and at step i include variable x_i iff $\ell(\mathcal{A}^*_{i-1} \cup \{x_i\}) \leq \ell(\mathcal{A}^*_{i-1})$, where $\mathcal{A}^*_{i-1}$ is the partial result from step $i-1$ and $\ell(\mathcal{A})$ gives the number of monomials of φ satisfied by $\mathcal{A}$ plus the number of clauses of ψ not satisfied by $\mathcal{A}$. Hence, $\mathcal{A}^*_i$ is computed in a way as to minimize the value of ℓ where ℓ is similar to the left hand side of condition (5.10). Fredman and Khachiyan [FK96] mainly include condition (5.10) in their algorithm A to ensure the existence of a sufficiently frequent variable that then guarantees the validity of estimations made in their worst-case analysis.

A pseudocode listing of FK-algorithm A is given as Algorithm 22. We give some further brief remarks. As for the initial call of FK-algorithm A, the global variable $\mathcal{A}$ is the empty set. Note that we already discussed how appropriate assignments are found in case of violation of conditions (5.7)–(5.10).

In case of small inputs consisting of at most one term each (line 3 of the listing), there are only very few possibilities. If there are no variables at all, φ is the empty DNF (which is unsatisfiable) or contains the empty monomial (which is valid). The equivalent CNF of the empty DNF contains the empty clause only, and the equivalent CNF of the empty monomial is the empty CNF. Hence, if the inputs are not equivalent but contain no variables, any assignment serves as a witness. If the formulas contain variables but only one term each, the previous check of condition (5.8) already ensures equivalence as then the formulas must be identical!

As for the recursive process, the FK-algorithm A decomposes the original input instance (φ, ψ) as follows. It selects a "splitting" variable x that appears with frequency at least $1/\log(|\varphi| + |\psi|)$ in either φ or ψ. The existence of such a variable is ensured by condition (5.10) [FK96]. Then φ and ψ can be rewritten as

$$\begin{aligned} \varphi &\equiv (x \wedge \varphi_0) \vee \varphi_1, \\ \psi &\equiv (x \vee \psi_0) \wedge \psi_1, \end{aligned}$$

where φ_1 (resp. ψ_1) contains the monomials (resp. clauses) of φ (resp. ψ) that do not contain x and φ_0 (resp. ψ_0) are the other monomials (resp. clauses) from which x was excluded. Now deciding equivalence of (φ, ψ) is equivalent to deciding equivalence of the two smaller problems

$$(\varphi_1, \psi_0 \wedge \psi_1), \tag{5.11}$$

$$(\varphi_0 \vee \varphi_1, \psi_1). \tag{5.12}$$

Note that (5.11) corresponds to setting x to `false` in the original instance (φ, ψ) whereas (5.12) corresponds to setting x to `true`. Hence, if the call on subproblem (5.11) returns an assignment showing non-equivalence, the original call can return exactly this assignment (remember the set notion of assignments). In case that the second subproblem (5.12) is not equivalent, the algorithm adds the splitting variable x to the assignment.

As for the worst-case analysis, Fredman and Khachiyan show the following.

Proposition 5.3.1 ([FK96]). *FK-algorithm A runs in time $n^{O(\log^2 n)}$.*

The main tool in their analysis of FK-algorithm A is to base the estimation of the number of recursive calls on the fact that for equivalent normal forms condition (5.10) holds and hence the splitting variable is sufficiently frequent.

Algorithm 22 The FK-algorithm A (FK-A)

Input: irredundant, monotone DNF φ and CNF ψ
Output: $\emptyset$ in case of equivalence; otherwise, assignment $\mathcal{A}$ with $\mathcal{A}(\varphi) \neq \mathcal{A}(\psi)$

make φ and ψ irredundant
if one of conditions (5.7)–(5.10) is violated **then return** appropriate assignment
if $|\varphi| \cdot |\psi| \leq 1$ **then return** appropriate assignment found by a trivial check
else
  find a variable x appearing with frequency $\geq 1/\log(|\varphi| + |\psi|)$ in either φ or ψ
  $\mathcal{A} \leftarrow$ FK-A$(\varphi_1, \psi_0 \wedge \psi_1)$
  if $\mathcal{A} = \emptyset$ **then**
    $\mathcal{A} \leftarrow$ FK-A$(\varphi_0 \vee \varphi_1, \psi_1)$
    if $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
return $\mathcal{A}$

As for the practical performance, an experimental study of a randomized version of FK-algorithm A showed it to be quite efficient [KBEG06]. There is also a version by Tamaki designed to run with polynomial space [Tam00].

5.3.3 FK-algorithm B

FK-algorithm A does not exploit the fact that the second recursive call is only performed if the first call did not yield a witness for non-equivalence, but so does FK-algorithm B. Assume that the input (5.11) of the first recursive call is equivalent. Now in the second call, we try to find an assignment $\mathcal{A}$ with $\mathcal{A}(\varphi_0 \vee \varphi_1) \neq \mathcal{A}(\psi_1)$ (a witness for the non-equivalence of the second pair (5.12)). As φ_1 is equivalent to $\psi_0 \wedge \psi_1$ this then gives $\mathcal{A}(\varphi_0) \vee \mathcal{A}(\psi_0 \wedge \psi_1) \neq \mathcal{A}(\psi_1)$. If now $\mathcal{A}(\psi_0) = 1$, we have $\mathcal{A}(\varphi_0) \vee \mathcal{A}(\psi_1) \neq \mathcal{A}(\psi_1)$, which implies $\mathcal{A}(\psi_1) = 0$ and $\mathcal{A}(\varphi_0) = 1$. This is a contradiction to condition (5.7), which says that every clause of ψ_1 has a non-empty intersection with every monomial of φ_0. Hence, actually, for the second recursive call on (5.12) it suffices to find an assignment $\mathcal{A}$ with $\mathcal{A}(\psi_0) = 0$ and $\mathcal{A}(\psi_1) \neq \mathcal{A}(\varphi_0)$. As for $\mathcal{A}(\psi_0) = 0$, note that we only have to check the maximal assignments not satisfying ψ_0, of which there are exactly $|\psi_0|$ (for each clause $c \in \psi_0$ the assignment that does not contain exactly the variables of c). Hence, for each clause c of ψ_0, FK-algorithm B is recursively called on an adjusted pair (φ_0^c, ψ_1^c), where the superscript c denotes that all variables from c are set to `false` in the respective formula.

Note that in case we started testing equivalence of φ and ψ by first examining the second pair (5.12), an analogous argumentation yields that pair (5.11) then is equivalent to finding an assignment $\mathcal{A}$ with $\mathcal{A}(\varphi_0) = 1$ and $\mathcal{A}(\varphi_1) \neq \mathcal{A}(\psi_0)$. Hence, for each monomial m of φ_0, FK-algorithm B is recursively called on an adjusted pair (φ_1^m, ψ_0^m), where the superscript m in this case denotes that all variables from m are set to `true` in the respective formula. Note that in case of non-equivalence we now also have to include the corresponding monomial m in the respective witness (remember our set notion of assignments).

A pseudocode listing of FK-algorithm B is given as Algorithm 23. The algorithm

Algorithm 23 The FK-algorithm B (FK-B)

Input: irredundant, monotone DNF φ and CNF ψ

Output: $\emptyset$ in case of equivalence; otherwise, assignment $\mathcal{A}$ with $\mathcal{A}(\varphi) \neq \mathcal{A}(\psi)$

1: make φ and ψ irredundant
2: $\nu = |\varphi| \cdot |\psi|$;
3: **if** one of conditions (5.7)–(5.9) is violated **then return** appropriate assignment
4: **if** $\min\{|\varphi|, |\psi|\} \leq 2$ **then return** appropriate assignment found by a trivial check
5: **else**
6:     choose some variable x from the formulas
7:     $\varepsilon(\nu) \leftarrow 1/\chi(\nu)$
8:     $\varepsilon_x^{\varphi} \leftarrow |\{m \in \varphi : x \in m\}|/|\varphi|$
9:     $\varepsilon_x^{\psi} \leftarrow |\{c \in \psi : x \in c\}|/|\psi|$;
10:     **if** $\varepsilon_x^{\varphi} \leq \varepsilon(\nu)$ **then**
11:         $\mathcal{A} \leftarrow$ FK-B$(\varphi_1, \psi_0 \wedge \psi_1)$
12:         **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A}$
13:         **for all** clauses $c \in \psi_0$ **do**
14:             $\mathcal{A} \leftarrow$ FK-B(φ_0^c, ψ_1^c)
15:             **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
16:     **else if** $\varepsilon_x^{\psi} \leq \varepsilon(\nu)$ **then**
17:         $\mathcal{A} \leftarrow$ FK-B$(\varphi_0 \vee \varphi_1, \psi_1)$
18:         **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
19:         **for all** monomials $m \in \varphi_0$ **do**
20:             $\mathcal{A} \leftarrow$ FK-B(φ_1^m, ψ_0^m)
21:             **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup m$
22:     **else**
23:         $\mathcal{A} \leftarrow$ FK-B$(\varphi_1, \psi_0 \wedge \psi_1)$
24:         **if** $\mathcal{A} = \emptyset$ **then**
25:             $\mathcal{A} \leftarrow$ FK-B$(\varphi_0 \vee \varphi_1, \psi_1)$
26:             **if** $\mathcal{A} \neq \emptyset$ **then return** $\mathcal{A} \cup \{x\}$
27: **return** $\mathcal{A}$

exploits the above described decomposition of the second recursive call whenever useful. The decision, if it is useful and which of the two recursive calls is performed first, is done according to the frequency of the "splitting" variable x (chosen in line 5). Therefore, in line 6 the algorithm computes a "threshold" frequency $\varepsilon(\nu) = 1/\chi(\nu)$, where $\nu = |\varphi| \cdot |\psi|$ is the *volume* of φ and ψ and χ is the function defined by $\chi(n)^{\chi(n)} = n$. Note that $\chi(n) \sim \log n / \log \log n = o(\log n)$.

If the frequency of the splitting variable in φ is less than $\varepsilon(\nu)$, the FK-algorithm B uses (5.11) as input of the first recursive call and uses the more sophisticated version of the second call to solve (5.12). If elsewise the frequency of the splitting variable in ψ is less than $\varepsilon(\nu)$, the FK-algorithm B uses (5.12) as input of the first recursive call and then solves (5.11) using the improved decomposition. If otherwise the splitting variable is more frequent than $\varepsilon(\nu)$ in both, φ and ψ, the FK-algorithm B just branches as FK-algorithm A. Note that condition (5.10) is not necessary any more as we do not have to guarantee a sufficiently frequent splitting variable.

As for the easy cases in line 3, note that if $\min\{|\varphi|, |\psi|\} \leq 1$ we have a similar argumentation as for FK-algorithm A. If the formulas are not empty, one contains just one term and the other then has to contain singleton terms for each variable. Otherwise, it is easy to give a witness for non-equivalence according to the situation. Similarly, if the minimum is 2, a brute force multiplication of the two terms and a following comparison to the other normal form is sufficiently efficient.

As for the worst-case analysis, Fredman and Khachiyan give a better upper bound on the running time than for FK-algorithm A.

Proposition 5.3.2 ([FK96])**.** *FK-algorithm B runs in time $n^{o(\log n)}$.*

The main tool in the analysis is that the new branching guarantees better bounds on the size of the inputs of recursive calls than in the analysis of FK-algorithm A.

As for the practical performance, none of the so far published experimental studies of Monet algorithms includes FK-algorithm B as the assumption is that, despite the theoretically worse running time, FK-algorithm A will perform better than FK-algorithm B in experiments [BMR03, KBEG06].

As we will show in Section 6.2, this assumption has to be adjusted, as in fact FK-algorithm B turns out to be competitive even in practical experimentation.

5.4 Concluding remarks

We have proven superpolynomial lower bounds for the DL-, the BMR-, the KS-, and the HBC-algorithm in terms of the size of the input and output. The bounds show that, like the underlying Berge-multiplication algorithm, these algorithms are not output-polynomial.

We are not aware of any other nontrivial lower bounds for algorithms generating the transversal hypergraph, although we conjecture that none of the known algorithms is output-polynomial. Extending the existing lower bounds to other algorithms does not seem to be so straightforward.

Consider for instance Takata-multiplication, an improvement of Berge-multiplication suggested by Takata [Tak07]. Roughly speaking, the idea is not to process the edges one after the other but to partition them and multiply them using some more sophisticated partitioning scheme. Very recently, Elbassioni proved a quasi-polynomial upper bound on the running time of Takata-multiplication [Elb06b]. But giving a superpolynomial lower bound for Takata-multiplication requires the construction of new hypergraphs. Takata's hypergraphs $\mathcal{G}_i$ and our hypergraphs $\mathcal{G}_i'$ are solved too fast by Takata-multiplication.

There are also no nontrivial lower bounds known for FK-algorithms A and B. Though Gurvich and Khachiyan [GK97] note that it should be possible to give a superpolynomial lower bound for FK-algorithm A using formulas associated to hypergraphs very similar to the $\mathcal{G}_i$, the proof is still open. Giving a lower bound for FK-algorithm B—considered to be the fastest known Monet algorithm—seems to be even more involved.

Chapter 6

Algorithm Engineering

As for evaluating the practical performance of MONET algorithms, there have been several experimental studies [BMR03, DL05, KBEG06, KS05, LJ03, TT02, US03]. Unfortunately, all have some lack of coverage. None of the published studies includes the FK-algorithm B by Fredman and Khachiyan [FK96], the MONET algorithm with the best known worst-case performance (cf. Section 5.3). Actually, the folklore assumption in the literature is that the operations performed by FK-algorithm B to ensure recursion on smaller sub-problems compared to the less involved FK-algorithm A just complicate the implementation and do only pay off theoretically [BMR03, KBEG06]. Thus, unfortunately, the practical performance of FK-algorithm B has not been systematically examined. Hence, it is not clear at all, which of the currently known algorithms is the best choice on which kind of instances.

In this chapter, we start working on closing this gap. In contrast with the folklore assumption, we experimentally show FK-algorithm B to be competitive and even superior to FK-algorithm A on many instances.

The chapter is organized as follows. In Section 6.1 we briefly discuss some details of the implementation of the FK-algorithms. The experimental setting and the corresponding results are stated in Section 6.2. Some concluding remarks follow in Section 6.3.

6.1 A few implementation details

The FK-algorithms were implemented using Java. We decided to represent variable sets—like terms and assignments—as *bitmaps*, which is just a sequence of bits where bit i is set iff x_i is contained in the corresponding term. This allows us to process operations on formulas as logical operations on bitmaps, which can be performed very fast if using some Java predefined data type. Hence, our choice for internally representing bitmaps is the `long` data type—one of the primitive Java data types. Note that we can use only 63 of the 64 bits of a `long` variable (the remaining bit being the reserved sign bit). This restricts us to formulas with at most 63 variables. Hence, we compared several other possibilities of representing bitmaps. Namely, we tried using `BigInteger`, `BitSet`, and an own structure composed of an `Array` of sufficiently many `long`'s. Somehow surprisingly, the best overall performance for formulas with more than 63 variables was achieved by our own array of `long`'s structure that beats the Java proprietary data types in our pretests. For formulas with less than 63 variables, not that surprisingly, a single `long` turns out to be the best choice.

Now that we know how to represent variable sets, we still have to internally represent complete normal forms. In our pretests we compared implementations of the FK-algorithms using the classes `Array` and `Vector` to store a set of bitmaps. An advantage of `Array` is the faster access compared to `Vector`. In contrast, `Vector` might have advantages in the process of making inputs irredundant as in an `Array` implementation we have to manually close "gaps" in the array caused by redundant terms. Our pretests favored the `Array` implementation.

Hence, in our implementations a formula is stored as an `Array` of bitmaps—that itself are stored as `long`'s respectively `Array`'s of `long`'s according to the number of variables.

Both FK-algorithms use an irredundantization procedure in line 1. But note that, when making the DNF/CNF of a recursive call irredundant, not the whole DNF/CNF has to be considered as the terms that included the splitting variable cannot be redundant in the resulting formulas. Redundancy can only appear in the formulas $\varphi_0 \vee \varphi_1$ and $\psi_0 \wedge \psi_1$ and there only terms in φ_1 and ψ_1 have to be checked for redundancy. Hence, in our implementations, irredundantization is always carried out *before* a recursive call. This significantly speeds up computation. Note that in case of FK-algorithm B we can also save some processing time in irredundantization of the many calls replacing the second call of FK-algorithm A. If we set the variables of a monomial m of φ_0 to `true` only monomials of φ_1^m may be redundant. Analogously, in case of setting the variables of a clause c of ψ_0 to `false` only clauses of ψ_1^c may be redundant.

In our implementations we also slightly adopt the choice of the splitting variable to speed up computation. As for FK-algorithm A we do not check which variables are sufficiently frequent but choose a variable with the highest frequency in either φ or ψ. As for FK-algorithm B we always choose a variable with the smallest frequency in either φ or ψ to reach the improved branching whenever possible. But we do not really compute the "threshold" frequency as there is no closed form for the function χ. Instead of computing $\varepsilon(\nu)$ via χ, we compute the frequencies of our splitting variable x_i and set $y_\varphi = 1/\varepsilon_i^\varphi$. When we now have to check whether $\varepsilon_i^\varphi \leq \varepsilon(\nu)$ we instead perform the equivalent check $y_\varphi^{y_\varphi} \geq \nu$ that can be implemented more easily. An analogous check is performed for ε_i^ψ.

6.2 Experimental results

To ensure comparability, we use test instances that were also used in previous studies [KBEG06, KS05]. We only slightly changed the known test bed in the sense that we added some additional instances not used before. Namely, we have so-called DTH instances that we derive by a role exchange from the TH instances.

Finally, we use a class of instances that are "hard" for several other MONET algorithms in the sense of theoretical lower bound analysis [Tak07, Hag07a] (cf. Chapter 5). The DNFs of our test instances are defined as follows (equivalent CNFs were previously computed by a brute force multiplication using the DL-algorithm [DL05] if necessary):

Matching (M(v)): v variables (v is even) $x_1, \ldots, x_v$ and monomials $(x_{i-1} \wedge x_i)$ for $2 \leq i \leq v$, i is even (in a graph this would form an induced matching). Hence, the DNF has $v/2$ monomials and the CNF has $2^{v/2}$ clauses.

Dual Matching (DM(v)): roles of DNF and CNF of the respective M(v) instance are exchanged. Hence, the CNF is very small.

Threshold (TH(v)): v variables (v is even) $x_1, \ldots, x_v$ and monomials $(x_i \wedge x_j)$ for $1 \leq i < j \leq v$, j is even. This yields $v^2/4$ monomials and $v/2 + 1$ clauses.

Dual Threshold (DTH(v)): roles of DNF and CNF of the respective TH(v) instance are exchanged.

Self-Dual Threshold (SDTH(v)): the monomials of SDTH(v) are obtained from the TH and DTH instances as follows: $\{\{x_{v-1}, x_v\}\} \cup \{\{x_{v-1}\} \cup m : m \in \mathrm{TH}(v-2)\} \cup \{\{x_v\} \cup m : m \in \mathrm{DTH}(v-2)\}$. The effect is that the equivalent CNF has the same set of terms. The number of terms is $(v-2)^2/4 + v/2 + 1$.

Self-Dual Fano-Plane (SDFP(v)): v variables and $(k-2)^2/4 + k/2 + 1$ monomials, where $k = (v-2)/7$. The construction starts with the DNF φ_0 that contains the monomials $\{x_1, x_2, x_3\}$, $\{x_1, x_5, x_6\}$, $\{x_1, x_7, x_4\}$, $\{x_2, x_4, x_5\}$, $\{x_2, x_6, x_7\}$, $\{x_3, x_4, x_6\}$, and $\{x_3, x_5, x_7\}$, representing the set of lines in a Fano plane. Now let $\varphi = \varphi_1 \vee \cdots \vee \varphi_k$, where $\varphi_1, \ldots, \varphi_k$ are k disjoint copies of φ_0. Denote by ψ the equivalent CNF of φ; its 7^k clauses are obtained by taking one monomial from each of the k copies of φ_0. We obtain the monomial set of SDFP(v) as $\{\{x_{v-1}, x_v\}\} \cup \{\{x_{v-1}\} \cup m : m \in \varphi\} \cup \{\{x_v\} \cup c : c \in \psi\}$.

Takata: these DNFs are "hard" for several MONET algorithms and are used in proving lower bounds [Tak07, Hag07a] (cf. Chapter 5). The starting point is $\varphi_1 = x_1$. The DNF φ_i is then obtained by multiplying out $\varphi_i = (\alpha \vee \beta) \wedge (\gamma \vee \delta)$, where α, β, γ, and δ are disjoint copies of φ_{i-1}. This gives $2^{2(2^i-1)}$ monomials in the DNF and 2^{2^i-1} clauses in the equivalent CNF.

The experimentation was done on an AMD Athlon 64 3700+ with 2.2 GHz and 1GB of RAM running a Debian/GNU Linux 4.0 with kernel version 2.6.18. As for compiling and interpreting the bytecode we used the JDK and JRE by Sun in version 1.5.0.10 in the 64 bit variant. Table 6.1 summarizes our experimental results on equivalent input instances. In the table, we show the total CPU time, in seconds. Times are normalized over five runs for each instance. Figures 6.1

M		$v = 20$	$v = 24$	$v = 28$	$v = 30$	$v = 32$	$v = 34$	$v = 36$	$v = 38$	$v = 40$
	FK-A	0.94	2.25	14.14	45.28	108.58	264.68	677.69	1888.78	5396.90
	FK-B	0.54	1.36	6.25	19.94	68.43	245.36	944.16	3538.28	15107.78
DM		$v = 20$	$v = 24$	$v = 28$	$v = 30$	$v = 32$	$v = 34$	$v = 36$	$v = 38$	$v = 40$
	FK-A	0.88	3.78	19.36	45.10	107.20	267.32	684.25	1970.00	5962.20
	FK-B	0.53	1.45	6.46	19.67	69.70	256.71	905.17	3575.83	14130.80
TH		$v = 40$	$v = 60$	$v = 80$	$v = 100$	$v = 120$	$v = 140$	$v = 160$	$v = 180$	$v = 200$
	FK-A	0.68	1.73	1.47	1.93	2.89	4.52	6.85	10.64	16.79
	FK-B	0.04	0.30	0.51	0.61	0.94	0.90	1.19	1.71	2.00
DTH		$v = 40$	$v = 60$	$v = 80$	$v = 100$	$v = 120$	$v = 140$	$v = 160$	$v = 180$	$v = 200$
	FK-A	0.48	1.29	1.48	2.09	3.06	4.66	6.67	10.23	13.92
	FK-B	0.04	0.30	0.48	0.65	0.96	0.93	1.47	1.56	2.04
SDTH		$v = 42$	$v = 62$	$v = 82$	$v = 102$	$v = 122$	$v = 142$	$v = 162$	$v = 182$	$v = 202$
	FK-A	1.28	1.39	1.72	3.06	5.13	9.24	16.84	27.08	41.33
	FK-B	0.30	0.51	0.83	1.16	1.41	2.49	4.81	7.35	10.81
SDFP		$v = 16$		$v = 23$		$v = 30$		$v = 37$		
	FK-A	0.47		2.42		15.40		801.17		
	FK-B	0.10		1.44		6.52		113.68		

Table 6.1: Performance of the FK-algorithms. Running time in seconds.

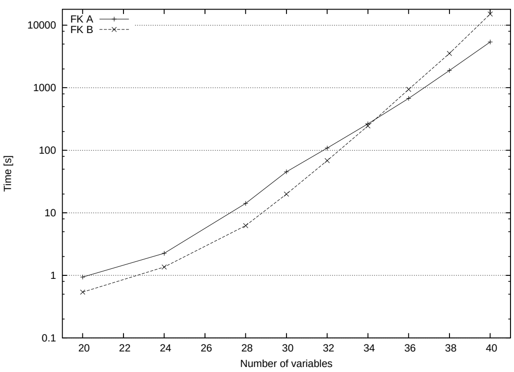

Figure 6.1: Running times on M(v)

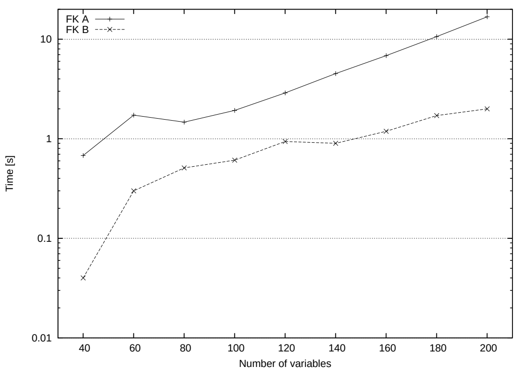

Figure 6.2: Running times on TH(v)

to 6.4 graphically show our results on several of the test instance classes. Note that the time axes are scaled logarithmically.

In Table 6.1 there are no results for the Takata instances or for non-equivalent inputs. As for the Takata instances, the reason is their exponential growing term set. The Takata DNFs φ_1 or φ_2 and their equivalent CNFs are just too small to give meaningful running times. Furthermore, storing φ_4 would require more than 40GB in a usual data format so that we decided to only test φ_3 and its equivalent CNF. This instance is solved by FK-algorithm B in 753.62 seconds whereas FK-algorithm A did not finish within 5 hours (18,000 seconds).

As for non-equivalent inputs, we tested both FK-algorithms on non-equivalent inputs that we consider to be "hardest". Namely, leaving out just one term of the DNF or CNF results in "nearly" equivalent instances. Consequently, the running times of our implementations are then just a little faster than the ones we report for the respective equivalent inputs. Not surprisingly, leaving out more terms or using some completely different CNFs speeds up computation as then larger and larger parts of recursion tree are not traversed. Hence, the running times in Table 6.1 are somehow the "worst" for the respective instance classes with the FK-algorithms traversing the whole recursion tree.

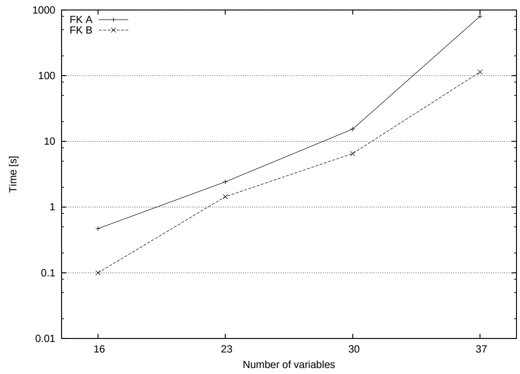

Figure 6.3: Running times on SDFP(v)

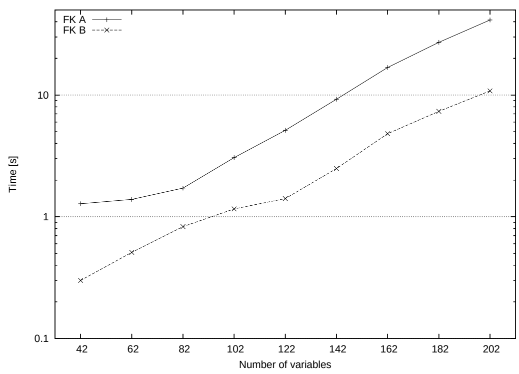

Figure 6.4: Running times on SDTH(v)

6.3 Conclusion

Comparing our results for FK-algorithm A and FK-algorithm B, we can conclude that FK-algorithm B is competitive on all classes, except for large Matching or Dual Matching instances. What exactly happens on these instances is an interesting issue to be addressed in future research. Utz-Uwe Haus mentioned that one reason might be internal sorting of the terms [Hau08]. Anyway, in contrast to the folklore assumption, our experiments show that FK-algorithm B should not *a priori* be excluded from experimental studies of Monet algorithms any more.

As for the computational variants of the FK-algorithms—given the DNF, compute the CNF—the relative behavior stays the same. But the running time increases dramatically and, compared to a Java implementation of the DL-algorithm, our implementations of the computational variants of the FK-algorithms are rather slow.

A future task that should be addressed in algorithm engineering for Monet is the creation of a comprehensive, systematic experimental evaluation of all the known approaches for the computational variant of Monet on a broad range of instances. We started working towards such a study, but it is still work in progress—e. g., many algorithms pending to be implemented on our platform. Hence, only the results for the FK-algorithms, that are of interest independently of other results, have been included in this thesis.

Other interesting future tasks for the FK-algorithms include the improvement of our implementations of the computational variants and a practical examination of the application of some more sophisticated recursion stopping rules that may cause an earlier interruption of the branching process.

Chapter 7

Fixed-Parameter Tractability

In this chapter we analyze the parameterized complexity of MONET. This means that we analyze versions of MONET that have some parameters as input in addition to the DNF φ and the CNF ψ. Briefly, a parameterized problem with parameter k is *fixed-parameter tractable* if it can be solved by some algorithm running in time $O(f(k) \cdot \mathrm{poly}(n))$, where f is a function depending on k only, n is the size of the input, and $\mathrm{poly}(n)$ is any polynomial in n. The class FPT contains all fixed-parameter tractable problems. For a more general survey on fixed-parameter tractability and parameterized complexity we refer to the monographs of Downey and Fellows [DF99], and Niedermeier [Nie06].

We show that MONET is in FPT for the parameters number v of variables in φ and ψ, number m of monomials in φ, a parameter q describing the variable frequencies in φ, and a parameter bounding the unions of transversals or edges of φ's associated hypergraph. Note that some of the results of [EGM03, Mak03] can be also interpreted as parameterized results, e. g., for the number of monomials as parameter. Below we shall improve these running times.

The chapter is organized as follows. In Sections 7.1 to 7.3 we give the FPT results for the parameters v, m, and q. Section 7.4 then is devoted to results based on using the Apriori technique (similar to the HBC-algorithm discussed in Section 5.2) whose outcome is an FPT result for the computational variant of MONET in its transversal hypergraph formulation, where parameters bound unions of edges or transversals. Some concluding remarks follow in Section 7.5.

7.1 Number of variables as parameter

A first super-naïve fixed-parameter tractability result for the number v of variables is at hand by simply checking all of the possible 2^v assignments for an instance (φ, ψ) of size n. This yields an $O(2^v \cdot n)$ time algorithm for MONET. To considerably improve this time bound, we use the notion of the maximum latency introduced by Makino and Ibaraki [MI97].

For a monotone formula α we denote by $T(\alpha)$ (resp. $F(\alpha)$) the set of assignments that satisfy α (do not satisfy α). We say that a MONET-instance (φ, ψ) is *well-formed* if φ and ψ are not empty but $T(\varphi) \cap F(\psi)$ is. Testing whether a given MONET-instance is well-formed can be accomplished in polynomial time.

The first condition is obviously trivial and the second is equivalent to testing the validity of $\varphi \to \psi$, which is an easy quadratic time procedure [DG96].

Definition 7.1.1 (maximum latency)**.** *Let* (φ, ψ) *be a well-formed* MONET*-instance. By* U *we denote the set of assignments that are neither in* $T(\varphi)$ *nor in* $F(\psi)$*, i.e.,* $U = F(\varphi) \cap T(\psi)$*. The* latency *of* (φ, ψ) *is defined as*

$$\lambda(\varphi, \psi) = \min\{|\mathcal{A}_U \bigtriangleup t| : \mathcal{A}_U \in U, t \text{ is a term of } \varphi \text{ or } \psi\},$$

where $\bigtriangleup$ *denotes the symmetric difference. For well-formed* MONET*-instances with* v *variables the* maximum latency *is defined as*

$$\Lambda(v) = \max\{\, \lambda(\varphi, \psi) \,:\, \varphi \text{ and } \psi \text{ have } v \text{ variables}\}.$$

Makino and Ibaraki proved the following tight bound on the maximum latency.

Proposition 7.1.2 ([MI97])**.** $\Lambda(v) = \lfloor v/4 \rfloor + 1$.

Finding an assignment $\mathcal{A}_U \in U$ that does not satisfy φ but satisfies ψ is equivalent to prove $(\varphi, \psi) \notin$ MONET. Hence, with Proposition 7.1.2 a well-formed MONET-instance (φ, ψ) can be tested for equivalence by checking all the assignments that differ in at most $\lfloor v/4 \rfloor + 1$ variables from any term of φ and ψ. We will use this idea in an algorithm that has a better running time than the first super-naïve approach. For the analysis we will need the following combinatorial observation.

Lemma 7.1.3. *Let* $0 < \varepsilon < \frac{1}{2}$*. Then we have*

$$\sum_{i=1}^{\lfloor \varepsilon k \rfloor} \binom{k}{i} \in O\left(\left[\left(\frac{1}{\varepsilon}\right)^{\varepsilon} \cdot \left(\frac{1}{1-\varepsilon}\right)^{1-\varepsilon}\right]^{k} \cdot \frac{1}{\sqrt{k}}\right).$$

Proof. It is well-known (see, e. g., [GKP94]) that asymptotically

$$\sum_{i=0}^{\lfloor \varepsilon k \rfloor} \binom{k}{i} = 2^{k \cdot h(\varepsilon) - \frac{1}{2}\log k + O(1)},$$

where $h(\varepsilon) = -\varepsilon \log \varepsilon - (1-\varepsilon)\log(1-\varepsilon)$ is the entropy function. Expanding the asymptotic equation yields the lemma. $\square$

Theorem 7.1.4. *Let* (φ, ψ) *be a* MONET*-instance of size* n *having* v *variables. Then* $(\varphi, \psi) \in$ MONET *can be decided in time*

$$O\left(\left[\left(\frac{1}{\frac{1}{4}+\frac{1}{v}}\right)^{\frac{1}{4}+\frac{1}{v}} \left(\frac{1}{\frac{3}{4}-\frac{1}{v}}\right)^{\frac{3}{4}-\frac{1}{v}}\right]^{v} \cdot \frac{1}{\sqrt{v}} \cdot n^2\right).$$

v	running time
≥ 5	$O(1.991^v \cdot \frac{1}{\sqrt{v}} \cdot n^2)$
≥ 10	$O(1.911^v \cdot \frac{1}{\sqrt{v}} \cdot n^2)$
≥ 20	$O(1.843^v \cdot \frac{1}{\sqrt{v}} \cdot n^2)$
≥ 50	$O(1.792^v \cdot \frac{1}{\sqrt{v}} \cdot n^2)$
≥ 100	$O(1.774^v \cdot \frac{1}{\sqrt{v}} \cdot n^2)$
≥ 1000	$O(1.757^v \cdot \frac{1}{\sqrt{v}} \cdot n^2)$

Table 7.1: Running time from Theorem 7.1.4 for special values of v

Proof. If the instance is not well-formed, it is rejected in quadratic time. Otherwise, for $v \leq 4$, we check all the at most 16 assignments in a brute-force manner in constant time.

For $v \geq 5$, we check all assignments that differ from any term of φ or ψ in at most $\Lambda(v)$ variables. This means that we check the $(\lfloor v/4 \rfloor + 1)$-neighborhoods of the terms of φ and ψ, which suffices as follows from Proposition 7.1.2. For each such assignment $\mathcal{A}$ we test in $O(n)$ time whether φ and ψ get the same value. There are at most n terms in φ and ψ. Hence, the running time of an algorithm checking all the necessary assignments can be bounded by $O(s \cdot n^2)$, where s denotes the number of assignments in a $\Lambda(v)$-neighborhood. We have $s = \sum_{i=1}^{\lfloor \varepsilon v \rfloor} \binom{v}{i}$, where $\varepsilon = \frac{1}{4} + \frac{1}{v}$. For $v \geq 5$ we have $\varepsilon < \frac{1}{2}$. Hence, the estimation of Lemma 7.1.3 can be applied and the theorem follows. $\square$

Table 7.1 contains the running time stated in Theorem 7.1.4 for special v in a more readable format. Note that an estimation for $v \to \infty$ yields a lower bound of

$$\Omega(1.7547^v \cdot \frac{1}{\sqrt{v}} \cdot n^2).$$

7.2 Number of monomials as parameter

We show that Monet is fixed-parameter tractable with the number m of monomials in φ as parameter.

Theorem 7.2.1. *Let (φ, ψ) be a* Monet*-instance of size n with m monomials in φ. Then $(\varphi, \psi) \in$* Monet *can be decided in time $O(2^{m \cdot (m - \log m + 4)} \cdot m^3 + n^2)$.*

Proof. Note that m monomials can split the set of variables into at most 2^m classes of variables that appear in exactly the same monomials (actually there are at most $2^m - 1$ classes but this would only complicate the below estimations). Choosing representatives for each class and replacing variables with these representatives yields a modified DNF φ' with m monomials and at most 2^m variables. Hence, the irredundant, equivalent CNF ψ' of φ' cannot have clauses that contain more than m variables. We compute ψ' by adapting the KS-algorithm of Kavvadias and Stavropoulos [KS05] (cf. Section 5.1.4) for hypergraph transversal generation. The main idea of the KS-algorithm is to process a depth-first search in a search tree that is built as follows. The root of the tree corresponds to a monomial of φ'. If the subset of variables on the path from the root to the current node (this subset forms a clause candidate) does not intersect all monomials of φ', the KS-algorithm expands it by picking a monomial that is not yet intersected and generating edges for each so-called *appropriate* variable in this monomial. Briefly, a variable is appropriate if adding it to the current candidate set does not result in a set where another variable could be left out and still all monomials except the last one are intersected. Checking a monomial for appropriate vertices can be done in time $O((2^m \cdot m)^2)$ since a monomial contains at most 2^m variables, the current clause candidate set has size at most m, and the size of φ' is bounded by $2^m \cdot m$. Expanding only by appropriate variables ensures that generated clauses are minimal and that no repetitions occur [KS05].

If all monomials are covered, the KS-algorithm outputs the clause and starts backtracking. In the worst case, the search tree that is traversed by the KS-algorithm contains a node corresponding to each variable subset of size at most m (written on the paths from the root to the nodes). There are $\sum_{i=0}^{m} \binom{2^m}{i} \leq m \cdot \binom{2^m}{m}$ such subsets. Using Stirling's formula and the fact that for the Eulerian constant we have $e \approx 2.718 < 4$ we get $\binom{2^m}{m} \leq 2^{m\cdot(m-\log m+2)}$. Since in the worst case for each node the appropriate variables have to be determined, the KS-algorithm needs $O(2^{m\cdot(m-\log m+4)} \cdot m^3)$ time to compute the CNF ψ'.

From ψ' we compute a CNF ψ'' without representatives. This is done by systematically processing the representatives one after the other. Let y be the currently processed representative that stands for the variables $x_{i_1}, \ldots, x_{i_k}$. For each occurrence of y the respective clause is copied k times and in the j-th copy we replace y by x_{i_j}. Since ψ' is irredundant, all the intermediate results of the computation and ψ'' are irredundant. Thus, we can immediately reject whenever an intermediate result gets larger than ψ. Hence, the time needed to compute ψ'' is $O(n)$ as n is an upper bound on the size of ψ. When there is no representative left we have to check whether ψ'' and ψ are identical. This can be accomplished in time $O(n^2)$. $\square$

Note that we have the same result with the number of clauses of ψ as parameter since we could simply exchange the roles of DNF and CNF.

The running time has been subsequently improved in the transversal hypergraph formulation of Monet. The essential part of the improvement is a fixed-parameter sub-transversal check. A subset S of vertices of a hypergraph $\mathcal{H}$ is a *sub-transversal* of $\mathcal{H}$ iff there is a minimal transversal $T \in \mathrm{Tr}(\mathcal{H})$ such that $T \supseteq S$. In general, testing if S is a sub-transversal is an NP-hard problem even if $\mathcal{H}$ is a graph [BEGK00]. But in the case that $|S|$ is bounded by a constant such a check can be done in polynomial time. Note that we just cite the following results as the main work was done by Khaled Elbassioni and Imran Rauf.

Lemma 7.2.2 ([EHR08b]). *Given a hypergraph $\mathcal{H}$ with m edges and a vertex subset S of size $|S| = s$, checking whether S is a sub-transversal of $\mathcal{H}$ can be done in time $O((m/s)^s \cdot nm)$.*

Using Lemma 7.2.2 in a backtracking algorithm that computes all the minimal transversals starting from the empty set (that clearly is a sub-transversal) yields the following.

Theorem 7.2.3 ([EHR08b]). *Let $\mathcal{G}, \mathcal{H}$ be two hypergraphs with $|\mathcal{G}| = m$, $|\mathcal{H}| = m'$, and $|V| = n$. Then $\mathrm{Tr}(\mathcal{H}) = \mathcal{G}$ can be decided in time $O(e^{(m/e)} \cdot n^2m^2m')$.*

Note that $e^{(m/e)} \approx 1.45^m$ and, thus, this improves the running time of Theorem 7.2.1. An implication of Theorem 7.2.3, which we will need in Section 7.4, is the following.

Corollary 7.2.4. *For a hypergraph $\mathcal{H}$, we can generate the first k minimal transversals in time $O(e^{(k/e)} \cdot k^3n^2m)$, where $n = |V|$ and $m = |\mathcal{H}|$.*

Proof. The idea is to keep a partial list $\mathcal{G}$ of minimal transversals, initially empty. If $|\mathcal{G}| < k$, we use the backtracking method from Theorem 7.2.3 on $\mathcal{G}$ to generate at most $m + 1$ elements of $\mathrm{Tr}(\mathcal{G})$. If it terminates with $\mathrm{Tr}(\mathcal{G}) = \mathcal{H}$, then all elements of $\mathrm{Tr}(\mathcal{H})$ have been generated. Otherwise, $X \in \mathrm{Tr}(\mathcal{G}) \setminus \mathcal{H}$ is a witness for $(\mathcal{G}, \mathcal{H}) \notin$ TransHyp, and so by symmetry $\bar{X}$ contains a new minimal transversal of $\mathcal{H}$ that extends $\mathcal{G}$. □

7.3 Variable degrees as parameter

For a Monet instance (φ, ψ) we denote by q the largest number of monomials over all variables x that do not include x, i.e.,

$$q = \max_{x \in V} |\{\mu : x \notin \mu, \text{ where } \mu \text{ is a monomial of } \varphi\}|.$$

We show that Monet is fixed-parameter tractable with q as parameter.

Theorem 7.3.1. *Let (φ, ψ) be a Monet-instance of size n and q as defined above. Then $(\varphi, \psi) \in$ Monet can be decided in time $O(2^{q\cdot(q-\log q+4)} \cdot q^3n + n^3)$.*

Proof. For an irredundant, monotone normal form α we denote by $\alpha^{x=0}$ (resp. $\alpha^{x=1}$) the normal form that is obtained by setting x to `false` (`true`) and removing redundant terms. These irredundant forms $\alpha^{x=0}$ and $\alpha^{x=1}$ can be easily computed in quadratic time.

Note that testing $(\varphi, \psi) \in$ Monet is equivalent to testing $(\varphi^{x=0}, \psi^{x=0}) \in$ Monet and $(\varphi^{x=1}, \psi^{x=1}) \in$ Monet for any variable x from φ. Our fixed-parameter algorithm exactly processes these tests. Note that setting $x = 0$ yields a DNF $\varphi^{x=0}$ with at most q monomials. Hence, we can apply Theorem 7.2.1 and decide $(\varphi^{x=0}, \psi^{x=0}) \in$ Monet in time $O(2^{q\cdot(q-\log q+4)} \cdot q^3 + n^2)$.

For the second test we recursively call the algorithm with $(\varphi^{x=1}, \psi^{x=1})$ as input. Note that $\varphi^{x=1}$ contains at most $n-1$ variables. If $\varphi^{x=1}$ does not contain any variable, the equivalence test is trivial. Hence, there are at most $n-1$ recursive calls which results in an overall running time of $O(2^{q\cdot(q-\log q+4)} \cdot q^3 n + n^3)$. $\square$

Note that again the roles of DNF and CNF may be exchanged to get the statement for variable frequencies in ψ as well. The algorithm runs in polynomial time if q is a constant. This yields a new polynomial time special case of Monet.

Using the running time of Theorem 7.2.3, Theorem 7.3.1 has also been subsequently improved in the transversal hypergraph formulation of Monet (we just cite the following results as the main work was done by Khaled Elbassioni and Imran Rauf).

Theorem 7.3.2 ([EHR08b]). *Let $\mathcal{G}, \mathcal{H}$ be two hypergraphs with $|\mathcal{G}| = m, |\mathcal{H}| = m'$, $q = \max_v |\{H \in \mathcal{H} : v \notin H\}|$, and $|V| = n$. Then $\mathrm{Tr}(\mathcal{H}) = \mathcal{G}$ can be decided in time $O(e^{(q/e)} \cdot q^2 n^3 m')$.*

Again, note that $e^{(q/e)} \approx 1.45^q$. Moreover, note that q can be seen as the complementary degree of a vertex in a hypergraph. Applying Theorem 7.2.3, there is also an FPT result for the *maximum degree* of a hypergraph. Let p be the maximum degree of a vertex in hypergraph $\mathcal{H}$, i. e., $p = \max_v |\{H \in \mathcal{H} : v \in H\}|$.

Theorem 7.3.3 ([EHR08b]). *Let $\mathcal{H}$ be a hypergraph on $|V| = n$ vertices in which the degree of each vertex v is bounded by p. Then all minimal transversals of $\mathcal{H}$ can be found in time $O\left(\left(\min\{2^p, m'\} \cdot e^{p/e} \cdot p^2 n + m'\right) \cdot n^2 m m'\right)$, where $m = |\mathcal{H}|$ and $m' = |\mathrm{Tr}(\mathcal{H})|$.*

The key idea in the proof of Theorem 7.3.3 is to order the vertices and at step i only compute transversals of the edges containing v_i and, apart from it, only vertices with smaller index. The number of such edges clearly is bounded by p and, thus, Theorem 7.2.3 can be applied.

7.4 Results based on the Apriori technique

Gunopulos et al. [GKM+03] showed that generating transversals of hypergraphs $\mathcal{H}$ with edges of size at least $n-c$ can be done in time $O(2^c \cdot \mathrm{poly}(n, m, m'))$, where $n = |V|$, $m = |\mathcal{H}|$ and $m' = |\mathrm{Tr}(\mathcal{H})|$. This is a fixed-parameter algorithm for c as parameter, which shows that the transversals can be generated in polynomial time for $c \in O(\log n)$. The computation is done by an Apriori (level-wise) algorithm [AS94]. Using the same approach, we generalize the result and show below that we can compute all the minimal transversals in time $O(2^c \cdot (m')^k \cdot \mathrm{poly}(n, m))$ or in time $O(e^{k/e} \cdot n^{c+1} \cdot \mathrm{poly}(m, m'))$ if the *union* of any k distinct minimal transversals has size at least $n-c$. Equivalently, if any k distinct maximal independent sets of a hypergraph $\mathcal{H}$ intersect in at most c vertices, then all maximal independent sets can be computed within the same time bound. Recall that an *independent set* of a hypergraph $\mathcal{H}$ is a subset of its vertices which does not contain any hyperedge of $\mathcal{H}$.

And again using the same idea, we show that the maximal frequent sets of an $m \times n$ database can be computed in $O(2^c \cdot (nm')^{2^{k-1}+1} \cdot \text{poly}(n,m))$ time if any k rows of it intersect in at most c items, where m' is the number of such maximal frequent sets.

Note that for $c \in O(\log n)$ we have output-polynomial algorithms for all four problems.

7.4.1 The generalized Apriori algorithm

Let $f : V \to \{0,1\}$ be a monotone Boolean function, that is, for which $f(X) \geq f(Y)$ whenever $X \supseteq Y$. We assume that f is given by a polynomial-time evaluation oracle requiring maximum time T_f, given the input. The Apriori approach for finding all maximal subsets X such that $f(X) = 0$ (maximal false sets of f), works by traversing all subsets X of V, for which $f(X) = 0$, in increasing size, until all maximal such sets have been identified. The procedure is given as Algorithm 24.

Lemma 7.4.1. *If any maximal false set of f contains at most c vertices, then Apriori given as Algorithm 24 finds all such sets in $O(2^c \cdot m'n \cdot T_f)$ time, where $n = |V|$ and m' is the number of maximal false sets.*

Proof. The correctness of this Apriori style method can be shown straightforwardly (cf. , e. g., [AS94, GKM+03]). To see the time bound, note that for each maximal false set we check at most 2^c candidates (all the subsets) before adding it to $\mathcal{C}$. For each such candidate we check whether it is a false set and whether it cannot be extended by adding more vertices. □

7.4.2 Intersections of maximal independent sets

Our results in this section are based on using an Apriori approach as described in the previous section. We start with an easy result very similar to the algorithm described in [GKM+03] for generating minimal transversals. The following lemma can be seen as a proof of concept in using the Apriori technique.

Lemma 7.4.2. *If any maximal independent set of a hypergraph $\mathcal{H}$ with n vertices contains at most k vertices, all the m' maximal independent sets can be computed in time $O(2^k \cdot \text{poly}(n,m,m'))$, where $|\mathcal{H}| = m$.*

Proof. An appropriate procedure is given as Algorithm 25. Correctness is straightforward as is for the generalized Apriori.

The time needed by Algorithm 25 can be bound as follows. Note that for each maximal independent set we check at most 2^k candidates (all the subsets) before adding it to $\mathcal{I}$. For each such candidate we check whether it is independent (clearly polynomial in m) and whether we cannot add another vertex and still

Algorithm 24 The generalized Apriori algorithm

Input: a monotone Boolean function $f : V \to \{0,1\}$
Output: the maximal sets $X \subseteq V$ such that $f(X) = 0$

$C_1 \leftarrow V$
$i \leftarrow 1$
while $C_i \neq \emptyset$ **do**
  for all $X, Y \in C_i, |X \cap Y| = i - 1$ **do**
    $Z \leftarrow X \cup Y$
    if $f(Z) = 0$ **then**
      if $f(Z \cup \{v\}) = 1$, for all $v \in V \setminus Z$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{Z\}$
      else
        $C_{i+1} \leftarrow C_{i+1} \cup \{Z\}$
  $i \leftarrow i + 1$
output $\mathcal{C}$

have independence (also polynomial, as there are at most n vertices that can be added). Thus, the overall-running time is $O(2^k \cdot \mathrm{poly}(n, m, m'))$. $\square$

We now want to get rid of the size bound on the independent sets and replace it by a bound on the size of their intersections.

Let k and c be two positive integers. We consider hypergraphs $\mathcal{H}$ satisfying the following condition:

(C1) Any k distinct maximal independent sets $I_1, \ldots, I_k$ of $\mathcal{H}$ intersect in at most c vertices, i.e., $|I_1 \cap \cdots \cap I_k| \leq c$.

We shall derive below fixed-parameter algorithms with respect to either c or k. We note that condition (C1) can be checked in polynomial time for $c = O(1)$ and $k = O(\log n)$. Indeed, (C1) holds if and only if every set $X \subseteq V$ of size $|X| = c+1$ is contained in at most $k-1$ maximal independent sets of $\mathcal{H}$. The latter condition can be checked in time $O(e^{k/e} \cdot n^{c+1} \cdot \mathrm{poly}(n, m, k))$ as follows from the following lemma.

Lemma 7.4.3. *Given a hypergraph $\mathcal{H}$ with vertex set V and a subset $S \subseteq V$ of vertices, we can check in polynomial time whether S is contained in k different maximal independent sets. Furthermore, k such sets can be generated in time $O(e^{k/e} \cdot \mathrm{poly}(n, m, k))$.*

Proof. Clearly, this check is equivalent to checking if S does not contain an edge of $\mathcal{H}$ and if the truncated hypergraph $\mathcal{H}^{\bar{S}} = \min\left(\{H \cap \bar{S} : H \in \mathcal{H}\}\right)$, has k maximal independent sets, or equivalently k minimal transversals. By Corollary 7.2.4, this can be done in $O(e^{k/e} \cdot \mathrm{poly}(n, m, k))$ time. $\square$

Algorithm 25 Apriori for finding maximal independent sets of size at most k

Input: a hypergraph $\mathcal{H}$ on vertex set V

$C_1 \leftarrow V \setminus \{v : \{v\} \in \mathcal{H}\}$
$i \leftarrow 1$
while $C_i \neq \emptyset$ and $i \leq k$ **do**
  for all $a, b \in C_i, |a \cap b| = i - 1$ **do**
    $c \leftarrow a \cup b$
    if c is an independent set **then**
      if no $c \cup \{v\}, v \in V \setminus c$ is an independent set **then**
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{c\}$
      else
        $C_{i+1} \leftarrow C_{i+1} \cup \{c\}$
  $i \leftarrow i + 1$
output $\mathcal{I}$

Algorithm 26 Algorithm generating intersecting maximal independent sets

Input: a hypergraph $\mathcal{H}$ on vertex set V
Output: the set of maximal independent sets of $\mathcal{H}$

$\mathcal{I} \leftarrow \emptyset$
use Apriori to find set of maximal k-independent set intersections $\mathcal{X}$
for all $X \in \mathcal{X}$ **do**
  for all $Y \subseteq X$ **do**
    for all $v \in V \setminus Y$ **do**
      if $|\mathrm{Ind}(\mathcal{H})[Y \cup \{v\}]| \leq k - 1$ **then**
        $\mathcal{I} \leftarrow \mathcal{I} \cup \mathrm{Ind}(\mathcal{H})[Y \cup \{v\}]$ (obtained using Corollary 7.2.4)
output $\mathcal{I}$

Denote by $\mathrm{Ind}(\mathcal{H})$ the set of maximal independent sets of a hypergraph $\mathcal{H}$ and for a subset S of $\mathcal{H}$'s vertices, denote by $\mathrm{Ind}(\mathcal{H})[S]$ the maximal independent sets containing S.

Theorem 7.4.4. *If any k distinct maximal independent sets of a hypergraph $\mathcal{H}$ intersect in at most c vertices, then all maximal independent sets can be computed in time $O(2^c \cdot (m')^k \cdot \mathrm{poly}(n, m))$ or in time $O(e^{k/e} \cdot n^{c+1} \cdot \mathrm{poly}(m, m'))$, where $n = |V|$, $m = |\mathcal{H}|$ and $m' = |\mathrm{Ind}(\mathcal{H})|$.*

Proof. (i) c as a parameter: we first use Apriori to find the set $\mathcal{X}$ of all maximal subsets contained in at least k distinct maximal independent sets of $\mathcal{H}$. By (C1) the size of each such subset is at most c. To do this we use Apriori with the monotone Boolean function defined by $f(X) = 0$ if and only if $X \subseteq I_1 \cap \cdots \cap I_k$, for k distinct maximal independent sets $I_1, \ldots, I_k$. The procedure is given as Algorithm 26. By Lemmas 7.4.1 and 7.4.3, all the intersections in $\mathcal{X}$ can be found in time $2^c \cdot e^{k/e} \cdot |\mathcal{X}| \cdot \mathrm{poly}(n, m, k)$. Thus the total running time can be bounded by $2^c \cdot e^{k/e} \cdot (m')^k \cdot \mathrm{poly}(n,m,k)$ since $|\mathcal{X}| \le (m')^k$. It remains to argue that any maximal independent set $I \in \mathrm{Ind}(\mathcal{H})$ is generated by the procedure. To see this, let Y be a maximal subset such that $Y = I \cap I_1 \cap \cdots \cap I_r$, where $I, I_1, \ldots, I_r$, are distinct maximal independent sets of $\mathcal{H}$ with $r \ge k-1$, and let $v \in I \setminus (I_1 \cap \cdots \cap I_r)$. Note that such v exists since $I \not\subseteq I_1 \cap \cdots \cap I_r$ as $I, I_1, \ldots, I_r$ are distinct maximal independent sets. Then by maximality of Y, $Y \cup \{v\}$ is contained in at most $k-1$ maximal independent sets, one of which is I, and hence will be considered by the procedure in Step 7.

(ii) k as a parameter: Let $\mathcal{I}_1 = \{I \in \mathrm{Ind}(\mathcal{H}) : |I| \le c\}$ and $\mathcal{I}_2 = \mathrm{Ind}(\mathcal{H}) \setminus \mathcal{I}_1$. Elements of $\mathcal{I}_1$ can be found using our proof of concept Apriori procedure given as Algorithm 25 (or by testing all subsets of size at most c for maximal independence). Elements of $\mathcal{I}_2$ can be found by noting that each of them contains a set of size $c+1$, and that each such set is contained in at most $k-1$ elements of $\mathcal{I}_2$ by (C1). Thus, for each set X of size $c+1$, we can use Lemma 7.4.3 to find all maximal independent sets containing X. □

7.4.3 Unions of minimal transversals or edges

We now use the results on independent sets to generate all minimal transversals.

Theorem 7.4.5. *Let $\mathcal{H}$ be a hypergraph on $n = |V|$ vertices, and k, c be positive integers.*

(i) If any k distinct minimal transversals of $\mathcal{H}$ have a union of at least $n-c$ vertices, we can compute all minimal transversals in $O(2^c \cdot (m')^k \cdot \mathrm{poly}(n,m))$ or $O(e^{k/e} \cdot n^{c+1} \cdot \mathrm{poly}(m,m'))$ time, where $m = \mathcal{H}$ and $m' = |\mathrm{Tr}(\mathcal{H})|$.

(ii) If any k distinct hyperedges of $\mathcal{H}$ have a union of at least $n-c$ vertices, we can compute all minimal transversals in time $O(2^c \cdot m^k \cdot \mathrm{poly}(n,m'))$ or in time $O(e^{k/e} \cdot n^{c+1} \cdot \mathrm{poly}(m,m'))$, where $m = \mathcal{H}$ and $m' = |\mathrm{Tr}(\mathcal{H})|$.

Proof. Both results are immediate from Theorem 7.4.4. The first part (i) follows by noting that each minimal transversal is the complement of a maximal independent set, and hence any k maximal independent sets are guaranteed to intersect in at most c vertices. The second part (ii) follows by maintaining a partial list $\mathcal{G} \subseteq \mathrm{Tr}(\mathcal{H})$, and switching the roles of $\mathcal{H}$ and $\mathcal{G}$ in (i) to compute the minimal transversals of $\mathcal{G}$ using Theorem 7.4.5. Since condition (i) is satisfied with respect to $\mathcal{G}$, we can either verify $\mathcal{H} = \mathrm{Tr}(\mathcal{G})$, or extend $\mathcal{G}$ by finding a witness for the non-transversality (in a way similar to Corollary 7.2.4). □

7.4.4 Intersections of maximal frequent sets or transactions

Consider the problem of finding the maximal frequent item sets in a collection of m transactions on n items, mentioned in Section 3.8. Here, a transaction simply is a set of items. An item set is *maximal frequent* for a frequency t if it occurs in at least t of the transactions and none of its supersets does. Some basic FPT results for this problem can be found in [HCW06].

As another application of the approach used in Theorem 7.4.4 to compute intersecting maximal independent sets we obtain the following results.

Theorem 7.4.6. *If any k distinct maximal frequent sets intersect in at most c items, we can compute all maximal frequent sets in $O(2^c \cdot (nm')^k \cdot \mathrm{poly}(n,m))$ time, where m' is the number of maximal frequent sets.*

Proof. The proof is analogous to that of Theorem 7.4.4. Just note that the set of transactions forms a hypergraph and replace "independent" by "frequent." To complete the proof, we need the following procedure to find k maximal frequent sets containing a given set. For $1 \le i \le k$ and frequent set X, let $F_1, \dots, F_{i-1}$ be the maximal frequent sets containing X and let Y be the set with the property that $X \cup Y$ is frequent and $\forall j < i, \exists y \in Y : y \notin F_j$. Then any maximal frequent set containing $X \cup Y$ is different from $F_1, \dots, F_{i-1}$ by construction and thus giving us a new maximal frequent set. The running time of the above procedure can be bounded by $O(n^k \cdot \mathrm{poly}(n,m))$. Combining it with Lemma 7.4.1 gives us the stated running time. □

Corollary 7.4.7. *If any k distinct transactions intersect in at most c items, then all maximal frequent sets can be computed in time $O(2^c \cdot (nm')^{2^{k-1}+1} \cdot poly(n,m))$, where m' is the number of maximal frequent sets.*

Proof. Note that if $t \ge k$ then every maximal frequent set has size at most c which in turn implies $O(2^c \cdot \mathrm{poly}(n,m) \cdot m')$ time algorithm using straightforward Apriori approach, so we may assume otherwise. Consider the intersection X of l distinct maximal frequent sets and let $|X| > c$, we bound the maximum such l. Since the intersection size is more then c, at most $k-1$ transactions define these l distinct maximal frequent sets and so $l \le \sum_{j=t}^{k-1} \binom{k-1}{j} \le 2^{k-1}$. □

7.5 Concluding remarks

There is at least one important open question besides improving the running times of our FPT algorithms. Giving an FPT algorithm for Monet with respect to the parameter size l of a largest monomial remains open. Note that there is an easy search tree FPT algorithm for the *two* parameters size l of a largest monomial and size k of a largest clause similar to a result in [GSS02].

However, assume Monet can be shown to be M[1]- or W[1]-hard for l. We have FPT $\subseteq$ M[1] $\subseteq$ W[1], and FPT $=$ M[1] iff the exponential time hypothesis fails, i. e., n variable 3Sat can be solved in time $2^{o(n)}$. Hence, M[1]- or W[1]-hardness of Monet corresponds to Monet not being FPT under a reasonable assumption. Thus, a $f(l) \cdot \mathrm{poly}(n)$ running time for Monet would be quite unlikely, which would be the very first argument against polynomiality of Monet. Hence, we expect showing M[1]- or W[1]-hardness for parameter l to be a tough problem.

Finally, our conjecture is that Monet is FPT for l but this might turn out to be equivalent to showing Monet $\in$ P.

Chapter 8

Conclusion

In this thesis, we have examined the problem MONET—the MO(notone) N(ormal form) E(quivalence) T(est)—that asks to decide equivalence of a monotone disjunctive normal form φ and a monotone conjunctive normal form ψ. MONET can be seen as a covering problem that asks to enumerate all (in some sense) minimal solutions of some system. Hence, there is a huge number of similar questions in problems from diverse applications. In this thesis we examined several issues related to the problem MONET.

We first started by focussing on identification of restrictions of the inputs that sufficiently simplify the problem in the sense of polynomial time solutions. Such restrictions are the "easy" classes of MONET. For several of these easy classes we have shown that they actually are solvable using only logarithmic space, which improves the previously known polynomial time bounds. Among our results are $\text{MONET}_{\text{cnm}}$, where the DNF is allowed to contain only a constant number of monomials; $\text{MONET}_{\text{cupp}}$, where the DNF is allowed to contain only monomials of constant size; and $\text{MONET}_{\text{clow}}$, where each monomial of the DNF is only allowed not to contain a constant number of variables. As for the more structural restrictions, we have shown that MONET with a regular, a 2-monotonic, or an aligned DNF is decidable in logarithmic space.

An open issue is to show more easy classes to be solvable with logarithmic space algorithms. Our conjecture is that at least instances with β-acyclic or μ-equivalent DNFs are also solvable in logarithmic space. Another issue is to prove lower bounds for easy classes. Such lower bounds for special classes could be useful when proving hardness of MONET. Note that we do not have any nontrivial lower bound or hardness result for MONET yet.

In a second step, we have also analyzed algorithms for the general problem MONET without any restrictions. Thereby, we have proven superpolynomial lower bounds for the DL-, the BMR-, the KS-, and the HBC-algorithm in terms of the size of the input and output. This means that none of these algorithms is fast in the sense of output-polynomial running time.

An important open issue is to prove lower bounds for other MONET algorithms as well. Our conjecture is that none of the known algorithms is fast in the sense of (output-)polynomial time. However, transforming our results to other algorithms seems not to be that straightforward. For instance, Takata's multiplication method, which is an improvement of the Berge-multiplication, seems to require the construction of new hard instances, as the instances we used for establishing lower bounds—Takata's hypergraphs $\mathcal{G}_i$ and our hypergraphs $\mathcal{G}'_i$—are solved too fast. There are also no nontrivial lower bounds known for the FK-algorithms A and B, of which variant B is considered to be the (theoretically) fastest known MONET algorithm.

However, in practical settings the usual assumption was that FK-algorithm B is far more involved to implement than FK-algorithm A and that this effort will not pay off in practically better running times. Contrarywise, our experimental results with the FK-algorithms show FK-algorithm B to be competitive on almost all instances. Nevertheless, as for algorithm engineering of MONET algorithms, the big open question is a comprehensive, systematic experimental evaluation of all the known algorithms. We have just started working on this issue, so it is still work in progress—e. g., many algorithms pending to be implemented on our platform.

Finally, we have shown MONET to be fixed-parameter tractable with respect to various parameters. Namely, MONET is fixed-parameter tractable for the parameters number v of variables in φ and ψ, number m of monomials in φ, a parameter q describing the variable frequencies in φ, and a parameter bounding the unions of transversals or edges of φ's associated hypergraph. The important open problem for the parameterized complexity of MONET is giving an FPT algorithm with respect to the parameter size l of a largest monomial. Our conjecture is that MONET is FPT for l but this might turn out to be equivalent to showing MONET $\in \mathsf{P}$.

Bibliography

[AB00] Chandrabose Aravindan and Peter Baumgartner. Theorem proving techniques for view deletion in databases. *Journal of Symbolic Computation*, 29(2):119–147, 2000.

[AE06] Yossi Azar and Thomas Erlebach, editors. *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*. Springer, 2006.

[Afr06] Foto N. Afrati. On approximation algorithms for data mining applications. In Evripidis Bampis, Klaus Jansen, and Claire Kenyon, editors, *Efficient Approximation and Online Algorithms – Recent Progress on Classical Combinatorial Optimization Problems and New Applications*, volume 3484 of *Lecture Notes in Computer Science*, pages 1–29. Springer, 2006.

[AIP04] Fabrizio Angiulli, Giovambattista Ianni, and Luigi Palopoli. On the complexity of inducing categorical and quantitative association rules. *Theoretical Computer Science*, 314(1-2):217–249, 2004.

[AK03] Bernhard Anrig and Jürg Kohlas. Model-based reliability and diagnostics: A common framework for reliability and diagnostics. *International Journal of Intelligent Systems*, 18(10):1001–1033, 2003.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[AMS+96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15,* *1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[Bay98] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 85–93. ACM Press, 1998.

[BBE+07] Endre Boros, Konrad Borys, Khaled M. Elbassioni, Vladimir Gurvich, Kazuhisa Makino, and Gábor Rudolf. Generating minimal k-vertex connected spanning subgraphs. In Guohui Lin, editor, *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Banff, Canada, July 16-19, 2007, Proceedings*, volume 4598 of *Lecture Notes in Computer Science*, pages 222–231. Springer, 2007.

[BCE+00] Endre Boros, Yves Crama, Oya Ekin, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. Boolean normal forms, shellability, and reliability computations. *SIAM Journal on Discrete Mathematics*, 13(2):212–226, 2000.

[BCG+96] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996.

[BD96] Rachel Ben-Eliyahu and Rina Dechter. On computing minimal models. *Annals of Mathematics and Artificial Intelligence*, 18(1):3–27, 1996.

[BEG+02] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities. *SIAM Journal on Computing*, 31(5):1624–1643, 2002.

[BEG+08] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, Kazuhisa Makino, and Vladimir Oudalov. A complete characterization of Nash-solvability of bimatrix games in terms of the exclusion of certain 2×2 subgames. In Edward A. Hirsch, Alexander A. Razborov, Alexei L. Semenov, and Anatol Slissenko, editors, *Computer Science - Theory and Applications, Third International Computer Science Symposium in Russia, CSR 2008, Moscow, Russia, June 7-12, 2008, Proceedings*, volume 5010 of *Lecture Notes in Computer Science*, pages 99–109. Springer, 2008.

[BEGK00] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Processing Letters*, 10(4):253–266, 2000.

[BEGK02a] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Generating dual-bounded hypergraphs. *Optimization Methods and Software*, 17(5):749–781, 2002.

[BEGK02b] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Matroid intersections, polymatroid inequalities, and related problems. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 143–154. Springer, 2002.

[BEGK03a] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. An inequality for polymatroid functions and its applications. *Discrete Applied Mathematics*, 131(2):255–281, 2003.

[BEGK03b] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Extending the Balas-Yu bounds on the number of maximal independent sets in graphs to hypergraphs and lattices. *Mathematical Programming*, 98(1-3):355–368, 2003.

[BEGK04] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Computing many maximal independent sets for sparse hypergraphs in parallel. Technical Report RRR 38-2004, RUTCOR, Rutgers University, October 2004.

[BEGM07] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Generating vertices of polyhedra and related monotone generation problems. Technical Report RRR 12-2007, RUTCOR, Rutgers University, March 2007.

[BEM08] Endre Boros, Khaled M. Elbassioni, and Kazuhisa Makino. On Berge multiplication for monotone Boolean dualization. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 48–59. Springer, 2008.

[Ben99] Claude Benzaken. From logical gates synthesis to chromatic bicritical clutters. *Discrete Applied Mathematics*, 96-97:259–305, 1999.

[Ber89] Claude Berge. *Hypergraphs*, volume 45 of *North-Holland Mathematical Library*. North-Holland, 1989.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[BG07a] José L. Balcázar and Gemma C. Garriga. Horn axiomatizations for sequential data. *Theoretical Computer Science*, 371(3):247–264, 2007.

[BG07b] Endre Boros and Vladimir Gurvich. On complexity of algorithms for modeling disease transmission and optimal vaccination strategy. Technical Report RRR 16-2007, RUTCOR, Rutgers University, April 2007.

[BGH98] Endre Boros, Vladimir Gurvich, and Peter L. Hammer. Dual subimplicants of positive Boolean functions. *Optimization Methods and Software*, 10(2):147–156, 1998.

[BGKM01] Endre Boros, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. Dual-bounded generating problems: Partial and multiple transversals of a hypergraph. *SIAM Journal on Computing*, 30(6):2036–2050, 2001.

[BGKM03] Endre Boros, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. On maximal frequent and minimal infrequent sets in binary matrices. *Annals of Mathematics and Artificial Intelligence*, 39(3):211–221, 2003.

[BGKM04] Endre Boros, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. Dual-bounded generating problems: weighted transversals of a hypergraph. *Discrete Applied Mathematics*, 142(1-3):1–15, 2004.

[BGM07] Endre Boros, Vladimir Gurvich, and Kazuhisa Makino. Minimal and locally minimal games and game forms. Technical Report RRR 28-2007, RUTCOR, Rutgers University, November 2007.

[BHHM07] Eric Berberich, Matthias Hagen, Benjamin Hiller, and Hannes Moser. Experiments. Manuscript of a chapter of the book *"Algorithm Engineering"* that will appear as a volume of *Lecture Notes in Computer Science*, Springer-Verlag, January 2007.

[BHIK97] Endre Boros, Peter L. Hammer, Toshihide Ibaraki, and Kazuhiko Kawakami. Polynomial-time recognition of 2-monotonic positive Boolean functions given by an oracle. *SIAM Journal on Computing*, 26(1):93–109, 1997.

[BHL+05] Boualem Benatallah, Mohand-Said Hacid, Alain Léger, Christophe Rey, and Farouk Toumani. On automating web services discovery. *The VLDB Journal*, 14(1):84–96, 2005.

[BHP+06] Boualem Benatallah, Mohand-Said Hacid, Hye-Young Paik, Christophe Rey, and Farouk Toumani. Towards semantic-driven, flexible and scalable framework for peering and querying e-catalog communities. *Information Systems*, 31(4-5):266–294, 2006.

[BHRT03] Boualem Benatallah, Mohand-Said Hacid, Christophe Rey, and Farouk Toumani. Request rewriting-based web service discovery. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 2003.

[BI95a] Jan C. Bioch and Toshihide Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123(1):50–63, 1995.

[BI95b] Jan C. Bioch and Toshihide Ibaraki. Decompositions of positive self-dual Boolean functions. *Discrete Mathematics*, 140(1-3):23–46, 1995.

[BI95c] Jan C. Bioch and Toshihide Ibaraki. Generating and approximating nondominated coteries. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):905–914, 1995.

[BIM99] Jan C. Bioch, Toshihide Ibaraki, and Kazuhisa Makino. Minimum self-dual decompositions of positive dual-minor Boolean functions. *Discrete Applied Mathematics*, 96-97:307–326, 1999.

[Bio98] Jan C. Bioch. Dualization, decision lists and identification of monotone discrete functions. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):69–91, 1998.

[BMR03] James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA* [ICD03], pages 485–488.

[Bor94] Endre Boros. Dualization of aligned Boolean functions. Technical Report RRR 9-94, RUTCOR, Rutgers University, March 1994.

[Bor06] Konrad Borys. *On Generation of Cut Conjunctions, Minimal k-Connected Spanning Subgraphs, Minimal Connected and Spanning*

Subsets and Vertices. PhD thesis, Rutgers, The State University of New Jersey, October 2006.

[BS87] Paola Bertolazzi and Antonio Sassano. An $O(mn)$ algorithm for regular set-covering problems. *Theoretical Computer Science*, 54:237–247, 1987.

[BS88] Paola Bertolazzi and Antonio Sassano. A class of polynomially solvable set-covering problems. *SIAM Journal on Discrete Mathematics*, 1(3):306–316, 1988.

[BS05] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In Manuel V. Hermenegildo and Daniel Cabeza, editors, *Practical Aspects of Declarative Languages, 7th International Symposium, PADL 2005, Long Beach, CA, USA, January 10-11, 2005, Proceedings*, volume 3350 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2005.

[Cad92] Marco Cadoli. The complexity of model checking for circumscriptive formulae. *Information Processing Letters*, 44(3):113–118, 1992.

[CC07] Madalina Croitoru and Ernesto Compatangelo. Extending conceptual graphs for representing partial knowledge. In *7th IJCAI International Workshop on Nonmonotonic Reasoning, Action and Change, Hyderabad, India, January 7-8, 2007*, 2007.

[CCL03] Alain Casali, Rosine Cicchetti, and Lotfi Lakhal. Extracting semantics from data cubes using cube transversals and closures. In Lise Getoor, Ted E. Senator, Pedro Domingos, and Christos Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 69–78. ACM, 2003.

[CH07] Yves Crama and Peter L. Hammer. *Boolean Functions: Theory, Algorithms and Applications*. Cambridge University Press, 2007. (forthcoming).

[CKK02] Ricardo A. Collado, Alexander K. Kelmans, and Daniel Krasner. On convex polytopes in the plane “containing” and “avoiding” zero. Technical Report 33, DIMACS, July 2002.

[CM03] Nathalie Caspard and Bernard Monjardet. The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.

[Coo71] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings Third Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[Cra87] Yves Crama. Dualization of regular Boolean functions. *Discrete Applied Mathematics*, 16(1):79–85, 1987.

[Cro06] Madalina Croitoru. *Conceptual Graphs at Work: Efficient Reasoning and Applications*. PhD thesis, Department of Computing Science, University of Aberdeen, 2006.

[Dam06] Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[Dam07] Peter Damaschke. The union of minimal hitting sets: Parameterized combinatorial bounds and counting. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 2007.

[DCS06] Nicolas Durand, Bruno Crémilleux, and Einoshin Suzuki. Visualizing transactional data with multiple clusterings for knowledge discovery. In Floriana Esposito, Zbigniew W. Ras, Donato Malerba, and Giovanni Semeraro, editors, *Foundations of Intelligent Systems, 16th International Symposium, ISMIS 2006, Bari, Italy, September 27-29, 2006, Proceedings*, volume 4203 of *Lecture Notes in Computer Science*, pages 47–57. Springer, 2006.

[Dem80] János Demetrovics. On the equivalence of candidate keys with sperner systems. *Acta Cybernetica*, 4:247–252, 1980.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[DFH$^+$01] Vida Dujmovic, Michael R. Fellows, Michael T. Hallett, Matthew Kitching, Giuseppe Liotta, Catherine McCartin, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Matthew Suderman, Sue Whitesides, and David R. Wood. A fixed-parameter approach to two-layer planarization. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, volume 2265 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.

[DG96] James P. Delgrande and Arvind Gupta. The complexity of minimum partial truth assignments and implication in negation-free formulae. *Annals of Mathematics and Artificial Intelligence*, 18(1):51–67, 1996.

[DH08] Arnaud Durand and Miki Hermann. On the counting complexity of propositional circumscription. *Information Processing Letters*, 106(4):164–170, 2008.

[DHSW03] Jean-Guillaume Dumas, Frank Heckenbach, B. David Saunders, and Volkmar Welker. Computing simplicial homology based on efficient Smith Normal Form algorithms. In Michael Joswig and Nobuki Takayama, editors, *Algebra, Geometry, and Software Systems [outcome of a Dagstuhl seminar]*, pages 177–206. Springer, 2003.

[DK88] Elias Dahlhaus and Marek Karpinski. A fast parallel algorithm for computing all maximal cliques in a graph and the related problems (extended abstract). In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT 88, 1st Scandinavian Workshop on Algorithm Theory, Halmstad, Sweden, July 5-8, 1988, Proceedings*, volume 318 of *Lecture Notes in Computer Science*, pages 139–144. Springer, 1988.

[DL05] Guozhu Dong and Jinyan Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems*, 8(2):178–202, 2005.

[DMP99] Carlos Domingo, Nina Mishra, and Leonard Pitt. Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries. *Machine Learning*, 37(1):89–110, 1999.

[Dom97] Carlos Domingo. Polynominal time algorithms for some self-duality problems. In Gian Carlo Bongiovanni, Daniel P. Bovet, and Giuseppe Di Battista, editors, *Algorithms and Complexity, Third Italian Conference, CIAC '97, Rome, Italy, March 12-14, 1997, Proceedings*, volume 1203 of *Lecture Notes in Computer Science*, pages 171–180. Springer, 1997.

[DT87] János Demetrovics and Vu Duc Thi. Keys, antikeys and prime atrributes. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio computatorica*, 8:35–52, 1987.

[DT95] János Demetrovics and Vu Duc Thi. Some remarks on generating Armstrong and inferring functional dependencies relation. *Acta Cybernetica*, 12(2):167–180, 1995.

[DT99] János Demetrovics and Vu Duc Thi. Describing candidate keys by hypergraphs. *Computers and Artificial Intelligence*, 18(2):191–207, 1999.

[Duq91] Vincent Duquenne. The core of finite lattices. *Discrete Mathematics*, 88(2-3):133–147, 1991.

[EG91] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. Technical Report CD-TR 91/16, TU Wien, January 1991.

[EG95] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[EG02] Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and AI. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *Logics in Artificial Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings*, volume 2424 of *Lecture Notes in Computer Science*, pages 549–564. Springer, 2002.

[EGM03] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

[EHR08a] Khaled Elbassioni, Matthias Hagen, and Irman Rauf. A lower bound for the HBC transversal hypergraph generation. Manuscript, January 2008.

[EHR08b] Khaled M. Elbassioni, Matthias Hagen, and Imran Rauf. Some fixed-parameter tractable classes of hypergraph duality and related problems. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 2008.

[EIM99] Thomas Eiter, Toshihide Ibaraki, and Kazuhisa Makino. Bidual Horn functions and extensions. *Discrete Applied Mathematics*, 96-97:55–88, 1999.

[EIM02] Thomas Eiter, Toshihide Ibaraki, and Kazuhisa Makino. Recognition and dualization of disguised bidual Horn functions. *Information Processing Letters*, 82(6):283–291, 2002.

[Eit91] Thomas Eiter. *On Transversal Hypergraph Computation and Deciding Hypergraph Saturation.* PhD thesis, Technische Universität Wien, 1991.

[Eit94] Thomas Eiter. Exact transversal hypergraphs and application to Boolean μ-functions. *Journal of Symbolic Computation*, 17(3):215–225, 1994.

[EL03] Mark Steven Eker and Denis Patrick Lincoln. Modeling reaction pathways. United States Patent 20030154003, July 2003.

[Elb02a] Khaled M. Elbassioni. An algorithm for dualization in products of lattices and its applications. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 2002.

[Elb02b] Khaled M. Elbassioni. *Incremental Algorithms for Enumerating Extremal Solutions of Monotone Systems of Submodular Inequalities and Their Applications*. PhD thesis, Rutgers, The State University of New Jersey, 2002.

[Elb02c] Khaled M. Elbassioni. On dualization in products of forests. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2002.

[Elb06a] Khaled M. Elbassioni. Finding all minimal infrequent multi-dimensional intervals. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, volume 3887 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 2006.

[Elb06b] Khaled M. Elbassioni. On the complexity of the multiplication method for monotone CNF/DNF dualization. In Azar and Erlebach [AE06], pages 340–351.

[Elb08] Khaled M. Elbassioni. On the complexity of monotone dualization and generating minimal hypergraph transversals. *Discrete Applied Mathematics*, 156(11):2109–2123, 2008.

[EM07] Thomas Eiter and Kazuhisa Makino. On computing all abductive explanations from a propositional Horn theory. *Journal of the ACM*, 54(5), 2007.

[EMG08] Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.

[Epp05] David Eppstein. All maximal independent sets and dynamic dominance for sparse graphs. In *Proceedings of the Sixteenth Annual*

ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005, pages 451–459. SIAM, 2005.

[ER08] Khaled Elbassioni and Imran Rauf. Polynomial-time dualization of r-exact hypergraphs with applications in geometry. Manuscript, June 2008.

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[Fer05a] Henning Fernau. Parameterized algorithmics: A graph-theoretic approach. Habilitationsschrift, Universität Tübingen, April 2005.

[Fer05b] Henning Fernau. Two-layer planarization: Improving on parameterized algorithmics. *Journal of Graph Algorithms and Applications*, 9(2):205–238, 2005.

[FFJS04] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.

[FGBS96] John Franco, Giorgio Gallo, Hans Kleine Büning, and Ewald Speckenmeyer, editors. *Workshop on Satisfiability, Siena, Italy*, 1996.

[FK96] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

[FMP04] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. ABS: Adaptive borders search of frequent itemsets. In Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors, *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[FV04] Amir Fijany and Farrokh Vatan. New approaches for efficient solution of hitting set problem. Technical Report 03-2736, Jet Propulsion Laboratory, January 2004.

[FV05] Amir Fijany and Farrokh Vatan. New high performance algorithmic solution for diagnosis problem. Technical Report 04-3689, Jet Propulsion Laboratory, March 2005.

[FVB+02] Amir Fijany, Farrokh Vatan, Anthony Barrett, Ed Baroth, and Ryan Mackey. Novel model-based diagnosis approaches for advanced IVHM systems. Technical Report 02-0898, Jet Propulsion Laboratory, April 2002.

[FVB+03a] Amir Fijany, Farrokh Vatan, Anthony Barrett, Mark James, and Ryan Mackey. An advanced model-based diagnosis engine. Technical Report 03-0852, Jet Propulsion Laboratory, May 2003.

[FVB+03b] Amir Fijany, Farrokh Vatan, Anthony Barrett, Mark James, Colin Williams, and Ryan Mackey. Novel model-based diagnosis approaches for advanced IVHM systems. Technical Report 02-3188, Jet Propulsion Laboratory, March 2003.

[FVBM02] Amir Fijany, Farrokh Vatan, Anthony Barrett, and Ryan Mackey. New approaches for solving the diagnosis problem. Technical Report IPN Progress Report 42-149, The Interplanetary Network Progress Report, May 2002.

[Gar06] Gemma C. Garriga. *Formal methods for mining structured objects*. PhD thesis, Universitat Politècnica de Catalunya, April 2006.

[GB85] Hector Garcia-Molina and Daniel Barbará. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.

[GHM05] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF and isomorphism of monotone formulas. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 410–421. Springer, 2005.

[GHM08] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Information and Computation*, 206(6):760–775, 2008.

[GK97] Vladimir Gurvich and Leonid Khachiyan. On the frequency of the most frequently occurring variable in dual monotone DNFs. *Discrete Mathematics*, 169(1-3):245–248, 1997.

[GK99] Vladimir Gurvich and Leonid Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discrete Applied Mathematics*, 96-97:363–373, 1999.

[GK04a] Nicola Galesi and Oliver Kullmann. Polynomial time SAT decision, hypergraph transversals and the hermitian rank. In Holger H. Hoos and David G. Mitchell, editors, *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 2004.

[GK04b] Daya Ram Gaur and Ramesh Krishnamurti. Average case self-duality of monotone Boolean functions. In Ahmed Y. Tawfik and Scott D. Goodwin, editors, *Advances in Artificial Intelligence, 17th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2004, London, Ontario, Canada, May 17-19, 2004, Proceedings*, volume 3060 of *Lecture Notes in Computer Science*, pages 322–338. Springer, 2004.

[GK07] Daya Ram Gaur and Ramesh Krishnamurti. Self-duality of bounded monotone Boolean functions and related problems. *Discrete Applied Mathematics*, 2007. (to appear).

[GKM+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharm. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[GKMT97] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, and Hannu Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona*, pages 209–216. ACM Press, 1997.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

[GL90] Georg Gottlob and Leonid Libkin. Investigation on Armstrong relations, dependency inference, and excluded functional dependencies. *Acta Cybernetica*, 9(4):385–402, 1990.

[GLS02] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[GM08] Daya Ram Gaur and Kazuhisa Makino. On the fractional chromatic number of monotone self-dual Boolean functions. *Discrete Mathematics*, 2008.

[GMS97] Dimitrios Gunopulos, Heikki Mannila, and Sanjeev Saluja. Discovering all most specific sentences by randomized algorithms. In Foto N. Afrati and Phokion G. Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1997.

[God97] Parke Godfrey. Minimization in cooperative response to failing database queries. *International Journal of Cooperative Information Systems*, 6(2):95–149, 1997.

[Got04] Georg Gottlob. Hypergraph transversals. In Dietmar Seipel and Jose Maria Turull Torres, editors, *Foundations of Information and Knowledge Systems, Third International Symposium, FoIKS 2004, Wilhelminenburg Castle, Austria, February 17-20, 2004, Proceedings*, volume 2942 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 2004.

[GPS98] Goran Gogic, Christos H. Papadimitriou, and Martha Sideri. Incremental recompilation of knowledge. *Journal of Artificial Intelligence Research*, 8:23–37, 1998.

[GS83] Nathan Goodman and Oded Shmueli. Syntactic characterization of tree database schemas. *Journal of the ACM*, 30(4):767–786, 1983.

[GSS02] Georg Gottlob, Francesco Scarcello, and Martha Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1-2):55–86, 2002.

[GSW89] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artifial Intelligence*, 41(1):79–88, 1989.

[Gur75] Vladimir Gurvich. Solution of positional games in pure strategies. *USSR Comput. Math. and Math. Phys.*, 15(2):74–87, 1975.

[Gur88] Vladimir Gurvich. Equilibrium in pure strategies. *Soviet Mathematics Doklady*, 38(3):597–602, 1988.

[Hae98] Rolf Haenni. Generating diagnoses from conflict sets. In Diane J. Cook, editor, *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference, May 18-20, 1998, Sanibel Island, Florida, USA*, pages 420–424. AAAI Press, 1998.

[Hag06] Matthias Hagen. Logarithmic space instances of monotone normal form equivalence testing. Technical Report Reports on Computer Science 06-10, Institut für Informatik, Friedrich-Schiller-Universität Jena, June 2006.

[Hag07a] Matthias Hagen. Lower bounds for three algorithms for the transversal hypergraph generation. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science, 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers*, volume 4769 of *Lecture*

Notes in Computer Science, pages 316–327. Springer, 2007. Journal version accepted for publication at *Discrete Applied Mathematics*.

[Hag07b] Matthias Hagen. On the fixed-parameter tractability of the equivalence test of monotone normal forms. *Information Processing Letters*, 103(4):163–167, 2007.

[Hau08] Utz-Uwe Haus. Personal communication, June 2008.

[HBC07] Céline Hébert, Alain Bretto, and Bruno Crémilleux. A data mining formalization to improve hypergraph minimal transversal computation. *Fundamenta Informaticae*, 80(4):415–433, 2007.

[HCW06] Matthew Hamilton, Rhonda Chaytor, and Todd Wareham. The parameterized complexity of enumerating frequent itemsets. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 227–238. Springer, 2006.

[Héb07] Céline Hébert. *Extraction et usage de motifs minimaux en fouille de données, contribution au domaine des hypergraphes.* PhD thesis, Université de Caen, September 2007.

[HHM09] Matthias Hagen, Peter Horatschek, and Martin Mundhenk. Experimental comparison of the two Fredman-Khachiyan-algorithms. In *Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX 2009, New York, USA, January 3, 2009*, 2009. To appear.

[HKI08] Sven Hartmann and Gabriele Kern-Isberner, editors. *Foundations of Information and Knowledge Systems, 5th International Symposium, FoIKS 2008, Pisa, Italy, February 11-15, 2008, Proceedings*, volume 4932 of *Lecture Notes in Computer Science*. Springer, 2008.

[HKS08] Utz-Uwe Haus, Steffen Klamt, and Tamon Stephen. Computing knock-out strategies in metabolic networks. *Journal of Computational Biology*, 15(3):259–268, 2008.

[HLRT02a] Mohand-Saïd Hacid, Alain Léger, Christophe Rey, and Farouk Toumani. Computing concept covers: A preliminary report. In *Proceedings of the 2002 International Workshop on Description Logics (DL 2002). Toulouse, France, April 19-21, 2002*, 2002.

[HLRT02b] Mohand-Said Hacid, Alain Léger, Christophe Rey, and Farouk Toumani. Dynamic discovery of e-services. In Philippe Pucheral, editor, *18èmes Journées Bases de Données Avancées, BDA '02, 21-25 octobre 2002, Evry, Actes (Informal Proceedings)*, 2002.

[HLRT03] Mohand-Saïd Hacid, Alain Léger, Christophe Rey, and Farouk Toumani. An algorithm and a prototype for the dynamic discovery of e-services. Technical Report RR03/05, LIMOS, July 2003.

[HM03] Wen-Lian Hsu and Ross M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, 2003.

[HMP04] Haym Hirsh, Nina Mishra, and Leonard Pitt. Version spaces and the consistency problem. *Artificial Intelligence*, 156(2):115–138, 2004.

[HPP79] Peter L. Hammer, Uri N. Peled, and M. A. Pollatschek. An algorithm to dualize a regular switching function. *IEEE Transactions on Computers*, 28(3):238–243, 1979.

[HY05] Takashi Harada and Masafumi Yamashita. Transversal merge operation: A nondominated coterie construction method for distributed mutual exclusion. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):183–192, 2005.

[IB07] Marcin Imielinski and Calin Belta. On the computation of minimal cut sets in genome scale metabolic networks. In *Proceedings of American Control Conference, ACC'07, New York, NY, USA, July 9-13, 2007*, pages 1329–1334, 2007.

[ICD03] *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*. IEEE Computer Society, 2003.

[IK93] Toshihide Ibaraki and Tiko Kameda. A theory of coteries: Mutual exclusion in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):779–794, 1993.

[IKM99] Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Functional dependencies in Horn theories. *Artificial Intelligence*, 108(1-2):1–30, 1999.

[IKM03] Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Inferring minimal functional dependencies in Horn and q-Horn theories. *Annals of Mathematics and Artificial Intelligence*, 38(4):233–255, 2003.

[Jär00] Jouni Järvinen. Difference functions of dependence spaces. *Acta Cybernetica*, 14(4):619–630, 2000.

[JBPT06] Hélène Jaudoin, Boualem Benatallah, Jean-Marc Petit, and Farouk Toumani. Towards a scalable algorithm for query rewriting using views in presence of value constraints: Extended version. Technical report, LIMOS, Aubiere cedex, France, 2006.

[JD00] Stefan Janaqi and Pierre Duchet. Generator-preserving contractions and a min-max result on the graphs of planar polyominoes. *Ars Combinatoria*, 55, 2000.

[JN06] Philippe Janssen and Lhouari Nourine. Minimum implicational basis for $\wedge$-semidistributive lattices. *Information Processing Letters*, 99(5):199–202, 2006.

[Joh91] David S. Johnson. Open and closed problems in NP-completeness. Lecture given at the International School of Mathematics "G. Stampacchia": Summer School "NP-Completeness: The First 20 Years", Erice (Sicily), Italy, June 2027, 1991.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[KBB+06] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Enumerating spanning and connected subsets in graphs and matroids. In Azar and Erlebach [AE06], pages 444–455.

[KBB+08] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Generating cut conjunctions in graphs and related problems. *Algorithmica*, 51(3):239–263, 2008.

[KBE+05] Leonid G. Khachiyan, Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005.

[KBE+07] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Dual-bounded generating problems: Efficient and inefficient points for discrete probability distributions and sparse boxes for multidimensional data. *Theoretical Computer Science*, 379(3):361–376, 2007.

[KBEG06] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Applied Mathematics*, 154(16):2350–2372, 2006.

[KBEG07a] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. A global parallel algorithm for the hypergraph transversal problem. *Information Processing Letters*, 101(4):148–155, 2007.

[KBEG07b] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. On dualization of hypergraphs with bounded edge-intersections and other related classes of hypergraphs. *Theoretical Computer Science*, 382(2):139–150, 2007.

[KBEG08a] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. Generating all minimal integral solutions to AND-OR systems of monotone inequalities: Conjunctions are simpler than disjunctions. *Discrete Applied Mathematics*, 156(11):2020–2034, 2008.

[KBEG08b] Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. On enumerating minimal dicuts and strongly connected subgraphs. *Algorithmica*, 50(1):159–172, 2008.

[KG04] Steffen Klamt and Ernst Dieter Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.

[Kha95] Roni Khardon. Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995.

[Kha96] Roni Khardon. *Learning to be Competent.* PhD thesis, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1996.

[Kha00] Leonid Khachiyan. Transversal hypergraphs and families of polyhedral cones. In *International Conference on Advances in Convex Analysis and Global Optimization, Honoring the memory of C. Carathéodory (1873-1950), June 5-9, 2000, Pythagorion, Samos, Greece, Proceedings*, pages 6–8, 2000.

[KISI00] Yeon-Dae Kwon, Yasunori Ishihara, Shougo Shimizu, and Minoru Ito. Computational complexity of finding highly co-occurrent itemsets in market basket databases. *IEICE Transactions on Communications/Electronics/Information and Systems*, E00-A(1):1–12, 2000.

[KKS93] Henry A. Kautz, Michael J. Kearns, and Bart Selman. Reasoning with characteristic models. In *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993*, pages 34–39. The AAAI Press/The MIT Press, 1993.

[Kla06] Steffen Klamt. Generalized concept of minimal cut sets in biochemical networks. *Biosystems*, 83(2-3):233–247, 2006.

[KLM06] Oliver Kullmann, Inês Lynce, and João Marques-Silva. Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 22–35. Springer, 2006.

[KLM07] Naouel Karam, Serge Linckels, and Christoph Meinel. Semantic composition of lecture subparts for a personalized e-learning. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, volume 4519 of *Lecture Notes in Computer Science*, pages 716–728. Springer, 2007.

[KM95] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.

[KMR92] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[KMR99] Roni Khardon, Heikki Mannila, and Dan Roth. Reasoning with examples: Propositional formulae and database dependencies. *Acta Informatica*, 36(4):267–286, 1999.

[Koe05] Henning Koehler. Computing and representing the set of all canonical covers. Technical Report 2005/27, Department of Information Systems, Massey University, 2005.

[Koe08] Henning Koehler. Autonomous sets – A method for hypergraph decomposition with applications in database theory. In Hartmann and Kern-Isberner [HKI08], pages 78–95.

[KP04] Dogan Kesdogan and Lexi Pimenidis. The hitting set attack on anonymity protocols. In Jessica J. Fridrich, editor, *Information Hiding, 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers*, volume 3200 of *Lecture Notes in Computer Science*, pages 326–339. Springer, 2004.

[KPS93] Dimitris J. Kavvadias, Christos H. Papadimitriou, and Martha Sideri. On Horn envelopes and hypergraph transversals. In Kam-Wing Ng, Prabhakar Raghavan, N. V. Balasubramanian, and Francis Y. L. Chin, editors, *Algorithms and Computation, 4th International Symposium, ISAAC '93, Hong Kong, December 15-17, 1993, Proceedings*, volume 762 of *Lecture Notes in Computer Science*, pages 399–405. Springer, 1993.

[KR96] Roni Khardon and Dan Roth. Reasoning with models. *Artificial Intelligence*, 87(1-2):187–213, 1996.

[KS03a] Dimitris J. Kavvadias and Elias C. Stavropoulos. Checking monotone Boolean duality with limited nondterminism. Technical Report TR2003/07/02, University of Patras, July 2003.

[KS03b] Dimitris J. Kavvadias and Elias C. Stavropoulos. Monotone Boolean dualization is in $\mathsf{coNP}[\log^2 n]$. *Information Processing Letters*, 85(1):1–6, 2003.

[KS05] Dimitris J. Kavvadias and Elias C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. *Journal of Graph Algorithms and Applications*, 9(2):239–264, 2005.

[KSG07] Steffen Klamt, Julio Saez-Rodriguez, and Ernst Dieter Gilles. Structural and functional analysis of cellular networks with *CellNetAnalyzer*. *BMC Systems Biology*, 1(2), 2007.

[KSS00] Dimitris J. Kavvadias, Martha Sideri, and Elias C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3-4):157–162, 2000.

[KST93] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser Verlag, 1993.

[Law66] Eugene L. Lawler. Covering problems: Duality relations and a new method of solution. *SIAM Journal on Applied Mathematics*, 14(5):1115–1132, 1966.

[Li06] Dong Haoyuan Li. Mining sequential patterns with transversal hypergraph computation. Master's thesis, University of Montpellier II, June 2006.

[LJ02] Li Lin and Yunfei Jiang. Computing minimal hitting sets with genetic algorithm. In Markus Stumptner and Franz Wotawa, editors, *Proceedings of the 13th International Workshop on Principles of Diagnosis (DX-2002), May 2nd-4th, 2002, Semmering Austria*, pages 77–80, 2002.

[LJ03] Li Lin and Yunfei Jiang. The computation of hitting sets: Review and new algorithms. *Information Processing Letters*, 86(4):177–184, 2003.

[LLK80] Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[LLT07] Dong (Haoyuan) Li, Anne Laurent, and Maguelonne Teisseire. On transversal hypergraph enumeration in mining sequential patterns. In *11th International Database Engineering and Applications Symposium (IDEAS 2007), Banff, Canada, September 2007*, pages 303–307. IEEE Computer Society, 2007.

[Lov92] László Lovász. Combinatorial optimization: Some problems and trends. Technical Report 92-53, DIMACS, 1992.

[LPL00] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, volume 1777 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2000.

[LS08] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.

[Mak02] Kazuhisa Makino. A linear time algorithm for recognizing regular Boolean functions. *Journal of Algorithms*, 43(2):155–176, 2002.

[Mak03] Kazuhisa Makino. Efficient dualization of $O(\log n)$-term monotone disjunctive normal forms. *Discrete Applied Mathematics*, 126(2-3):305–312, 2003.

[Man02] Heikki Mannila. Local and global methods in data mining: Basic techniques and open problems. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2002.

[Man04] Thomas Manoukian. High performance algorithms for discovering emerging patterns. Master's thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia, March 2004.

[Man05] Heikki Mannila. Finding frequent patterns from data. Slides of lectures at International School “Eduardo R. Caianiello”, 10th Course and Workshop of the PASCAL Network of Excellence, *The Analysis of Patterns*, Centre “Ettore Majorana” for Scientific Culture, Erice, Italy, October 28 - November 6, 2005, 2005.

[McC04] Ross M. McConnell. A certifying algorithm for the consecutive-ones property. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 768–777. SIAM, 2004.

[MFP04] Fabien De Marchi, Frédéric Flouvat, and Jean-Marc Petit. Adaptive strategies for mining the positive border of interesting patterns: Application to inclusion dependencies in databases. In Jean-François Boulicaut, Luc De Raedt, and Heikki Mannila, editors, *Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, March 11-13, 2004, Revised Selected Papers*, volume 3848 of *Lecture Notes in Computer Science*, pages 81–101. Springer, 2004.

[MGM06] Bob Mungamuru, Hector Garcia-Molina, and Subhasish Mitra. How to safeguard your sensitive data. In *25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006),2-4 October 2006, Leeds, UK*, pages 199–211. IEEE Computer Society, 2006.

[MI96] Kazuhisa Makino and Toshihide Ibaraki. Interior and exterior functions of Boolean functions. *Discrete Applied Mathematics*, 69(3):209–231, 1996.

[MI97] Kazuhisa Makino and Toshihide Ibaraki. The maximum latency and identification of positive Boolean functions. *SIAM Journal on Computing*, 26(5):1363–1383, 1997.

[MI98] Kazuhisa Makino and Toshihide Ibaraki. A fast and simple algorithm for identifying 2-monotonic positive Boolean functions. *Journal of Algorithms*, 26(2):291–305, 1998.

[MI99] Kazuhisa Makino and Toshihide Ibaraki. Inner-core and outer-core functions of partially defined Boolean functions. *Discrete Applied Mathematics*, 96-97:443–460, 1999.

[Mis02] Nina Mishra. Maximal frequent sets and the monotone CNF/DNF problem. Lecture notes, Stanford, 2002.

[MK01] Kazuhisa Makino and Tiko Kameda. Transformations on regular non-dominated coteries and their applications. *SIAM Journal on Discrete Mathematics*, 14(3):381–407, 2001.

[MOI03] Kazuhisa Makino, Hirotaka Ono, and Toshihide Ibaraki. Interior and exterior functions of positive Boolean functions. *Discrete Applied Mathematics*, 130(3):417–436, 2003.

[MP97] Nina Mishra and Leonard Pitt. Generating all maximal independent sets of bounded-degree hypergraphs. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory (COLT 1997), July 6-9, 1997, Nashville, Tennessee, USA*, pages 211–217. ACM, 1997.

[MP03] Fabien De Marchi and Jean-Marc Petit. Zigzag: a new algorithm for mining large inclusion dependencies in database. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA* [ICD03], pages 27–34.

[MR86] Heikki Mannila and Kari-Jouko Räihä. Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33(2):126–141, 1986.

[MR87] Heikki Mannila and Kari-Jouko Räihä. Dependency inference. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 155–158. Morgan Kaufmann, 1987.

[MR92a] Heikki Mannila and Kari-Jouko Räihä. *Design of Relational Databases*. Addison Wesley, 1992.

[MR92b] Heikki Mannila and Kari-Jouko Räihä. On the complexity of inferring functional dependencies. *Discrete Applied Mathematics*, 40(2):237–243, 1992.

[MR94] Heikki Mannila and Kari-Jouko Räihä. Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering*, 12(1):83–99, 1994.

[MS94] Robert J. McEliece and Kumar N. Sivarajan. Performance limits for channelized cellular telephone systems. *IEEE Transactions on Information Theory*, 40(1):21–34, 1994.

[MT96a] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations (Extended abstract). In *Proceedings* *of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 189–194, 1996.

[MT96b] Heikki Mannila and Hannu Toivonen. On an algorithm for finding all interesting sentences. In Robert Trappl, editor, *Proceedings of the 13th European Meeting on Cybernetics and Systems Research (EMCSR'96), Vienna*, pages 973–978. Austrian Society for Cybernetic Studies, April 1996.

[MT97] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[MT98] Heikki Mannila and Hannu Toivonen. *Knowledge Discovery in Databases: The Search for Frequent Patterns.* 1998.

[Mun89] Daniele Mundici. Functions computed by monotone Boolean formulas with no repeated variables. *Theoretical Computer Science*, 66(1):113–114, 1989.

[Mur71] Saburo Muroga. *Threshold Logic and Its Applications.* Wiley-Interscience, New York, 1971.

[Nic02] Tara Nicholson. Diagnostic hypergraphs: when the problem is the solution. In *1st North American Summer School for Logic Language and Information. Stanford University, June 2002*, pages 67–74, 2002.

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pap97] Christos H. Papadimitriou. NP-completeness: A retrospective. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, volume 1256 of *Lecture Notes in Computer Science*, pages 2–6. Springer, 1997.

[PB88] J. Scott Provan and Michael O. Ball. Efficient recognition of matroid and 2-monotonic systems. In Richard D. Ringeisen and Fred S. Roberts, editors, *Applications of Discrete Mathematics, Proceedings of the third Conference on Discrete Mathematics, Clemson University, Clemson, South Carolina, May 14 - 16, 1986*, volume 33 of *Proceedings in applied mathematics*, pages 122–134. SIAM, 1988.

[PCPO02] Andreas Polyméris, Ricardo Contreras, María Angélica Pinninghoff, and Esteban Osses. Response-ability and its complexity. In *Second International Workshop Computational Models of Scientific Reasoning and Applications (CMSRA), Monte Carlo Resort, Las Vegas, Nevada, USA, June 24-27, 2002, On-Line Proceedings*, 2002.

[PDA93a] Luís Moniz Pereira, Carlos Viegas Damásio, and José Júlio Alferes. Diagnosis and debugging as contradiction removal. In *Logic Programming and Non-monotonic Reasoning, LPNMR, Proceedings of the Second International Workshop, Lisbon, Portugal, June 1993*, pages 316–330, 1993.

[PDA93b] Luís Moniz Pereira, Carlos Viegas Damásio, and José Júlio Alferes. Diagnosis and debugging as contradiction removal in logic programs. In Miguel Filgueiras and Luís Damas, editors, *Progress in Artificial Intelligence, 6th Portuguese Conference on Artificial Intelligence, EPIA '93, Porto, Portugal, October 6-8, 1993, Proceedings*, volume 727 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 1993.

[Pfa06] John L. Pfaltz. Using concept lattices to uncover causal dependencies in software. In Rokia Missaoui and Jürg Schmid, editors, *Formal Concept Analysis, 4th International Conference, ICFCA 2006, Dresden, Germany, February 13-17, 2006, Proceedings*, volume 3874 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2006.

[Pfa07] John L. Pfaltz. Establishing logical rules from empirical data. In *Proceedings of 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), Volume 2.*, pages 202–209, 2007.

[Pfe02] Marc E. Pfetsch. *The Maximum Feasible Subsystem Problem and Vertex-Facet Incidences of Polyhedra*. PhD thesis, Technische Universität Berlin, October 2002.

[Pit02] Toniann Pitassi. Propositional proof complexity, lecture 1. Lecture notes, September 2002.

[Pop04] Viara Nikolaeva Popova. *Knowledge Discovery and Monotonicity*. PhD thesis, Erasmus University Rotterdam, 2004.

[PS85] Uri N. Peled and Bruno Simeone. Polynomial time algorithms for regular set-covering and threshold problems. *Discrete Applied Mathematics*, 12(1):57–69, 1985.

[PS94] Uri N. Peled and Bruno Simeone. An $O(nm)$-time algorithm for computing the dual of a regular Boolean function. *Discrete Applied Mathematics*, 49(1-3):309–323, 1994.

[PT02] John L. Pfaltz and Christopher M. Taylor. Scientific discovery through iterative transformations of concept lattices. In *Proceedings Workshop on Discrete Mathematics and Data Mining, 2nd SIAM International Conference on Data Mining, April 11-13, Arlington, Virginia, USA, 2002*, pages 65–74, 2002.

[Qui53] Willard van Orman Quine. Two theorems about truth functions. *Boletin de la Sociedad Matemática Mexicana*, 10:64–70, 1953.

[RB03] Kotagiri Ramamohanarao and James Bailey. Discovery of emerging patterns and their use in classification. In Tamás D. Gedeon and Lance Chun Che Fung, editors, *AI 2003: Advances in Artificial Intelligence, 16th Australian Conference on Artificial Intelligence, Perth, Australia, December 3-5, 2003, Proceedings*, volume 2903 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003.

[RC06] François Rioult and Bruno Crémilleux. Mining correct properties in incomplete databases. In Saso Dzeroski and Jan Struyf, editors, *Knowledge Discovery in Inductive Databases, 5th International Workshop, KDID 2006, Berlin, Germany, September 18, 2006, Revised Selected and Invited Papers*, volume 4747 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2006.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Rei03] Steffen Reith. On the complexity of some equivalence problems for propositional calculi. In Branislav Rovan and Peter Vojtás, editors, *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 632–641. Springer, 2003.

[RF07] Kotagiri Ramamohanarao and Hongjian Fan. Patterns based classifiers. *World Wide Web*, 10(1):71–83, 2007.

[Rym92] Ron Rymon. Search through systematic set enumeration. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992*, pages 539–550. Morgan Kaufmann, 1992.

[Rym94a] Ron Rymon. An SE-tree-based prime implicant generation algorithm. *Annals of Mathematics and Artificial Intelligence*, 11(1-4):351–366, 1994.

[Rym94b] Ron Rymon. On kernel rules and prime implicants. In *Proceedings of the 12th National Conference on Artificial Intelligence, Volume 1.*

Seattle, WA, USA, July 31 - August 4, 1994, pages 181–186. AAAI Press, 1994.

[Sey74] Paul D. Seymour. On the two-colouring of hypergraphs. *The quarterly journal of mathematics Oxford*, 25(3):303–312, 1974.

[Sir04] Evren Sirin. Automated composition of web services using AI planning techniques. Master's thesis, Department of Computer Science, University of Maryland, 2004.

[SK90] Bart Selman and Henry A. Kautz. Model-preference default theories. *Artificial Intelligence*, 45(3):287–322, 1990.

[SK96] Bart Selman and Henry A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[SKA01] Ilya Shmulevich, Aleksey D. Korshunov, and Jaakko Astola. Almost all monotone Boolean functions are polynomially learnable using membership queries. *Information Processing Letters*, 79(5):211–213, 2001.

[SKU06] Ken Satoh, Ken Kaneiwa, and Takeaki Uno. Contradiction finding and minimal recovery for UML class diagrams. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006), 18-22 September 2006, Tokyo, Japan*, pages 277–280. IEEE Computer Society, 2006.

[SS98] Saswati Sarkar and Kumar N. Sivarajan. Hypergraph models for cellular mobile communication systems. *IEEE Transactions on Vehicular Technology*, 47(2):460–471, 1998.

[Sta01] Elias C. Stavropoulos. Recent complexity results on generation problems. Poster, 1st International Seminar on Mathematics of Computers and Decision Making, University of Patras, Greece, May 25–26, 2001, May 2001.

[STT98] Robert H. Sloan, Ken Takata, and György Turán. On frequent sets of Boolean matrices. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):193–209, 1998.

[SU03] Ken Satoh and Takeaki Uno. Enumerating maximal frequent sets using irredundant dualization. In Gunter Grieser, Yuzuru Tanaka, and Akihiro Yamamoto, editors, *Discovery Science, 6th International Conference, DS 2003, Sapporo, Japan, October 17-19,2003, Proceedings*, volume 2843 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 2003.

[SU05] Ken Satoh and Takeaki Uno. Enumerating minimally revised specifications using dualization. In Takashi Washio, Akito Sakurai, Katsuto Nakajima, Hideaki Takeda, Satoshi Tojo, and Makoto Yokoo, editors, *New Frontiers in Artificial Intelligence, Joint JSAI 2005 Workshop Post-Proceedings*, volume 4012 of *Lecture Notes in Computer Science*, pages 182–189. Springer, 2005.

[SU06] Ken Satoh and Takeaki Uno. Enumerating minimal explanations by minimal hitting set computation. In Jérôme Lang, Fangzhen Lin, and Ju Wang, editors, *Knowledge Science, Engineering and Management, First International Conference, KSEM 2006, Guilin, China, August 5-8, 2006, Proceedings*, volume 4092 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2006.

[Tak07] Ken Takata. A worst-case analysis of the sequential method to list the minimal hitting sets of a hypergraph. *SIAM Journal on Discrete Mathematics*, 21(4):936–946, 2007.

[Tam00] Hisao Tamaki. Space-efficient enumeration of minimal transversals of a hypergraph. In *Proceedings 75th SIGAL Conference of the Information Processing Society of Japan (IPSJ-AL 75)*, pages 29–36, 2000. Extended paper available from the author.

[TB06] Roger Ming Hieng Ting and James Bailey. Mining minimal contrast subgraph patterns. In Joydeep Ghosh, Diane Lambert, David B. Skillicorn, and Jaideep Srivastava, editors, *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*. SIAM, 2006.

[Thi86] Vu Duc Thi. Minimal keys and antikeys. *Acta Cybernetica*, 7(4):361–371, 1986.

[TIAS77] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[Toi96a] Hannu Toivonen. *Discovery of frequent patterns in large data collections*. PhD thesis, University of Helsinki, Department of Computer Science, 1996.

[Toi96b] Hannu Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 134–145. Morgan Kaufmann, 1996.

[Tom88] Anthony Tomasic. View update translation via deduction and annotation. In Marc Gyssens, Jan Paredaens, and Dirk Van Gucht, editors, *ICDT'88, 2nd International Conference on Database Theory, Bruges, Belgium, August 31 - September 2, 1988, Proceedings*, volume 326 of *Lecture Notes in Computer Science*, pages 338–352. Springer, 1988.

[Tor01] Vetle Torvik. *Data Mining and Knowledge Discovery: A Guided Approach Based on Monotone Boolean Functions*. PhD thesis, Louisiana State University, Baton Rouge, LA, October 2001.

[Tri08] Thu Trinh. Using transversals for discovering XML functional dependencies. In Hartmann and Kern-Isberner [HKI08], pages 199–218.

[TS05] Duc Vu Thi and Hoang Nguyen Son. On the dense families in the relational datamodel. *ASEAN Journal on Science and Technology for Development*, 22(3):241–249, 2005.

[TT01] Vetle I. Torvik and Evangelos Triantaphyllou. Inference of monotone Boolean functions. In Chris A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, volume 2, pages 472–480. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

[TT02] Vetle I. Torvik and Evangelos Triantaphyllou. Minimizing the average query complexity of learning monotone Boolean functions. *INFORMS Journal on Computing*, 14(2):144–174, 2002.

[TY84] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[TY07] Stéphan Thomassé and Anders Yeo. Total domination of graphs and small transversals of hypergraphs. *Combinatorica*, 27(4):473–487, 2007.

[Uno02] Takeaki Uno. A practical fast algorithm for enumerating minimal set coverings. In *Proceedings 83rd SIGAL Conference of the Information Processing Society of Japan, Tokyo, 15 March, 2002*, pages 9–16, 2002. In Japanese.

[US03] Takeaki Uno and Ken Satoh. Detailed description of an algorithm for enumeration of maximal frequent sets with irredundant dualization. In Bart Goethals and Mohammed Javeed Zaki, editors, *FIMI'03, Frequent Itemset Mining Implementations, Proceedings of the ICDM*

2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[Vin99a] Staal A. Vinterbo. A genetic algorithm for a family of set cover problems. Technical report, Norwegian University of Science and Technology, 1999.

[Vin99b] Staal A. Vinterbo. *Predictive Models in Medicine: Some Methods for Construction and Adaption*. PhD thesis, Norwegian University of Science and Technology, December 1999.

[VO00a] Staal A. Vinterbo and Lucila Ohno-Machado. A genetic algorithm approach to multi-disorder diagnosis. *Artificial Intelligence in Medicine*, 18(2):117–132, 2000.

[VØ00b] Staal A. Vinterbo and Aleksander Øhrn. Minimal approximate hitting sets and rule templates. *International Journal of Approximate Reasoning*, 25(2):123–143, 2000.

[WGR01] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances – Extended abstract. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings*, volume 2114 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2001.

[Wil00] Marcel Wild. Optimal implicational bases for finite modular lattices. *Quaestiones Mathematicae*, 23(2):153–161, 2000.

[Win62] Robert O. Winder. *Threshold Logic*. PhD thesis, Mathematics Department, Princeton University, 1962.

[Wot01] Franz Wotawa. A variant of Reiter's hitting-set algorithm. *Information Processing Letters*, 79(1):45–51, 2001.

[Zan02] Bruno Zanuttini. Approximation of relations by propositional formulas: Complexity and semantics. In Sven Koenig and Robert C. Holte, editors, *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, volume 2371 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 2002.

[ZB06] Zhou Zhu and James Bailey. Fast discovery of interesting collections of web services. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006), 18-22 December 2006, Hong Kong, China*, pages 152–160. IEEE Computer Society, 2006.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena, den 18. Juli 2008

Matthias Hagen

Lebenslauf

Persönliche Daten

Name	Matthias Hagen
Geburt	29. Oktober 1979 in Ilmenau

Bildungsweg

1994 – 1998	Gymnasium „Goetheschule“, Ilmenau, mathematisch-naturwissenschaftliche Spezialklasse
Juli 1998	Abitur
1999 – 2004	Studium der Informatik, Friedrich-Schiller-Universität (FSU) Jena
Juni 2004	Diplom

Akademische Laufbahn

seit Oktober 2004	Doktorand am Institut für Informatik der FSU
2004 – 2006	Landesgraduiertenstipendiat des Freistaates Thüringen
2004 – 2007	wissenschaftliche Hilfskraft an der FSU
Januar 2007 – März 2007	wissenschaftliche Hilfskraft an der Eberhard-Karls-Universität Tübingen
Februar 2007 – Dezember 2008	wissenschaftlicher Mitarbeiter an der Universität Kassel
September 2007 – Oktober 2008	wissenschaftlicher Mitarbeiter an der FSU
seit November 2008	wissenschaftlicher Mitarbeiter an der Bauhaus-Universität Weimar